THE POWER TO KNOW.

SAS® 9.3 In-Database Products
User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 In-Database Products: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.3 In-Database Products: User's Guide

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

1st electronic book, July 2011

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

Contents

What's New in SAS 9.3 In-Database Products: User's Guide . . . . . *vii*
Recommended Reading . . . . . *xi*

PART 1 Introduction 1

Chapter 1 • SAS In-Database Processing **3**
Introduction to SAS In-Database Processing 3
Deployed Components for In-Database Processing 4
Where to Go from Here 6

PART 2 SAS Scoring Accelerator 7

Chapter 2 • Introduction to the SAS Scoring Accelerator **9**
Overview of the SAS Scoring Accelerator 9
How It Works 10
Special Characters in Directory Names 11

Chapter 3 • Exporting the Scoring Model Files from SAS Enterprise Miner **13**
Overview of the Score Code Export Node 13
Comparing the Score Code Export Node with Registering Models on the SAS Metadata Server 14
Using the Score Code Export Node 14
Output Created by the Score Code Export Node 16

Chapter 4 • SAS Scoring Accelerator for Aster nCluster **25**
Publishing Scoring Model Files in Aster nCluster 25
Running the %INDAC_PUBLISH_MODEL Macro 26
Scoring Files and Functions inside the Aster nCluster Database 31

Chapter 5 • SAS Scoring Accelerator for DB2 under UNIX **35**
Publishing Scoring Model Files in DB2 35
Running the %INDB2_PUBLISH_MODEL Macro 36
Scoring Functions inside the DB2 Database 43

Chapter 6 • SAS Scoring Accelerator for Greenplum **47**
Publishing Scoring Model Files in Greenplum 47
Running the %INDGP_PUBLISH_MODEL Macro 48
Greenplum Permissions 53
Scoring Functions inside the Greenplum Database 53

Chapter 7 • SAS Scoring Accelerator for Netezza **57**
Publishing Scoring Model Files in Netezza 57
Running the %INDNZ_PUBLISH_MODEL Macro 58
Scoring Functions inside the Netezza Data Warehouse 64

Chapter 8 • SAS Scoring Accelerator for Teradata **69**
Publishing Scoring Model Files in Teradata 69
Running the %INDTD_PUBLISH_MODEL Macro 70
Scoring Functions inside the Teradata EDW 75

Chapter 9 • SAS Scoring Accelerator and SAS Model Manager **79**
Using the SAS Scoring Accelerator with SAS Model Manager 79

PART 3 Format Publishing and the SAS_PUT() Function 81

Chapter 10 • Deploying and Using SAS Formats inside the Database **83**
Using SAS Formats and the SAS_PUT() Function 83
How It Works 84
Special Characters in Directory Names 86
Considerations and Limitations with User-Defined Formats 87
Tips for Using the Format Publishing Macros 88
Tips for Using the SAS_PUT() Function 88
Determining Format Publish Dates 88

Chapter 11 • Deploying and Using SAS Formats in DB2 under UNIX **91**
User-Defined Formats in the DB2 Database 91
Publishing SAS Formats in DB2 91
Using the SAS_PUT() Function in the DB2 Database 97
DB2 Permissions 100

Chapter 12 • Deploying and Using SAS Formats in Netezza **103**
User-Defined Formats in the Netezza Data Warehouse 103
Publishing SAS Formats in Netezza 104
Using the SAS_PUT() Function in the Netezza Data Warehouse 109
Netezza Permissions 112

Chapter 13 • Deploying and Using SAS Formats in Teradata **113**
User-Defined Formats in the Teradata EDW 113
Publishing SAS Formats in Teradata 114
Data Types and the SAS_PUT() Function 118
Using the SAS_PUT() Function in the Teradata EDW 120
Teradata Permissions 123

PART 4 In-Database Procedures 125

Chapter 14 • Running SAS Procedures inside the Database **127**
Introduction to In-Database Procedures 127
Running In-Database Procedures 128
In-Database Procedures in DB2 under UNIX and PC Hosts, Netezza, and Oracle . . . 129
In-Database Procedures in Teradata 129
In-Database Procedure Considerations and Limitations 130
Using the MSGLEVEL Option to Control Messaging 133

PART 5 System Options Reference 135

Chapter 15 • System Options That Affect In-Database Processing **137**
Dictionary 137

PART 6 Appendixes 145

Appendix 1 • Scoring File Examples for Aster nCluster **147**
Example of a .ds2 Scoring File 147
Example of an Input and Output Variables Scoring File 167
Example of a User-Defined Formats Scoring File 174

Index **181**

What's New in SAS 9.3 In-Database Products: User's Guide

Overview

Starting in SAS 9.3, the user documentation for format publishing, in-database procedures, and the Scoring Accelerator have been combined into this document, *SAS 9.3 In-Database Products: User's Guide*.

Support for Teradata V13, Netezza V6.0, and Aster *n*Cluster V6 has been added.

Some Base SAS procedures have been enhanced for in-database processing inside Netezza.

Consolidation of In-Database User Documentation

Starting in SAS 9.3, the user documentation for these in-database technologies has been combined into this document, *SAS 9.3 In-Database Products: User's Guide*:

- Format publishing and the SAS_PUT() function were previously documented in *SAS/ACCESS for Relational Databases: Reference.*
- In-database procedures was previously documented in *SAS/ACCESS for Relational Databases: Reference.*

  Note: Each in-database procedure has its own specific considerations and limitations. For more information, see the documentation for the procedure.
- Scoring Accelerator was previously documented in the *SAS Scoring Accelerator: User's Guide* for each database.

The in-database installation and configuration documentation can be found in *SAS In-Database Products: Administrator's Guide*.

Compiled Publishing Macros

In SAS 9.3, all publishing macros are compiled for better security. There is no change in the way you run the publishing macros.

Additional Alias for INDCONN Macro Password Argument

You can now use PASS= for the password argument in the INDCONN macro variable.

In-Database Procedures

There are several enhancements to in-database procedures:

- You can use the SAS In-Database technology to run some Base SAS procedures inside Netezza.
- In BY-group processing, the NOTSORTED option is now ignored because the data is always returned in sorted order. Previously, the NOTSORTED option was not supported.

DB2 Changes

- Format publishing is now supported. Format publishing enables you to execute SAS PUT function calls inside the database. You can reference most of the formats that SAS supplies and the custom formats that you create with PROC FORMAT.

Netezza Changes

The following changes have been made for Netezza:

- Support for Netezza V6.0 has been added.
- Support for Netezza Performance Server (NPS) has been dropped.
- You can now run Netezza format and model publishing macros in fenced mode and in unfenced mode. Fenced mode means that the format and scoring functions that are published are isolated in a separate process in the Netezza database when they are

invoked, and an error does not cause the database to stop. When the format or scoring functions are ready for production, you can run the macro to publish the functions in unfenced mode.

Aster *n*Cluster Changes

The following changes have been made for Aster *n*Cluster:

- Support for Aster *n*Cluster V6 has been added.
- If you use Aster *n*Cluster V6, you can specify a schema where the scoring model files are published. You specify this schema in the INDCONN macro variable, and you can use the MODEL_SCHEMA parameter in the SAS_SCORE() function when you execute the scoring model.

Recommended Reading

Here is the recommended reading list for this title:

- *Base SAS Procedures User's Guide*
- *Base SAS Procedures Guide: Statistical Procedures*
- *Getting Started with SAS Enterprise Miner*
- *SAS/ACCESS for Relational Databases: Reference*
- *SAS Analytics Accelerator for Teradata: Guide*
- *SAS In-Database Products: Administrators Guide*
- *SAS Model Manager: User's Guide*
- *SAS /STAT User's Guide*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Part 1

Introduction

Chapter 1
SAS In-Database Processing *3*

Chapter 1

SAS In-Database Processing

Introduction to SAS In-Database Processing 3

Deployed Components for In-Database Processing 4
Deployed Components for Aster nCluster 4
Deployed Components for DB2 5
Deployed Components for Greenplum 5
Deployed Components for Netezza 5
Deployed Components for Teradata 6

Where to Go from Here 6

Introduction to SAS In-Database Processing

When using conventional processing to access data inside a database management system (DBMS), SAS asks the SAS/ACCESS engine for all rows of the table being processed. The SAS/ACCESS engine generates an SQL SELECT * statement that is passed to the DBMS. That SELECT statement fetches all the rows in the table, and the SAS/ACCESS engine returns them to SAS. As the number of rows in the table grows over time, network latency grows because the amount of data that is fetched from the DBMS to SAS increases.

SAS In-Database processing integrates SAS solutions, SAS analytic processes, and third-party database management systems. Using SAS In-Database processing, you can run scoring models, some SAS procedures, and formatted SQL queries inside the database. The following table lists the SAS products needed to use these features.

In-Database Feature	Software Required	DBMSs Supported
format publishing and the SAS_PUT() function	• Base SAS • SAS/ACCESS Interface to the DBMS	DB2 under UNIX Netezza Teradata
scoring models	• Base SAS • SAS/ACCESS Interface to the DBMS • SAS Scoring Accelerator • SAS Model Manager (optional)	Aster *n*Cluster DB2 under UNIX Greenplum Netezza Teradata

In-Database Feature	Software Required	DBMSs Supported
Base SAS procedures: FREQ RANK REPORT SORT SUMMARY/MEANS TABULATE	• Base SAS • SAS/ACCESS Interface to the DBMS	DB2 under UNIX and PC Hosts Oracle Netezza Teradata
SAS/STAT procedures: CORR CANCORR DMDB DMINE DMREG FACTOR PRINCOMP REG SCORE TIMESERIES VARCLUS	• Base SAS (for CORR) • SAS/ACCESS Interface to Teradata • SAS/STAT (for CANCORR, FACTOR,PRINCOMP, REG, SCORE, VARCLUS) • SAS/ETS (for TIMESERIES) • SAS Enterprise Miner (for DMDB, DMINE, DMREG) • SAS Analytics Accelerator	Teradata

Deployed Components for In-Database Processing

Deployed Components for Aster nCluster

Components that are deployed to Aster *n*Cluster for in-database processing are contained in a self-extracting TAR file (tkindbsrv-9.3-1_lax.sh). The TAR file is located in the ***SAS-install-directory*/SASTKInDatabaseServer/9.3/AsternClusteronLinuxx64/** directory.

The SAS Embedded Process is the component that is deployed in Aster *n*Cluster. The SAS Embedded Process contains run-time libraries, and other software that is installed on your Aster *n*Cluster system. The SAS scoring files created in Aster *n*Cluster accesses the routines within the run-time libraries.

In particular, the SAS System libraries and the SAS_SCORE() SQL/MR function are installed. The SAS Scoring Accelerator for Aster *n*Cluster uses these libraries and SQL/MR function to run scoring models inside the database.

For more information about these components, see the installation and configuration instructions in the *SAS In-Database Products: Administrator's Guide*.

Deployed Components for DB2

Components that are deployed to DB2 for in-database processing are contained in a self-extracting TAR file (acceldb2fmt-2.1-1_*.sh). The TAR file is located in the ***SAS-install-directory*/SASFormatsLibraryforDB2/2.1/DB2on<AIX | Linux64>/** directory.

The following components are deployed:

- The SAS formats library. The library contains many formats that are available in Base SAS.

  After you install the SAS formats library, the SAS scoring model functions and the SAS_PUT() function created in DB2 can access the routines within its run-time library.

- The binary files for the SAS_COMPILEUDF function.

  The%INDB2_PUBLISH_COMPILEUDF macro registers the SAS_COMPILEUDF function in the SASLIB schema of the DB2 database. The SAS_COMPILEUDF function compiles the scoring model source files in the DB2 database, links to the SAS formats library, and then copies the new object files to a specified location.

- The binary files for the SAS_DELETEUDF function.

  The %INDB2_PUBLISH_DELETEUDF macro registers the SAS_DELETEUDF function in the SASLIB schema of the DB2 database. The SAS_DELETEUDF function removes existing object files.

For more information about these components, see the installation and configuration instructions in the *SAS In-Database Products: Administrator's Guide*.

Deployed Components for Greenplum

Components that are deployed to Greenplum for in-database processing are contained in a self-extracting TAR file (accelgplmfmt-2.1-1_lax.sh). The TAR file is located in the ***SAS-install-directory*/SASFormatsLibraryforGreenplum/2.1/GreenplumonLinux64/** directory.

The following components are deployed:

- The SAS formats library. The library contains many formats that are available in Base SAS.

  After you install the SAS formats library, the SAS scoring model functions created in Greenplum can access the routines within its run-time library.

- The binary files for the SAS_COMPILEUDF function and other utility functions.

  The %INDGP_PUBLISH_COMPILEUDF macro registers the SAS_COMPILEUDF function and other utility functions in the database. The utility functions are called by the %INDGP_PUBLISH_MODEL scoring publishing macro.

For more information about these components, see the installation and configuration instructions in the *SAS In-Database Products: Administrator's Guide*.

Deployed Components for Netezza

Components that are deployed to Netezza for in-database processing are contained in a self-extracting TAR file (accelnetzfmt-2.1-1_lax.sh). The TAR file is located in the

***SAS-install-directory*/SASFormatsLibraryforNetezza/2.1/Netezza32bitTwinFin/** directory.

The following components are deployed:

- The SAS formats library. The library contains many formats that are available in Base SAS.

  The SAS formats library is published to the database as an object.

  After the %INDNZ_PUBLISH_JAZLIB macro publishes and registers the SAS formats library, the SAS scoring model functions and the SAS_PUT() function created in Netezza can access the routines within its run-time library.

- The binary files for SAS_COMPILEUDF and other utility functions.

  The %INDNZ_PUBLISH_COMPILEUDFmacro creates the SAS_COMPILEUDF, SAS_DictionaryUDF, and SAS_HextToText functions that are needed to facilitate the publishing of the scoring models, the SAS_PUT() function, and user-defined formats.

The %INDNZ_PUBLISH_JAZLIB and %INDNZ_PUBLISH_COMPILEUDF macros are typically run by your system or database administrator.

For more information, see the *SAS In-Database Products: Administrator's Guide*.

Deployed Components for Teradata

Components that are deployed to Teradata for in-database processing are contained in an RPM file (accelterfmt-2.1-1.x86_64.rpm). The TAR file is located in the ***SAS-install-directory*/SASFormatsLibraryforTeradata/2.1/TeradataonLinux/** directory.

The component that is deployed is the SAS formats library. The SAS formats library contains many of the formats that are available in Base SAS. After you install the SAS formats library, the SAS scoring model functions and the SAS_PUT() function created in Teradata can access the routines within its run-time library.

For more information about installing and configuring these components, see the *SAS In-Database Products: Administrator's Guide*.

Where to Go from Here

After the in-database deployment packages have been installed and configured, see the following topics to use in-database processing inside your database:

In-Database Processing Task	Documentation
Run scoring models	Chapter 2, “Introduction to the SAS Scoring Accelerator,” on page 9
Publish user-defined formats and use the SAS_PUT() function	Chapter 10, “Deploying and Using SAS Formats inside the Database,” on page 83
Run procedures inside the database	Chapter 14, “Running SAS Procedures inside the Database,” on page 127

Part 2

SAS Scoring Accelerator

Chapter 2
Introduction to the SAS Scoring Accelerator *9*

Chapter 3
Exporting the Scoring Model Files from SAS Enterprise Miner *13*

Chapter 4
SAS Scoring Accelerator for Aster nCluster *25*

Chapter 5
SAS Scoring Accelerator for DB2 under UNIX *35*

Chapter 6
SAS Scoring Accelerator for Greenplum *47*

Chapter 7
SAS Scoring Accelerator for Netezza *57*

Chapter 8
SAS Scoring Accelerator for Teradata *69*

Chapter 9
SAS Scoring Accelerator and SAS Model Manager *79*

Chapter 2

Introduction to the SAS Scoring Accelerator

Overview of the SAS Scoring Accelerator 9
How It Works 10
Special Characters in Directory Names 11

Overview of the SAS Scoring Accelerator

When using conventional processing to access data inside a DBMS, SAS Enterprise Miner asks the SAS/ACCESS engine for all rows of the table being processed. The SAS/ACCESS engine generates an SQL SELECT * statement that is passed to the database. That SELECT statement fetches all the rows in the table, and the SAS/ACCESS engine returns them to SAS Enterprise Miner. As the number of rows in the table grows over time, network latency grows because the amount of data that is fetched from the database to the SAS scoring process increases.

The SAS Scoring Accelerator embeds the robustness of SAS Enterprise Miner scoring models directly in the highly scalable database. By using the SAS In-Database technology and the SAS Scoring Accelerator, the scoring process is done inside the database and thus does not require the transfer of data.

The SAS Scoring Accelerator takes the models that are developed by SAS Enterprise Miner and translates them into scoring functions that can be deployed inside the database. After the scoring functions are published, the functions extend the database's SQL language and can be used in SQL statements like other database functions.

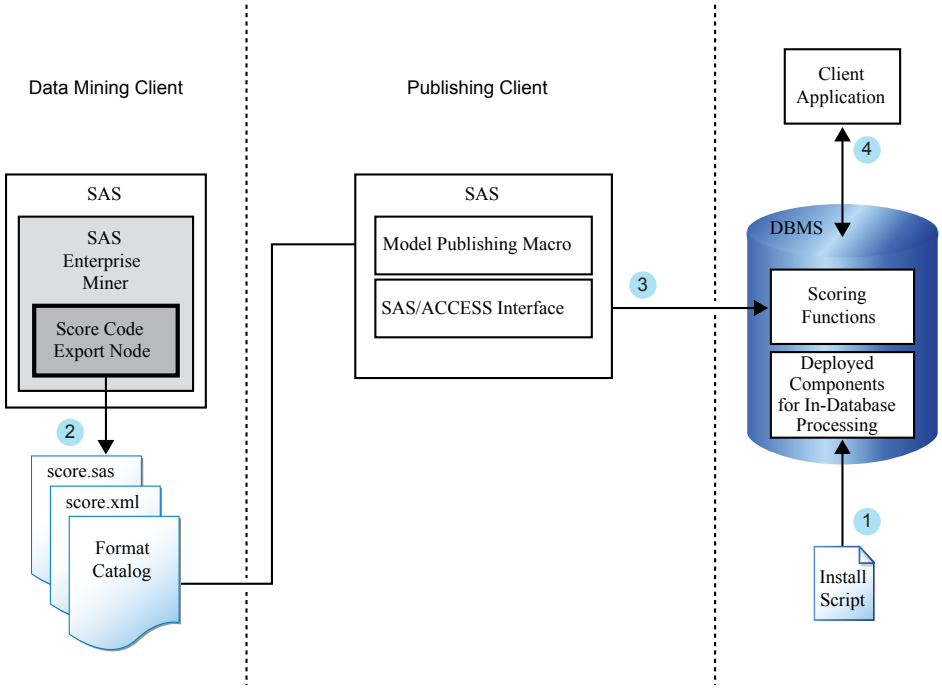
The SAS Scoring Accelerator consists of two components:

- the Score Code Export node in SAS Enterprise Miner. This extension exports the model scoring logic (including metadata about the required input and output variables) from SAS Enterprise Miner.
- the publishing client that includes a scoring publishing macro. This macro translates the scoring model into files that are used inside the database to run the scoring model. The publishing client then uses the SAS/ACCESS Interface to the database to publish the files to the database.

How It Works

Using SAS Enterprise Miner, you can generate SAS DATA step code that contains scoring functions. The SAS Scoring Accelerator takes the scoring model code, the associated property file that contains model inputs and outputs, and a catalog of user-defined formats. The SAS Scoring Accelerator deploys (or publishes) them to the database. Inside the database, one or more scoring functions are created and registered for use in SQL queries. Figure 1.1 illustrates this process.

Figure 2.1 *Process Flow Diagram*



1 Install the components that are necessary for in-database processing.

   For more information, see the *SAS In-Database Products: Administrator's Guide*.

   Note: This is a one-time installation process.

2 Use SAS Enterprise Miner to create a scoring model, and use the Score Code Export node to export files that are used to create the scoring functions to a score output directory.

   For more information, see Chapter 3, "Exporting the Scoring Model Files from SAS Enterprise Miner," on page 13.

3 Start SAS 9.3 and run the SAS publishing macros. This creates the files that are needed to build the scoring files and functions and publish those files to the database.

   For more information, see the section on publishing scoring model files in the Scoring Accelerator chapter for your database.

4 After the scoring functions are created, they are available to use in any SQL expression in the same way that the database's built-in functions are used.

For more information, see the topic on scoring functions inside the database in the Scoring Accelerator chapter for your database.

Special Characters in Directory Names

If the directory names that are used in the macros contain any of the following special characters, you must mask the characters by using the %STR macro quoting function. For more information, see the %STR function and macro string quoting topic in *SAS Macro Language: Reference*.

Character	How to Represent
blank[1]	%str( )
*[2]	%str(*)
;	%str(;)
,	%str(,)
=	%str(=)
+	%str(+)
-	%str(-)
>	%str(>)
<	%str(<)
^	%str(^)
|	%str(|)
&	%str(&)
#	%str(#)
/	%str(/)
~	%str(~)
%	%str(%%)
'	%str(%')
"	%str(%")
(	%str(%()
)	%str(%))

Character	How to Represent
¬	%str(¬)

[1]Only leading blanks require the %STR function, but you should avoid using leading blanks in directory names.

[2]Asterisks are allowed in UNIX directory names. Asterisks are not allowed in Windows directory names. In general, you should avoid using asterisks in directory names.

Here are some examples of directory names with special characters:

Directory	Code Representation
c:\temp\Sales(part1)	`c:\temp\Sales%str(%()part1%str(%))`
c:\temp\Drug "trial" X	`c:\temp\Drug %str(%")trial(%str(%") X`
c:\temp\Disc's 50% Y	`c:\temp\Disc%str(%')s 50%str(%%) Y`
c:\temp\Pay,Emp=Z	`c:\temp\Pay%str(,)Emp%str(=)Z`

Chapter 3

Exporting the Scoring Model Files from SAS Enterprise Miner

Overview of the Score Code Export Node **13**

Comparing the Score Code Export Node with Registering Models on the SAS Metadata Server **14**

Using the Score Code Export Node **14**
Using the Score Code Export Node in a Process Flow Diagram 14
Score Code Export Node Properties 15

Output Created by the Score Code Export Node **16**
Results Window 16
Output Files 17
Output Variables 18
Fixed Variable Names 19
SAS Enterprise Miner Tools Production of Score Code 20

Overview of the Score Code Export Node

Users of SAS Enterprise Miner develop data mining models that use measured attributes to either characterize or predict the value of an event. These models are developed on historical data where an event has been measured or inferred. The models are then applied to new data for which the attributes are known, but the event has not yet occurred. For example, a model can be created based on a credit institution's records of payments that customers made and missed last year. The model can then be used to predict which customers will miss payments this year.

SAS Enterprise Miner creates SAS language score code for the purpose of scoring new data. Users run this code in production systems to make business decisions for each record of new data.

The Score Code Export node is an extension for SAS Enterprise Miner that exports files that are necessary for score code deployment. Extensions are programmable add-ins for the SAS Enterprise Miner environment.

The following icon is the Score Code Export node as it appears in a SAS Enterprise Miner process flow diagram.

The following files are exported by the Score Code Export node:

- the SAS scoring model program (score.sas).
- a properties file that contains a description of the variables that are used and created by the scoring code (score.xml).
- a format catalog, if the scoring program contains user-defined formats.
- an XML file containing descriptions of the final variables that are created by the scoring code. This file can be kept for decision-making processes.
- a ten-row sample of the scored data set showing typical cases of the input attributes, intermediate variables, and final output variables. This data set can be used to test and debug new scoring processes.
- a ten-row sample table of the training data set showing the typical cases of the input attributes used to develop the score code.

For more information about the exported files, see "Output Files" on page 17. For more information about using SAS Enterprise Miner, see the SAS Enterprise Miner Help.

Comparing the Score Code Export Node with Registering Models on the SAS Metadata Server

SAS Enterprise Miner can register models directly in the SAS Metadata Server. Models registered in the SAS Metadata Server are used by SAS Data Integration Studio, SAS Enterprise Guide, and SAS Model Manager for creating, managing, and monitoring production and analytical scoring processes.

The Score Code Export node exports score code created by SAS Enterprise Miner into a format that can be used by the SAS Scoring Accelerator for Teradata. The exported files are stored in a directory, not the SAS Metadata Server.

The Score Code Export node does not replace the functionality of registering models in the SAS Metadata Server.

Using the Score Code Export Node

Using the Score Code Export Node in a Process Flow Diagram

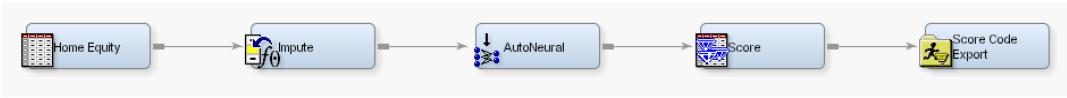
The **Score Code Export node** icon is located on the Utility tab, as shown in Figure 3.1:

Figure 3.1 *The Diagram Toolbar with the SAS Score Code Export Node Icon Highlighted*

To use the Score Code Export node, you need a process flow diagram that contains nodes that produce score code and that flow to a Score node. The Score node aggregates the score code for the entire analysis path. The Score node must precede the Score Code Export node in the process flow diagram.

This is a valid data mining process for exporting score code:

Figure 3.2 *Data Mining Process Flow Diagram*



Requirement: The Score Code Export node exports score code that contains only one DATA step. For a list of SAS Enterprise Miner nodes that produce score code, see "SAS Enterprise Miner Tools Production of Score Code" on page 20.

After the process flow diagram is in place, set the properties for the Score node and the Score Code Export node:

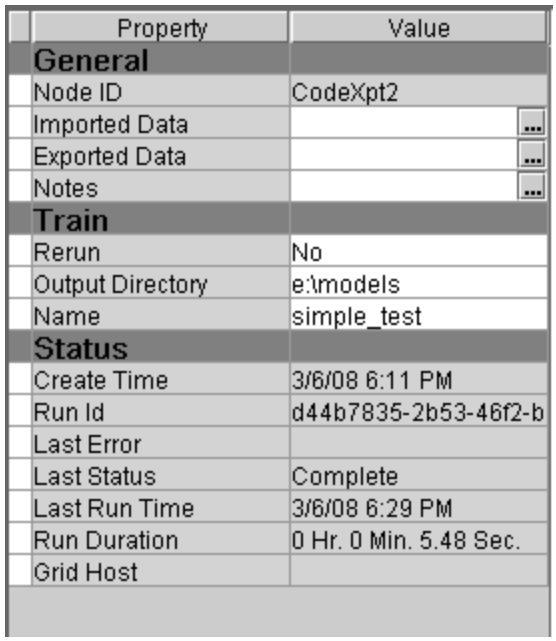
1. Select the Score node. Ensure that the following properties are set to the default value of Yes for each:
   - **Use Output Fixed Names**
   - **C Score**
2. Select the Score Code Export node and set the properties. The **Output Directory** property specifies the directory to store the export files. The **Name** property specifies the folder that contains the output files created by the Score Code Export node. For information about the properties, see "Score Code Export Node Properties" on page 15.

After the properties are set, you are ready to export the score code. Right-click the Score Code Export node and select **Run**. When SAS Enterprise Miner completes processing, the Run Status window opens to indicate that the run completed. Click the **Results** button to view the output variables and the listing output. For information about the output, see "Output Created by the Score Code Export Node" on page 16.

Score Code Export Node Properties

When the Score Code Export node is selected in the diagram workspace, the Properties panel displays all of the properties that the node uses and their associated values, as shown in Figure 3.3.

Figure 3.3 *Properties Panel*



Property	Value
General	
Node ID	CodeXpt2
Imported Data	...
Exported Data	...
Notes	...
Train	
Rerun	No
Output Directory	e:\models
Name	simple_test
Status	
Create Time	3/6/08 6:11 PM
Run Id	d44b7835-2b53-46f2-b
Last Error	
Last Status	Complete
Last Run Time	3/6/08 6:29 PM
Run Duration	0 Hr. 0 Min. 5.48 Sec.
Grid Host	

The following Train properties are associated with the Score Code Export node:

- **Rerun** – Use this property to force the node to run again. This property is useful if the macro variable controlling the target directory and folder name has changed.
- **Output Directory** – Enter a fully qualified name for the location of an output directory to contain the score code files. If no directory is entered, a default directory named Score is created in the SAS Enterprise Miner project directory. You can change the value of the default directory by setting the &EM_SCOREDIR=*directory* macro variable in the SAS Enterprise Miner project start-up code or server start-up code.
- **Name** – Enter the name of the model that you are creating. The name is used to create a new subdirectory in the output directory that contains the exported score files. If no name is entered, a default name is generated as a combination of the &SYSUSERID automatic macro variable and an incremental index (for example, userID, userID_2, userID_3).

  You can replace the &SYSUSERID automatic macro variable with a custom name by setting the &EM_SCOREFOLDER=*score-folder-name* macro variable in the SAS Enterprise Miner project start-up code or server start-up code. An incremental index preceded by an underscore is added to *score-folder-name*.

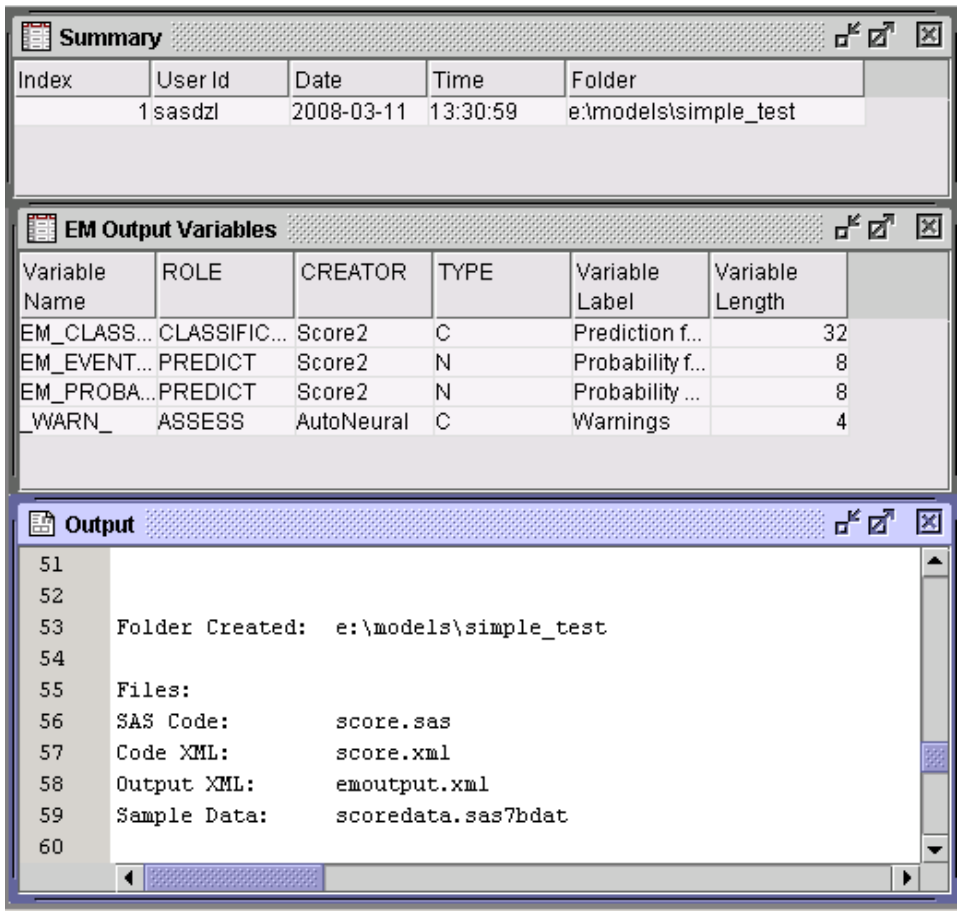
The General and Status properties for the Score Code Export node function just as they do for other nodes.

Output Created by the Score Code Export Node

Results Window

Using the values set in the Properties panel (Figure 3.3), the Score Code Export node creates the following output in the Results window:

Figure 3.4 *Results Using Sample Properties in the Properties Panel*



Output Files

The Score Code Export node writes the following output files, and a format catalog, if applicable, to the location specified by the Output Directory property. These files are used as input to the scoring publishing macro that creates the scoring functions.

Table 3.1 *Score Code Export Node Output Files*

File or Folder	Description

score.sas	SAS language score code created by SAS Enterprise Miner. This code can be used directly in a SAS program. A sample program based on the properties shown in Figure 3.3 looks like this:
	`data testout ;`
	`  set simpletest.scoredata ;`
	`  %include "c:\models\simpletest\score.sas";`
	`run;`

File or Folder	Description
score.xml	A description of the variables that are used and created by the scoring code. XML files are created by a machine process for the use of machine processes. Do not edit the XML file. **Restriction:** The maximum number of input variables for a scoring function is 128.
emoutput.xml	A description of the final variables that are created by the scoring code. This file can be kept for decision-making processes. These variables include the primary classification, prediction, probability, segment, profit, and loss variables created by a data mining process. The list does not include intermediate variables created by the analysis. For more information about these variables, see "Fixed Variable Names" on page 19. *Note:* The emoutput.xml file is not used by the scoring publishing macro.
scoredata.sas7bdat	A ten-row sample of the scored data set showing typical cases of the input attributes, intermediate variables, and final output variables. Use this data set to test and debug new scoring processes. *Note:* The scoredata.sas7bdat file is not used by the scoring publishing macro.
traindata.sas7bdat	A ten-row sample table of the training data set showing typical cases of the input attributes used to develop the score code. *Note:* The traindata.sas7bdat file is not used by the scoring publishing macro.
Format Catalog	If the training data contains SAS user-defined formats, the Score Code Export node creates a format catalog. The catalog contains the user-defined formats in the form of a lookup table. This file has an extension of .sas7bcat.

Output Variables

The score code produced by SAS Enterprise Miner creates both intermediate variables, such as imputed values of missing values, transformations, and encodings; and output variables, such as predicted value and probability. Any of these created variables can be used in a scoring process.

TIP The number of input parameters on a scoring function has a direct impact on performance. The more parameters there are, the more time it takes to score a row. A recommended best practice is to make sure that only variables that are involved in a model score evaluation are exported from SAS Enterprise Miner.

The most important output variables for the scoring process follow a naming convention using a prefix, as shown in the following table.

Table 3.2 *Output Variables*

Role	Type	Prefix	Key	Suffix	Example
Prediction	N	P_	Target variable name		P_amount
Probability	N	P_	Target variable name	Predicted event value	P_purchaseYES P_purchaseNO
Classification	$	I_	Target variable name		I_purchase
Expected Profit	N	EP_	Target variable name		EP_conversion
Expected Loss	N	EL_	Target variable name		EL_conversion
Return on Investment	N	ROI_	Target variable name		ROI_conversion
Decision	$	D_	Target variable name		D_conversion
Decision Tree Leaf	N	_NODE_			_NODE_
Cluster number or SOM cell ID	N	_SEGMENT_			_SEGMENT_

Fixed Variable Names

The Score node of SAS Enterprise Miner maps the output variable names to fixed variable names. This mapping is appropriate in cases where there is only one prediction target or one classification target. In other cases, refer to the output variable names described in the previous table.

Using the fixed variable names enables scoring users to build processes that can be reused for different models without changing the code that processes the outputs. These fixed names are listed in the emoutput.xml file and are described in the following table. Most scoring processes return one or more of these variables.

Table 3.3 *Fixed Variable Names*

Role	Type	Fixed Name	Description
Prediction	N	EM_PREDICTION	The prediction value for an interval target.
Probability	N	EM_PROBABILITY	The probability of the predicted classification, which can be any one of the target variable values.
Probability	N	EM_EVENTPROBABILITY	The probability of the target event. By default this is the first value in descending order. This is often the event of interest. The user can control the ordering in SAS Enterprise Miner.
Classification	$	EM_CLASSIFICATION	The predicted target class value.
Expected Profit	N	EM_PROFIT	Based on the selected decision.
Expected Loss	N	EM_LOSS	Based on the selected decision.
Return on Investment	N	EM_ROI	Based on the selected decision.
Decision	$	EM_DECISION	Optimal decision based on a function of probability, cost, and profit or loss weights.
Decision Tree Leaf, Cluster number, or SOM cell ID	N	EM_SEGMENT	Analytical customer segmentation.

SAS Enterprise Miner Tools Production of Score Code

The following table shows the types of score code created by each node in SAS Enterprise Miner. Users can develop their own nodes, known as extension nodes, which can create either SAS DATA step or SAS program score code. However, this code is not converted to PMML, C, or Java.

Table 3.4 *Types of Score Code Created by Node*

Node	SAS DATA Step	SAS Program	PMML	C	Java	DBMS
Sample						
Input Data	*	*	*	*	*	*
Sample	*	*	*	*	*	*
Partition	*	*	*	*	*	*
Append	N	Y	N	N	N	N

Node	SAS DATA Step	SAS Program	PMML	C	Java	DBMS
Merge	N	Y	N	N	N	N
Time Series	N	Y	N	N	N	N
Filter	Y When the user keeps the created filter variable.	*	N	Y	Y	Y
Explore						
Association	N	Y	Y	N	N	N
Cluster	Y	N	Y	Y	Y	Y
DMDB	*	*	*	*	*	*
Graph Explore	*	*	*	*	*	*
Market Basket	N	Y	N	N	N	N
Multiplot	*	*	*	*	*	*
Path	N	Y	Y	N	N	N
SOM	Y	N	N	Y	Y	Y
Stat Explore	*	*	*	*	*	*
Text Miner	N	Y	N	N	N	N
Variable Clustering	Y	N	N	Y	Y	Y**
Variable Selection	Y	N	N	Y	Y	Y
Modify						
Drop	*	*	*	*	*	*
Impute	Y	N	Y	Y	Y	Y
Interactive Binning	Y	N	N	Y	Y	Y
Replacement	Y	N	N	Y	Y	Y
Principal Components	Y	N	N	Y	Y	Y

Node	SAS DATA Step	SAS Program	PMML	C	Java	DBMS
Rules Builder	Y	N	N	Y	Y	Y**
Transform Variables	Y	N	N	Y	Y	Y
Model						
Autoneural	Y	N	Y	Y	Y	Y
Decision Tree	Y	N	Y	Y	Y	Y
Dmine Regression	Y	N	Y	Y	Y	Y
Dmine Neural	Y	N	N	Y	Y	Y
Ensemble	Y	N	N	Y	Y	Y
Gradient Boosting	Y	N	N	Y	Y	Y
MBR	N	Y	N	N	N	N
Model Import	*	*	*	*	*	*
Neural Network	Y	N	Y	Y	Y	Y
Partial Least Squares	Y	N	N	Y	Y	Y
Rule Induction	Y	N	N	Y	Y	Y
SVM : Linear Kernel	Y	N	Y	Y	Y	Y
SVM : Nonlinear Kernel	N	Y	N	N	N	N
Two Stage	Y	N	N	Y	Y	Y
Assess						
Cutoff	Y	N	N	Y	Y	Y
Decisions	Y	N	N	Y	Y	Y
Model Comparison	Y	N	N	Y	Y	Y
Score	Y	N	N	Y	Y	Y

Node	SAS DATA Step	SAS Program	PMML	C	Java	DBMS
Segment Profile	*	*	*	*	*	*
Utility						
Control Point	*	*	*	*	*	*
Start Groups	Y	N	N	Y	Y	Y
End Groups	Y	N	N	Y	Y	Y
Metadata	*	*	*	*	*	*
Reporter	*	*	*	*	*	*
SAS Code The user can enter either SAS DATA step code or SAS program code	Y	Y	N	N	N	N
Credit Scoring						
Credit Exchange	*	*	*	*	*	*
Interactive Grouping	Y	N	N	Y	Y	Y
Scorecard	Y	N	N	Y	Y	Y
Reject Inference	Y	N	N	Y	Y	Y

* The node does not produce this type of score code.

** There is limited support for user-written code in this node. User-written code could produce errors or unexpected results.

Chapter 4

SAS Scoring Accelerator for Aster *n*Cluster

Publishing Scoring Model Files in Aster nCluster **25**

Running the %INDAC_PUBLISH_MODEL Macro **26**
%INDAC_PUBLISH_MODEL Macro Run Process 26
%INDACPM Macro 26
INDCONN Macro Variable 27
%INDAC_PUBLISH_MODEL Macro Syntax 28
Model Publishing Macro Example 30
Aster nCluster Permissions 30

Scoring Files and Functions inside the Aster nCluster Database **31**
Aster nCluster Scoring Files 31
SAS_SCORE() Function 32

Publishing Scoring Model Files in Aster *n*Cluster

The integration of the SAS Embedded Process and Aster *n*Cluster allows scoring code to be running directly using the SAS Embedded Process on Aster *n*Cluster through a SQL/MR function.

The SQL/MR function is the framework for enabling execution of user-defined functions within Aster *n*Cluster through an SQL interface. A SAS SQL/MR function, SAS_SCORE(), performs the scoring of models published in Aster *n*Cluster.

The %INDAC_PUBLISH_MODEL macro uses some of the files that are created by the SAS Enterprise Miner Score Code Export node: the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog.

The %INDAC_PUBLISH_MODEL macro performs the following tasks:

- takes the score.sas and score.xml files that are created using the Score Code Export node and produces two files for each scoring model. The following files are produced:
  - sasscore_*modelname*.ds2 . This file contains code that is executed by the SAS_SCORE() function
  - sasscore_*modelname*_io.xml. This file contains the scoring model's input and output variables

- takes the format catalog, if available, and produces the sasscore_*modelname*_ufmt.xml file. This file contains user-defined formats for the scoring model that is being published.
- uses the SAS/ACCESS Interface to Aster *n*Cluster to insert the three scoring files into the NC_INSTALLED_FILES table.

After the scoring files are published, you can call the SAS_SCORE() function to execute the scoring model. For more information, see "SAS_SCORE() Function" on page 32.

Running the %INDAC_PUBLISH_MODEL Macro

%INDAC_PUBLISH_MODEL Macro Run Process

To run the %INDAC_PUBLISH_MODEL macro, complete the following steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Use the SAS Enterprise Miner Score Code Export node to create a score output directory and populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS 9.3 and submit one of these following sets of commands in the Program Editor or Enhanced Editor:

```
%indacpm;
%let indconn = user=myuserid password=XXXX
dsn=ncluster <schema=myschema>;

%indacpm;
%let indconn = user=myuserid password=XXXX server=myserver
 database=mydatabase <schema=myschema>;
```

   For more information, see "%INDACPM Macro" on page 26, and "INDCONN Macro Variable" on page 27.

4. Run the %INDAC_PUBLISH_MODEL macro.

   Messages are written to the SAS log that indicate the success or failure of the creation of the .ds2 and .xml scoring files.

   For more information, see "%INDAC_PUBLISH_MODEL Macro Syntax" on page 28.

%INDACPM Macro

The %INDACPM macro searches the autocall library for the indacpm.sas file. The indacpm.sas file contains all the macro definitions that are used in conjunction with the %INDAC_PUBLISH_MODEL macro. The indacpm.sas file should be in one of the directories listed in the SASAUTOS= system option in your configuration file. If the indacpm.sas file is not present, the %INDACPM macro call (%INDACPM; statement) issues the following message:

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Aster *n*Cluster. You must specify user, password, and either a DSN name or a server and database name. You must assign the INDCONN macro variable before the %INDAC_PUBLISH_MODEL macro is invoked.

The value of the INDCONN macro variable for the %INDAC_PUBLISH_MODEL macro has one of these formats:

USER=*username* PASSWORD=*password* DSN=*dsnname* <SCHEMA=*schemaname*>

USER=*username* PASSWORD=*password* DATABASE=*databasename* SERVER=*servername* <SCHEMA=*schemaname*>

Arguments

USER=*username*
specifies the Aster *n*Cluster user name (also called the user ID) that is used to connect to the database.

PASSWORD=*password*
specifies the password that is associated with your Aster *n*Cluster user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error.

DSN=*datasourcename*
specifies the configured Aster *n*Cluster data source to which you want to connect.

Requirement: You must specify either the DSN= argument alone, or the SERVER= and DATABASE= arguments together in the INDCONN macro variable.

DATABASE=*databasename*
specifies the Aster *n*Cluster database that contains the tables and views that you want to access.

Requirement: You must specify either the DSN= argument alone, or the SERVER= and DATABASE= arguments together in the INDCONN macro variable.

SERVER=*servername*
specifies the Aster *n*Cluster server name or the IP address of the server host.

Requirement: You must specify either the DSN= argument alone, or the SERVER= and DATABASE= arguments together in the INDCONN macro variable.

SCHEMA=*schemaname*
specifies the schema name for the database.

Default: your default schema. To determine your default schema name, use the **show search_path** command from the Aster Client Tool (ACT).

Restriction: The SCHEMA argument is valid only for Aster *n*Cluster 4.6. For Aster *n*Cluster 4.5, the scoring model XML files are published to the PUBLIC schema.

TIP The INDCONN macro variable is not passed as an argument to the %INDAC_PUBLISH_MODEL macro. This information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDAC_PUBLISH_MODEL Macro Syntax

%INDAC_PUBLISH_MODEL

(DIR=*input-directory-path*, MODELNAME=*name*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, DATABASE=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DIR=*input-directory-path*

specifies the directory where the scoring model program, the properties file, and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node. This directory contains the score.sas file, the score.xml file, and, if user-defined formats were used, the format catalog.

Requirement: You must use a fully qualified pathname.

Interaction: If you do not use the default filenames that are created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, and (if needed) FMTCAT= arguments.

See: "Special Characters in Directory Names" on page 11

MODELNAME=*name*

specifies the name that becomes part of the .ds2 and .xml scoring filenames.

Restriction: The names of the .ds2 and .xml scoring files are a combination of the model and type of filenames. A scoring filename cannot exceed 63 characters. For more information, see "Aster nCluster Scoring Files" on page 31.

Requirement: The name must be a valid SAS name. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*.

Interaction: Only the EM_ output variables are published in the sasscore_*modelname*_io.xml file. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19 and "Aster nCluster Scoring Files" on page 31.

DATASTEP=*score-program-filename*

specifies the name of the scoring model program file that was created by using the SAS Enterprise Miner Score Code Export node.

Default: score.sas

Restriction: Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:

If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the DATASTEP= argument.

The SAS file that is specified in the DATASTEP= argument is translated by the %INDAC_PUBLISH_MODEL macro into the sasscore_*modelname*.ds2 file and stored in the NC_INSTALLED_FILES table under either the PUBLIC schema (Aster *n*Cluster 4.5) or the schema that you specified in the INDCONN macro variable (Aster *n*Cluster 4.6).

XML=*xml-filename*
specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

Default: score.xml

Restrictions:
Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

The maximum number of output variables is 128.

Interactions:
If you use the default score.xml file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the XML= argument.

The XML file is renamed to sasscore_*modelname*_io.xml by the %INDAC_PUBLISH_MODEL macro and the file is stored in the NC_INSTALLED_FILES table under either the PUBLIC schema (Aster *n*Cluster 4.5) or the schema that you specified in the INDCONN macro variable (Aster *n*Cluster 4.6).

DATABASE=*database-name*
specifies the name of an Aster *n*Cluster database to which the scoring functions and formats are published.

Restriction: If you specify DSN= in the INDCONN macro variable, do not use the DATABASE argument.

Interaction: The database that is specified by the DATABASE argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "%INDAC_PUBLISH_MODEL Macro Run Process" on page 26.

Tip: You can publish the scoring files to a shared database where other users can access them.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file. The file contains all user-defined formats that were created by the FORMAT procedure and that are referenced in the DATA step scoring model program.

Restriction: Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:
If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the FMTCAT= argument.

If you do not use the default catalog name (FORMATS) or the default library (WORK or LIBRARY) when you create user-defined formats, you must use the FMTSEARCH system option to specify the location of the format catalog. For more information, see PROC FORMAT in the *Base SAS Procedures Guide*.

ACTION=CREATE | REPLACE | DROP
specifies one of the following actions that the macro performs:

CREATE
creates the sasscore_*modelname*.ds2, sasscore_*modelname*_io.xml, and sasscore_*modelname*_ufmt.xml files.

REPLACE
overwrites the current sasscore_*modelname*.ds2, sasscore_*modelname*_io.xml, and sasscore_*modelname*_ufmt.xml files, if those files by the same name are already registered.

DROP
causes the sasscore_*modelname*.ds2, sasscore_*modelname*_io.xml, and sasscore_*modelname*_ufmt.xml files to be dropped from the NC_INSTALLED_FILES table in the Aster *n*Cluster database.

Default: CREATE

Tip: If the scoring files have been previously defined and you specify ACTION=CREATE, you receive warning messages from Aster *n*Cluster. If the scoring files have been previously defined and you specify ACTION=REPLACE, no warnings are issued.

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process and sample SQL code (SampleSQL.txt). For more information about the SampleSQL.txt file, see "Aster nCluster Scoring Files" on page 31.

Tip: This argument is useful to debug a scoring model that fails to be published.

See: "Special Characters in Directory Names" on page 11

Model Publishing Macro Example

```
%indacpm;
%let indconn = server=yoursvr user=user1 password=open1
    database=yourdb schema=yoursch;
%indac_publish_model( dir=C:\SASIN\score, modelname=score);
```

The %INDAC_PUBLISH_MODEL macro produces these three files:

- sasscore_score.ds2. See "Example of a .ds2 Scoring File" on page 147.
- sasscore_score_io.xml. See "Example of an Input and Output Variables Scoring File" on page 167.
- sasscore_score_ufmt.xml. See "Example of a User-Defined Formats Scoring File" on page 174.

After the scoring files are installed, they can be invoked in Aster *n*Cluster using the SAS_SCORE() function. For more information, see "SAS_SCORE() Function" on page 32.

Aster nCluster Permissions

For Aster *n*Cluster 4.5, no permissions are needed by the person who runs the scoring publishing macros, because all functions and files are published to the PUBLIC schema.

For Aster *n*Cluster 4.6, the following permissions are needed for the schema by the person who runs the scoring publishing macros, because all functions and files can be published to a specific schema.

- USAGE permission
- INSTALL FILE permission
- CREATE permission

Without these permissions, the publishing of the %INDAC_PUBLISH_MODEL macro fails. To obtain these permissions, contact your database administrator.

For more information about specific permissions, see the *SAS In-Database Products: Administrator's Guide*.

Scoring Files and Functions inside the Aster *n*Cluster Database

Aster nCluster Scoring Files

The %INDAC_PUBLISH_MODEL macro produces three scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the SAS_SCORE() function
- sasscore_*modelname*_io.xml. This file contains the scoring model's input and output variables
- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published

These files are inserted into the PUBLIC schema (Aster *n*Cluster 4.5) or the schema that you specified in the INDCONN macro variable (Aster *n*Cluster 4.6). See Appendix 1, "Scoring File Examples for Aster nCluster," on page 147 for an example of each of these files.

There are four ways to see the scoring files that are created:

- Log on to the database using the Aster *n*Cluster command line processor and submit an SQL statement. The following example assumes that the model name that you used to create the scoring files is **reg**.

```
>act -h hostname -u username -w password -d databasename
>select name from nc_user_installed_files where name like 'sasscore_reg%';
```

  Three files are listed for each model:

```
    name
------------------------
 sasscore_reg.ds2
 sasscore_reg_io.xml
 sasscore_reg_ufmt.xml
```

- From SAS, use SQL procedure code that produces output in the LST file. The following example assumes that the model name that you used to create the scoring functions is **reg**.

```
proc sql noerrorstop;
  connect to aster (user=username password=password dsn=dsnname);

select *
   from connection to aster
      (select name, owner, uploadtime
         from nc_user_installed_files where
            name like 'sasscore_reg%');
   disconnect from aster;
quit;
```

You can also use the SASTRACE and SASTRACELOC system options to generate tracing information. For more information about these system options, see the *SAS System Options: Reference*.

- Look at the SampleSQL.txt file that is produced when the %INDAC_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the %INDAC_PUBLISH_MODELmacro.

  The SampleSQL.txt file contains basic code that, with modifications, can be used to run your score code inside Aster *n*Cluster.

  Note: The function and table names must be fully qualified if the functions and tables are not in the same database.

  For example, the SampleSQL.txt file refers to an ID column in **allmush1_intab** that is populated with a unique integer from 1 to *n*, with *n* being the number of rows in the table. The ID column uniquely identifies each row. You would replace the ID column with your own primary key column.

  The following example assumes that the model name that you used is **reg**.

```
drop table score_outtab;
create table score_outtab(
 id integer
,"EM_CLASSIFICATION" varchar(256)
,"EM_EVENTPROBABILITY" float
,"EM_PROBABILITY" float
);
insert into score_outtab(
 id
,"EM_CLASSIFICATION"
,"EM_EVENTPROBABILITY"
,"EM_PROBABILITY"
)
select id,
"EM_CLASSIFICATION",
"EM_EVENTPROBABILITY",
"EM_PROBABILITY"
from sas_score(on score_intab model('reg'));
```

- Look at the SAS log. A message that indicates whether the scoring files are successfully or not successfully created is printed to the SAS log.

SAS_SCORE() Function

Overview of the SAS_SCORE() Function

The SAS_SCORE() function is an SQL/MR function that executes the scoring model running on the SAS Embedded Process in Aster *n*Cluster. The SAS_SCORE() function is deployed and stored in the PUBLIC schema during the installation and configuration of the in-database deployment for Aster *n*Cluster.

For more information about installing and configuring the in-database deployment package for Aster *n*Cluster, see the *SAS In-Database Products: Administrator's Guide*.

Using the SAS_SCORE() Function

You can use the SAS_SCORE() function in the FROM clause in any SQL expression in the same way that Aster *n*Cluster SQL/MR functions are used.

The syntax of the SAS_SCORE() function is as follows:

FROM **SAS_SCORE**(ON *input-table* MODEL('*model-name*')
<MODEL_SCHEMA('*schemal-name*>))

Arguments

input-table
specifies the input table that is used by the SAS_SCORE() function.

model-name
specifies the name of the model. The value of this argument is the same as the value of MODELNAME=*name* argument for the %INDAC_PUBLISH_MODEL macro.

schema-name
specifies the name of the schema where the scoring model files are published.

Restriction: This argument is valid only for Aster *n*Cluster 4.6. For Aster *n*Cluster 4.5, the scoring model files are published to the PUBLIC schema.

Default: your default schema. To determine your default schema name, use the **`show search_path`** command from the Aster Client Tool (ACT).

Here is an example of using the SAS_SCORE function. In this example, the input table is **score_intab** and the model name is **reg**.

```
select id, em_classification, em_eventprobability, em_probability
   from sas_score (on score_intab model('reg') model_schema('mysch'));
```

Chapter 5

SAS Scoring Accelerator for DB2 under UNIX

Publishing Scoring Model Files in DB2 . . . . . **35**

Running the %INDB2_PUBLISH_MODEL Macro **36**
%INDB2_PUBLISH_MODEL Macro Run Process 36
%INDB2PM Macro 37
INDCONN Macro Variable 37
%INDB2_PUBLISH_MODEL Macro Syntax 38
Modes of Operation 41
Model Publishing Macro Example 41
DB2 Permissions 43

Scoring Functions inside the DB2 Database **43**
Scoring Function Names 43
Using the Scoring Functions 44

Publishing Scoring Model Files in DB2

The %INDB2_PUBLISH_MODEL macro creates the files that are needed to build the scoring functions and publishes the scoring functions with those files to a specified database in DB2. Only the EM_ output variables are published as DB2 scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19.

Note: Secure File Transfer Protocol (SFTP) is used to transfer the source files to the DB2 server during the publishing process. Certain software products that support SSH-2 or SFTP protocols must be installed before you can use the publishing macros. For more information, see *Setting up SSH Client Software in UNIX and Windows Environments for use with the SFTP Access Method* located at **http://support.sas.com/techsup/technote/ts800.pdf**.

The %INDB2_PUBLISH_MODEL macro uses some of the files that are created by the SAS Enterprise Miner Score Code Export node: the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog.

The %INDB2_PUBLISH_MODEL macro performs the following tasks:

- takes the score.sas and score.xml files and produces the set of .c and .h files. These .c and .h files are necessary to build separate scoring functions for each of a fixed set of quantities that can be computed by the scoring model code.

- if a format catalog is available, processes the format catalog and creates an .h file with C structures. These files are also necessary to build the scoring functions.
- produces a script of the DB2 commands that are used to register the scoring functions on the DB2 database.
- transfers the .c and .h files to DB2 using SFTP.
- calls the SAS_COMPILEUDF function to compile the source files into object files, links to the SAS formats library, and copies the new object files to ***db2path*/sqllib/function/SAS**, where ***db2path*** is the path that was defined during installation. The object filename is *dbname_schemaname_modelname_segnum*, where *segnum* is a sequence number that increments each time the model is replaced or recreated. The object file is renamed to avoid library caching in DB2.
- calls the SAS_DELETEUDF function to remove existing object files.
- uses the SAS/ACCESS Interface to DB2 to run the script to create the scoring functions with the object files.

The scoring functions are registered in DB2 with shared object files, which are loaded at run time. These functions are stored in a permanent location. The SAS object files and the SAS formats library are stored in the ***db2path*/sqllib/function/SAS** directory, where ***db2path*** is the path that was defined during installation. This directory is accessible to all database partitions.

DB2 caches the object files after they are loaded. Each time that the updated objects are used, the database must be stopped and restarted to clean up the cache, or the object files need to be renamed and the functions reregistered with the new object filenames. The SAS publishing process automatically handles the renaming to avoid stopping and restarting the database.

Note: You can publish scoring model files with the same model name in multiple databases and schemas. Because object files for the SAS scoring function are stored in the ***db2path*/sqllib/function/SAS** directory, the publishing macros use the database, schema, and model name as the object filename to avoid potential naming conflicts.

Running the %INDB2_PUBLISH_MODEL Macro

%INDB2_PUBLISH_MODEL Macro Run Process

To run the %INDB2_PUBLISH_MODEL macro, complete the following steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Use the SAS Enterprise Miner Score Code Export node to create a score output directory and populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS 9.3 and submit the following commands in the Program Editor or Enhanced Editor:

```
%indb2pm;
%let indconn = server=yourserver user=youruserid password=yourpwd
    database=yourdb schema=yourschema serveruserid=yourserveruserid;
```

For more information, see the "%INDB2PM Macro" on page 37 and the "INDCONN Macro Variable" on page 37.

4. Run the %INDB2_PUBLISH_MODEL macro.

   For more information, see "%INDB2_PUBLISH_MODEL Macro Syntax" on page 38.

   Messages are written to the SAS log that indicate the success or failure of the creation of the scoring functions.

%INDB2PM Macro

The %INDB2PM macro searches the autocall library for the indb2pm.sas file. The indb2pm.sas file contains all the macro definitions that are used in conjunction with the %INDB2_PUBLISH_MODEL macro. The indb2pm.sas file should be in one of the directories listed in the SASAUTOS= system option in your configuration file. If the indb2pm.sas file is not present, the %INDB2PM macro call (%INDB2PM; statement) issues the following message:

```
macro indb2pm not defined
```

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to DB2. You must specify server, user, password, and database information to access the machine on which you have installed the DB2 database. The schema name and the server user ID are optional. You must assign the INDCONN macro variable before the %INDB2_PUBLISH_MODEL macro is invoked.

Here is the syntax for the value of the INDCONN macro variable for the %INDB2_PUBLISH_MODEL macro:

SERVER=*server* USER=*user* PASSWORD=*password*
DATABASE=*database* <SCHEMA=*schema*>
<SERVERUSERID=*serveruserid*>

Arguments

SERVER=*server*
specifies the DB2 server name or the IP address of the server host.

Requirement: The name must be consistent with the way the host name was cached when PSFTP *server* was run from the command window. If the full server name was cached, you must use the full server name in the SERVER argument. If the short server name was cached, you must use the short server name. For example, if the long name, *disk3295.unx.comp.com*, is used when PSFTP was run, then *server=disk3295.unx.comp.com* must be specified. If the short name, *disk3295*, was used, then *server=disk3295* must be specified. For more information about running the PSFTP command, see *DB2 Installation and Configuration Steps* in the *SAS In-Database Products: Administrator's Guide*.

USER=*userid*
specifies the DB2 user name (also called the user ID) that is used to connect to the database.

PASSWORD=*password*
specifies the password that is associated with your DB2 user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error.

DATABASE=*database*
specifies the DB2 database that contains the tables and views that you want to access.

Requirement: The scoring model functions are created as Unicode functions. If the database is not a Unicode database, then the alternate collating sequence must be configured to use `identity_16bit`.

SCHEMA=*schema*
specifies the schema name for the database.

Default: If you do not specify a value for the SCHEMA argument, the value of the USER argument is used as the schema name.

SERVERUSERID=*serveruserid*
specifies the user ID for SAS SFTP and enables you to access the machine on which you have installed the DB2 database.

Default: If you do not specify a value for the SERVERUSERID argument, the value of the USER argument is used as the user ID for SAS SFTP.

Note: The person who installed and configured the SSH software can provide the SERVERUSERID (SFTP user ID) and the private key that need to be added to the pageant.exe (Windows) or SSH agent (UNIX). Pageant must be running for the SFTP process to be successful.

TIP The INDCONN macro variable is not passed as an argument to the %INDB2_PUBLISH_MODEL macro. This information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDB2_PUBLISH_MODEL Macro Syntax

%INDB2_PUBLISH_MODEL

(DIR=*input-directory-path*, MODELNAME=*name*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, DATABASE=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
<, MODE=FENCED | UNFENCED>
<, INITIAL_WAIT=*wait-time*>
<, FTPTIMEOUT=*timeout-time*>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DIR=*input-directory-path*
specifies the directory where the scoring model program, the properties file, and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node. This directory contains the score.sas file, the score.xml file, and, if user-defined formats were used, the format catalog.

Requirement: You must use a fully qualified pathname.

Interaction: If you do not use the default directory that is created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, and, if needed, FMTCAT= arguments.

See: "Special Characters in Directory Names" on page 11

MODELNAME=*name*
specifies the name that is prepended to each output function to ensure that each scoring function name is unique on the DB2 database.

Restriction: The scoring function name is a combination of the model and output variable names. A scoring function name cannot exceed 128 characters. For more information, see "Scoring Function Names" on page 43.

Requirement: The model name must be a valid SAS name that is 10 characters or fewer. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*.

Interaction: Only the EM_ output variables are published as DB2 scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19 and "Scoring Function Names" on page 43.

DATASTEP=*score-program-filename*
specifies the name of the scoring model program file that was created by using the SAS Enterprise Miner Score Code Export node.

Default: score.sas

Restriction: Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interaction: If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the DATASTEP= argument.

XML=*xml-filename*
specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

Default: score.xml

Restrictions:
Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

The maximum number of output variables is 128.

Interaction: If you use the default score.xml file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the XML= argument.

DATABASE=*database-name*
specifies the name of a DB2 database to which the scoring functions and formats are published.

Requirement: The scoring model functions are created as Unicode functions. If the database is not a Unicode database, then the alternate collating sequence must be configured to use `identity_16bit`.

Interaction: The database that is specified by the DATABASE argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "%INDB2_PUBLISH_MODEL Macro Run Process" on page 36.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created by the FORMAT procedure and that are referenced in the DATA step scoring model program.

Restriction: Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:

If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the FMTCAT= argument.

If you do not use the default catalog name (FORMATS) or the default library (WORK or LIBRARY) when you create user-defined formats, you must use the FMTSEARCH system option to specify the location of the format catalog. For more information, see PROC FORMAT in the *Base SAS Procedures Guide*.

ACTION=CREATE | REPLACE | DROP
specifies one of the following actions that the macro performs:

CREATE
creates a new function.

REPLACE
overwrites the current function, if a function by the same name is already registered.

DROP
causes all functions for this model to be dropped from the DB2 database.

Default: CREATE

Tip: If the function has been previously defined and you specify ACTION=CREATE, you will receive warning messages from DB2. If the function has been previously defined and you specify ACTION=REPLACE, no warnings are issued.

MODE=FENCED | UNFENCED
specifies whether the running code is isolated in a separate process in the DB2 database so that a program fault does not cause the database to stop.

Default: FENCED

Tip: After the SAS scoring functions are validated in fenced mode, you can republish them in unfenced mode. You might see a performance advantage when you run in unfenced mode.

See: "Modes of Operation" on page 41

INITIAL_WAIT=*wait-time*
specifies the initial wait time in seconds for SAS SFTP to parse the responses and complete the SFTP -batchfile process.

Default: 15 seconds

Interactions:
The INITIAL_WAIT= argument works in conjunction with the FTPTIMEOUT= argument. Initially, SAS SFTP waits the amount of time specified by the INITIAL_WAIT= argument. If the SFTP -batchfile process is not complete after the initial wait time, retries occur until the wait time is equal to or greater than the time-out value specified by the FTPTIMEOUT= argument. All retries double the previous wait time. SAS SFTP fails after the time-out value is reached or exceeded, and an error message is written to the SAS log.

For example, assume you use the default values. The initial wait time is 15 seconds. The first retry waits for 30 seconds. The second retry waits for 60 seconds. The third retry waits for 120 seconds. This is the default time-out value. So, the default initial wait time and time-out values enable four possible tries—the initial try, and three retries.

See: FTPTIMEOUT= argument

FTPTIMEOUT=*time-out-value*
specifies the time-out value in seconds if SAS SFTP fails to transfer the files.

Default: 120 seconds

Interactions:

The FTPTIMEOUT= argument works in conjunction with the INITIAL_WAIT= argument. Initially, SAS SFTP waits the amount of time specified by the INITIAL_WAIT= argument. If the SFTP -batchfile process is not complete after the initial wait time, retries occur until the wait time is equal to or greater than the time-out value specified by the FTPTIMEOUT= argument. All retries double the previous wait time. SAS SFTP fails after the time-out value is reached or exceeded and an error message is written to the SAS log.

For example, assume you use the default values. The initial wait time is 15 seconds. The first retry waits for 30 seconds. The second retry waits for 60 seconds. The third retry waits for 120 seconds. This is the default time-out value. So the default initial wait time and time-out values enable four possible tries—the initial try, and three retries.

Tip: Use this argument to control how long SAS SFTP waits to complete a file transfer before timing out. A time-out failure could indicate a network or key authentication problem.

See: INITIAL_WAIT= argument

OUTDIR=*diagnostic-output-directory*

specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process and sample SQL code (SampleSQL.txt). For more information about the SampleSQL.txt file, see "Scoring Function Names" on page 43.

Tip: This argument is useful when testing your scoring models.

See: "Special Characters in Directory Names" on page 11

Modes of Operation

The %INDB2_PUBLISH_MODEL macro has two modes of operation: fenced and unfenced. You specify the mode by setting the MODE= argument.

The default mode of operation is fenced. Fenced mode means that the scoring function that is published is isolated in a separate process in the DB2 database when it is invoked, and an error does not cause the database to stop. It is recommended that you publish the scoring functions in fenced mode during acceptance tests.

When the scoring function is ready for production, you can run the macro to publish the scoring function in unfenced mode. You could see a performance advantage if the scoring function is published in unfenced mode.

Model Publishing Macro Example

```
%indb2pm;
%let indconn = server=db2base user=user1 password=open1 database=mydb;
%indb2_publish_model( dir=C:\SASIN\baseball1, modelname=baseball1);
```

The %INDB2_PUBLISH_MODEL macro produces a text file of DB2 CREATE FUNCTION commands as shown in the following example.

Note: This example file is shown for illustrative purposes. The text file that is created by the %INDB2_PUBLISH_MODEL macro cannot be viewed and is deleted after the macro is complete.

```
CREATE FUNCTION baseball1_EM_eventprobablility
(
```

```
"CR_ATBAT" float,
"CR_BB" float,
"CR_HITS" float,
"CR_HOME" float,
"CR_RBI" float,
"CR_RUNS" float,
"DIVISION" varchar(31),
"LEAGUE" varchar(31),
"NO_ASSTS" float,
"NO_ATBAT" float,
"NO_BB" float,
"NO_ERROR" float,
"NO_HITS" float,
"NO_HOME" float,
"NO_OUTS" float,
"NO_RBI" float,
"NO_RUNS" float,
"YR_MAJOR" float
)
RETURNS varchar(33)
LANGUAGE C
NO SQL
PARAMETER STYLE SQL
DETERMINISTIC
FENCED THREADSAFE
NO EXTERNAL ACTION
ALLOW PARALLEL
NULL CALL
EXTERNAL NAME '/users/db2v9/sqllib/function/SAS/
   dbname_username_baseball1.so!baseball1_em_ eventprobablility '
```

After the scoring functions are installed, they can be invoked in DB2 using SQL, as illustrated in the following example. Each output value is created as a separate function call in the select list.

```
select baseball1_EM_eventprobability
(
"CR_ATBAT",
"CR_BB",
"CR_HITS",
"CR_HOME",
"CR_RBI",
"CR_RUNS",
"DIVISION",
"LEAGUE",
"NO_ASSTS",
"NO_ATBAT",
"NO_BB",
"NO_ERROR",
"NO_HITS",
"NO_HOME",
"NO_OUTS"
) as homeRunProb from MLBDB2;
```

DB2 Permissions

You must have DB2 user permissions to execute the SAS publishing macros to publish the scoring functions. Some of these permissions are as follows.

- EXECUTE user permission for functions that were published by another user
- READ user permission to read the SASUDF_COMPILER_PATH and SASUDF_DB2PATH global variables
- CREATE_EXTERNAL_ROUTINE user permission to the database to create functions
- CREATEIN user permission for the schema in which the scoring functions are published if a nondefault schema is used
- CREATE_NOT_FENCED_ROUTINE user permission to create functions that are not fenced

Permissions must be granted for each user that needs to publish a scoring function and for each database that the scoring model publishing uses. Without these permissions, publishing of the scoring functions fails.

The person who can grant the permissions and the order in which permissions are granted is important. For complete information and examples, see the installation and configuration instructions in the *SAS In-Database Products: Administrator's Guide*.

Scoring Functions inside the DB2 Database

Scoring Function Names

The names of the scoring functions that are built in DB2 have the following format:

`modelname_EM_outputvarname`

modelname is the name that was specified in the MODELNAME argument of the %INDB2_PUBLISH_MODEL macro. *modelname* is always followed by _EM_ in the scoring function name. For more information about the MODELNAME argument, see "%INDB2_PUBLISH_MODEL Macro Syntax" on page 38.

outputvarname is derived from the names of the EM_ output variables in the score.xml file that is generated from the SAS Enterprise Miner Score Code Export node. For more information about the score.xml file, see "Fixed Variable Names" on page 19.

One scoring function is created for each EM_ output variable in the score.xml file. For example, if the scoring model DATA step program takes ten inputs and creates three new variables, then three scoring functions are defined, each with the name of an output variable. For example, if you set MODELNAME=credit in the %INDB2_PUBLISH_MODEL macro and the EM_ output variables are "EM_PREDICTION", "EM_PROBABILITY", and "EM_DECISION", then the name of the scoring functions that are created would be "credit_EM_PREDICTION", "credit_EM_PROBABILITY", and "credit_EM_DECISION".

Note: A scoring function name cannot exceed 128 characters.

CAUTION:

When the scoring function is generated, the names are case insensitive. Consequently, if you have model names "Model01" and "model01", and you create

two scoring functions, the second scoring function overwrites the first scoring function.

Using the Scoring Functions

The scoring functions are available to use in any SQL expression in the same way that DB2 built-in functions are used. For an example, see "Model Publishing Macro Example" on page 41.

There are four ways to see the scoring functions that are created:

- From DB2, log on to the database using the DB2 client tool (command line processor) and submit an SQL statement. The following example assumes that the model name that you used to create the scoring functions is **mymodel** and the DB2 installation instance is located in **`/users/db2v9`**. The first line of code executes a db2profile script. The script sets the DB2 environment variables so the DB2 command line processor (CLP) can execute.

```
>./users/db2v9/sqllib/db2profile
>db2
db2 => connect to database user username using password
db2 => select * from syscat.functions where funcname like '%MYMODEL%'
```

- From SAS, use SQL procedure code that produces output in the LST file. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
proc sql noerrorstop;
  connect to db2 (user=username pw=password db=database);

select *
   from connection to db2
      (select * from syscat.functions where funcname like '%MYMODEL%');
   disconnect from db2;
quit;
```

  You can also use the SASTRACE and SASTRACELOC system options to generate tracing information. For more information about these system options, see the *SAS System Options: Reference*.

- Look at the SampleSQL.txt file that is produced when the %INDB2_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the macro.

  The SampleSQL.txt file contains basic code that, with modifications, can be used to run your score code inside DB2.

  For example, the SampleSQL.txt file refers to an ID column in **allmush1_intab** that is populated with a unique integer from 1 to *n*, with *n* being the number of rows in the table. The ID column uniquely identifies each row. You would replace the ID column with your own primary key column.

  Note: The function and table names must be fully qualified if the function and table are not in the same schema.

  The following example assumes that the model name that you used to create the scoring functions is **allmush1**.

```
drop table allmush1_outtab;
create table allmush1_outtab(
 id integer
```

```
,"EM_CLASSIFICATION" varchar(33)
,"EM_EVENTPROBABILITY" float
,"EM_PROBABILITY" float
);
insert into allmush1_outtab(
 id
,"EM_CLASSIFICATION"
,"EM_EVENTPROBABILITY"
,"EM_PROBABILITY"
)
select id,
 allmush1_em_classification("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_CLASSIFICATION",
 allmush1_em_eventprobability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_EVENTPROBABILITY",
 allmush1_em_probability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
```

```
,"STALKSHA"
,"VEILCOLO")
  as "EM_PROBABILITY"
from allmush1_intab ;
```

- You can look at the SAS log. A message that indicates whether a scoring function is successfully or not successfully executed is printed to the SAS log.

Chapter 6

SAS Scoring Accelerator for Greenplum

Publishing Scoring Model Files in Greenplum **47**

Running the %INDGP_PUBLISH_MODEL Macro **48**
%INDGP_PUBLISH_MODEL Macro Run Process 48
%INDGPPM Macro 49
INDCONN Macro Variable 49
%INDGP_PUBLISH_MODEL Macro Syntax 50
Model Publishing Macro Example 52

Greenplum Permissions **53**

Scoring Functions inside the Greenplum Database **53**
Scoring Function Names 53
Using the Scoring Functions 54

Publishing Scoring Model Files in Greenplum

The SAS publishing macros are used to publish the formats and the scoring functions in Greenplum.

The %INDGP_PUBLISH_MODEL macro creates the files that are needed to build the scoring functions and publishes the scoring functions with those files to a specified database in Greenplum. Only the EM_ output variables are published as Greenplum scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19.

The %INDGP_PUBLISH_MODEL macro uses some of the files that are created by the SAS Enterprise Miner Score Code Export node: the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog.

The %INDGP_PUBLISH_MODEL macro performs the following tasks:

- takes the score.sas and score.xml files and produces the set of .c and .h files. These .c and .h files are necessary to build separate scoring functions for each of a fixed set of quantities that can be computed by the scoring model code.
- if a format catalog is available, processes the format catalog and creates an .h file with C structures, which are also necessary to build the scoring functions.
- produces a script of the Greenplum commands that are used to register the scoring functions on the Greenplum database.
- transfers the .c and .h files to Greenplum.

- calls the SAS_COMPILEUDF function to compile the source files into object files and links to the SAS formats library.
- calls the SAS_COPYUDF function to copy the new object files to ***full-path-to-pkglibdir/SAS*** on the whole database array (master and all segments) , where ***full-path-to-pkglibdir*** is the path that was defined during installation.
- uses the SAS/ACCESS Interface to Greenplum to run the script to create the scoring functions with the object files.

The scoring functions are registered in Greenplum with shared object files, which are loaded at run time. These functions are stored in a permanent location. The SAS object files and the SAS formats library are stored in the ***full-path-to-pkglibdir/SAS*** directory on all nodes, where ***full-path-to-pkglibdir*** is the path that was defined during installation.

Greenplum caches the object files within a session.

Note: You can publish scoring model files with the same model name in multiple databases and schemas. Because all model object files for the SAS scoring function are stored in the ***full-path-to-pkglibdir/SAS*** directory, the publishing macros use the database, schema, and model name as the object filename to avoid potential naming conflicts.

Running the %INDGP_PUBLISH_MODEL Macro

%INDGP_PUBLISH_MODEL Macro Run Process

To run the %INDGP_PUBLISH_MODEL macro, complete the following steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Use the SAS Enterprise Miner Score Code Export node to create a score output directory and populate the directory with the score.sas file, the score.xml file, and, if needed, the format catalog.
3. Start SAS 9.3 and submit one of the following sets of commands in the Program Editor or Enhanced Editor:

```
%indgppm;
%let indconn = user=youruserid password=yourpwd
    dsn=yourdsn;

%indgppm;
%let indconn = user=youruserid password=yourpwd server=yourserver
    database=your db schema=yourschema;
```

   For more information, see "%INDGPPM Macro" on page 49 and "INDCONN Macro Variable" on page 49.

4. Run the %INDGP_PUBLISH_MODEL macro. For more information, see "%INDGP_PUBLISH_MODEL Macro Syntax" on page 50.

   Messages are written to the SAS log that indicate the success or failure of the creation of the scoring functions.

%INDGPPM Macro

The %INDGPPM macro searches the autocall library for the indgppm.sas file. The indgppm.sas file contains all the macro definitions that are used in conjunction with the %INDGP_PUBLISH_MODEL macro. The indgppm.sas file should be in one of the directories listed in the SASAUTOS= system option in your configuration file. If the indgppm.sas file is not present, the %INDGPPM macro call (%INDGPPM; statement) issues the following message:

```
macro indgppm not defined
```

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Greenplum. You must specify user, password, either the DSN or the server and database names, and schema. You must assign the INDCONN macro variable before the %INDGP_PUBLISH_MODEL macro is invoked.

The value of the INDCONN macro variable for the %INDGP_PUBLISH_MODEL macro has one of these formats:

USER=*username* PASSWORD=*password* DSN=*dsnname*

USER=*username* PASSWORD=*password* SERVER=*servername* DATABASE=*databasename* SCHEMA=*schemaname*

Arguments

USER=<'>*username*<'>
specifies the Greenplum user name (also called the user ID) that is used to connect to the database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

PASSWORD=<'>*password*<'>
specifies the password that is associated with your Greenplum user ID. If the password contains spaces or nonalphabetic characters, you must enclose it in quotation marks.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error.

DSN=<'>*datasourcename*<'>
specifies the configured Greenplum ODBC data source to which you want to connect. If the DSN contains spaces or nonalphabetic characters, you must enclose it in quotation marks.

Requirement: You must specify either the DSN= argument or the SERVER= and DATABASE= arguments in the INDCONN macro variable.

SERVER=<'>*servername*<'>
specifies the Greenplum server name or the IP address of the server host. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Requirement: You must specify either the DSN= argument or the SERVER= and DATABASE= arguments in the INDCONN macro variable.

DATABASE=<'>*databasename*<'>
specifies the Greenplum database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Requirement: You must specify either the DSN= argument or the SERVER= and DATABASE= arguments in the INDCONN macro variable.

SCHEMA=<'>*schemaname*<'>
specifies the schema name for the database.

Tip: If you do not specify a value for the SCHEMA argument, the value of the USER argument is used as the schema name. The schema must be created by your database administrator.

TIP The INDCONN macro variable is not passed as an argument to the %INDGP_PUBLISH_MODEL macro. This information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDGP_PUBLISH_MODEL Macro Syntax

%INDGP_PUBLISH_MODEL

(DIR=*input-directory-path*, MODELNAME=*name*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, DATABASE=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DIR=*input-directory-path*
specifies the directory where the scoring model program, the properties file, and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node. This directory contains the score.sas file, the score.xml file, and (if user-defined formats were used) the format catalog.

Requirement: You must use a fully qualified pathname.

Interaction: If you do not use the default directory that is created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, and (if needed) FMTCAT= arguments.

See: "Special Characters in Directory Names" on page 11

MODELNAME=*name*
specifies the name that is prepended to each output function to ensure that each scoring function name is unique on the Greenplum database.

Restriction: The scoring function name is a combination of the model and output variable names. A scoring function name cannot exceed 63 characters. For more information, see "Scoring Function Names" on page 53.

Requirement: The model name must be a valid SAS name that is 10 characters or fewer. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*.

Interaction: Only the EM_ output variables are published as Greenplum scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19 and "Scoring Function Names" on page 53.

DATASTEP=*score-program-filename*
specifies the name of the scoring model program file that was created by using the SAS Enterprise Miner Score Code Export node.

Default: score.sas

Restriction: Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interaction: If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the DATASTEP= argument.

XML=*xml-filename*
specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

Default: score.xml

Restrictions:
Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

The maximum number of output variables is 128.

Interaction: If you use the default score.xml file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the XML= argument.

DATABASE=*database-name*
specifies the name of a Greenplum database to which the scoring functions and formats are published.

Restriction: If you specify DSN= in the INDCONN macro variable, do not use the DATABASE argument.

Interaction: The database that is specified by the DATABASE= argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "%INDGP_PUBLISH_MODEL Macro Run Process" on page 48.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created by the FORMAT procedure and that are referenced in the DATA step scoring model program.

Restriction: Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:
If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the FMTCAT= argument.

If you do not use the default catalog name (FORMATS) or the default library (WORK or LIBRARY) when you create user-defined formats, you must use the FMTSEARCH system option to specify the location of the format catalog. For more information, see PROC FORMAT in the *Base SAS Procedures Guide*.

ACTION=CREATE | REPLACE | DROP
specifies one of the following actions that the macro performs:

CREATE
creates a new function.

REPLACE
overwrites the current function, if a function by the same name is already registered.

DROP
causes all functions for this model to be dropped from the Greenplum database.

Default: CREATE

Tip: If the function has been previously defined and you specify ACTION=CREATE, you receive warning messages from Greenplum. If the function has been previously defined and you specify ACTION=REPLACE, no warnings are issued.

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process and sample SQL code (SampleSQL.txt). For more information about the SampleSQL.txt file, see "Scoring Function Names" on page 53.

Tip: This argument is useful when testing your scoring models.

See: "Special Characters in Directory Names" on page 11

Model Publishing Macro Example

```
%indgppm;
%let indconn = user=user1 password=open1 dsn=green6 schema=myschema;
%indgp_publish_model( dir=C:\SASIN\baseball1, modelname=baseball1, outdir=C:\test);
```

The %INDGP_PUBLISH_MODEL macro produces a text file of Greenplum CREATE FUNCTION commands as shown in the following example.

Note: This example file is shown for illustrative purposes. The text file that is created by the %INDGP_PUBLISH_MODEL macro cannot be viewed and is deleted after the macro is complete.

```
CREATE FUNCTION baseball1_EM_eventprobablility
(
"CR_ATBAT" float,
"CR_BB" float,
"CR_HITS" float,
"CR_HOME" float,
"CR_RBI" float,
"CR_RUNS" float,
"DIVISION" varchar(31),
"LEAGUE" varchar(31),
"NO_ASSTS" float,
"NO_ATBAT" float,
"NO_BB" float,
"NO_ERROR" float,
"NO_HITS" float,
"NO_HOME" float,
"NO_OUTS" float,
"NO_RBI" float,
"NO_RUNS" float,
"YR_MAJOR" float
)
RETURNS varchar(33)
AS '/usr/local/greenplum-db-3.3.4.0/lib/postgresql/SAS/sample_dbitest_homeeq_5.so',
   'homeeq_5_em_classification'
```

After the scoring functions are installed, they can be invoked in Greenplum using SQL, as illustrated in the following example. Each output value is created as a separate function call in the select list.

```
select baseball1_EM_eventprobability
(
"CR_ATBAT",
"CR_BB",
"CR_HITS",
"CR_HOME",
"CR_RBI",
"CR_RUNS",
"DIVISION",
"LEAGUE",
"NO_ASSTS",
"NO_ATBAT",
"NO_BB",
"NO_ERROR",
"NO_HITS",
"NO_HOME",
"NO_OUTS"
) as homeRunProb from MLBGP;
```

Greenplum Permissions

You must have Greenplum superuser permissions to execute the %INDGP_PUBLISH_MODEL macro that publishes the scoring functions. Greenplum requires superuser permissions to create C functions in the database.

Scoring Functions inside the Greenplum Database

Scoring Function Names

The names of the scoring functions that are built in Greenplum have the following format:

*modelname*_EM_*outputvarname*

modelname is the name that was specified in the MODELNAME argument of the %INDGP_PUBLISH_MODEL macro. *modelname* is always followed by _EM_ in the scoring function name. For more information about the MODELNAME argument, see "%INDGP_PUBLISH_MODEL Macro Syntax" on page 50.

outputvarname is derived from the names of the EM_ output variables in the score.xml file that is generated from the SAS Enterprise Miner Score Code Export node. For more information about the score.xml file, see "Fixed Variable Names" on page 19.

One scoring function is created for each EM_ output variable in the score.xml file. For example, if the scoring model DATA step program takes ten inputs and creates three new variables, then three scoring functions are defined, each with the name of an output variable. For example, if you set MODELNAME=credit in the

%INDGP_PUBLISH_MODEL macro and the EM_ output variables are "EM_PREDICTION", "EM_PROBABILITY", and "EM_DECISION", then the name of the scoring functions that are created would be "credit_EM_PREDICTION", "credit_EM_PROBABILITY", and "credit_EM_DECISION".

Note: A scoring function name cannot exceed 63 characters.

CAUTION:

When the scoring function is generated, the names are case insensitive. Consequently, if you have model names "Model01" and "model01", and you create two scoring functions, the second scoring function overwrites the first scoring function.

Using the Scoring Functions

The scoring functions are available to use in any SQL expression in the same way that Greenplum built-in functions are used. For an example, see "Model Publishing Macro Example" on page 52.

TIP In Greenplum, character variables have a length of 32K. If you create an output table or data set to hold the scored rows, it is recommended that you create the table and define the variables. Here is an example.

```
proc sql noerrorstop;
connect to greenplum (<connection options>);
execute (create table scoretab (
   ID                    integer
   , EM_SEGMENT            float
   , EM_EVENTPROBABILITY   float
   , EM_PROBABILITY        float
   , EM_CLASSIFICATION     varchar (32)
   )
   distributed by (id)
) by greenplm;
execute ( insert into scoretab
select id,
function prefix_EM_SEGMENT (
   comma-delimited input column list
   ) as "EM_ SEGMENT",
function prefix_EM_EVENTPROBABILITY (
   comma-delimited input column list
   ) as "EM_EVENTPROBABILITY",
function prefix_EM_PROBABILITY (
   comma-delimited input column list
   ) as "EM_PROBABILITY"
cast(function prefix_EM_CLASSIFICATION (
   comma-delimited input column list
   ) as varchar(32)) as "EM_CLASSIFICATION",
from scoring_input_table
   order by id
) by greenplm;
quit;
```

There are four ways to see the scoring functions that are created:

- From Greenplum, start psql to connect to the database and submit an SQL statement. In this example, 'SCHEMA' is the actual schema value.

```
psql -h hostname -d databasename -U userid
select proname
   from pg_catalog.pg_proc f, pg_catalog.pg_namespace s
   where f.pronamespace=s.oid and upper(s.nspname)='SCHEMA';
```

- From SAS, use SQL procedure code that produces output in the LST file. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
proc sql noerrorstop;
  connect to greenplm (user=username pw=password dsn= dsnname db=database);

select *
   from connection to greenplm
      (select proname
       from pg_catalog.pg_proc f, pg_catalog.pg_namespace s
       where f.pronamespace=s.oid and upper(s.nspname)='SCHEMA');
   disconnect from greenplm;
quit;
```

  You can also use the SASTRACE and SASTRACELOC system options to generate tracing information. For more information about these system options, see the *SAS System Options: Reference*.

- Look at the SampleSQL.txt file that is produced when the %INDGP_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the macro.

  The SampleSQL.txt file contains basic code that, with modifications, can be used to run your score code inside Greenplum.

  For example, the SampleSQL.txt file refers to an ID column in **allmush1_intab** that is populated with a unique integer from 1 to *n*, with *n* being the number of rows in the table. The ID column uniquely identifies each row. You would replace the ID column with your own primary key column.

  Note: The function and table names must be fully qualified if the function and table are not in the same schema.

  The following example assumes that the model name that you used to create the scoring functions is **allmush1**.

```
drop table allmush1_outtab;
create table allmush1_outtab(
 id integer
,"EM_CLASSIFICATION" varchar(33)
,"EM_EVENTPROBABILITY" float
,"EM_PROBABILITY" float
);
insert into allmush1_outtab(
 id
,"EM_CLASSIFICATION"
,"EM_EVENTPROBABILITY"
,"EM_PROBABILITY"
)
select id,
 allmush1_em_classification("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
```

```
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_CLASSIFICATION",
 allmush1_em_eventprobability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_EVENTPROBABILITY",
 allmush1_em_probability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_PROBABILITY"
from allmush1_intab ;
```

- You can look at the SAS log. A message that indicates whether a scoring function is successfully or not successfully executed is printed to the SAS log.

Chapter 7

SAS Scoring Accelerator for Netezza

Publishing Scoring Model Files in Netezza **57**

Running the %INDNZ_PUBLISH_MODEL Macro **58**
%INDNZ_PUBLISH_MODEL Macro Run Process 58
%INDNZPM Macro 58
INDCONN Macro Variable 59
%INDNZ_PUBLISH_MODEL Macro Syntax 59
Modes of Operation 62
Model Publishing Macro Example 62
Netezza Permissions 64

Scoring Functions inside the Netezza Data Warehouse **64**
Scoring Function Names 64
Using the Scoring Functions 65

Publishing Scoring Model Files in Netezza

The SAS publishing macros are used to publish the user-defined formats and the scoring functions in Netezza.

The %INDNZ_PUBLISH_MODEL macro creates the files that are needed to build the scoring functions and publishes those files to a specified database in the Netezza data warehouse. Only the EM_ output variables are published as Netezza scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19.

The %INDNZ_PUBLISH_MODEL macro uses some of the files that are created by the SAS Enterprise Miner Score Code Export node: the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog.

The %INDNZ_PUBLISH_MODEL macro performs the following tasks:

- takes the score.sas and score.xml files that are created using the Score Code Export node and produces the set of .c, .cpp, and .h files that are necessary to build separate scoring functions for each of a fixed set of quantities that can be computed by the scoring model code
- if a format catalog is available, processes the format catalog and creates an .h file with C structures, which are also necessary to build the scoring functions
- produces a script of the Netezza commands that are necessary to register the scoring functions on the Netezza data warehouse

- transfers the .c, .cpp, and .h files to Netezza using the Netezza External Table interface
- calls the SAS_COMPILEUDF function to compile the source files into object files and to access the SAS formats library
- uses the SAS/ACCESS Interface to Netezza to run the script to create the scoring functions with the object files

Running the %INDNZ_PUBLISH_MODEL Macro

%INDNZ_PUBLISH_MODEL Macro Run Process

To run the %INDNZ_PUBLISH_MODEL macro, complete the following steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Use the SAS Enterprise Miner Score Code Export node to create a score output directory and populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS 9.3 and submit the following commands in the Program Editor or Enhanced Editor:

   ```
   %indnzpm;
   %let indconn = server=myserver user=myuserid password=XXXX database=mydb;
   ```

   For more information, see "%INDNZPM Macro" on page 58 and the "INDCONN Macro Variable" on page 59.
4. Run the %INDNZ_PUBLISH_MODEL macro.

   For more information, see "%INDNZ_PUBLISH_MODEL Macro Syntax" on page 59.

   Messages are written to the SAS log that indicate the success or failure of the creation of the scoring functions.

Note: The %INDNZ_PUBLISH_JAZLIB macro and the %INDNZ_PUBLISH_COMPILEUDF macro, if needed, must be run before you can publish your scoring models. Otherwise, the %INDNZ_PUBLISH_MODEL macro fails. These macros are typically run by your system or database administrator. For more information about these macros, see *SAS In-Database Products: Administrator's Guide*.

%INDNZPM Macro

The %INDNZPM macro searches the autocall library for the indnzpm.sas file. The indnzpm.sas file contains all the macro definitions that are used in conjunction with the %INDNZ_PUBLISH_MODEL macro. The indnzpm.sas file should be in one of the directories listed in the SASAUTOS= system option in your configuration file. If the indnzpm.sas file is not present, the %INDNZPM macro call (%INDNZPM; statement) issues the following message:

```
macro indnzpm not defined
```

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Netezza. You must specify server, user, password, and database information to access the machine on which you have installed the Netezza data warehouse. You must assign the INDCONN macro variable before the %INDNZ_PUBLISH_MODEL macro is invoked.

Here is the syntax for the value of the INDCONN macro variable for the %INDNZ_PUBLISH_MODEL macro:

SERVER=*server*USER=*user* PASSWORD=*password* DATABASE=*database*

Arguments

SERVER=*server*
specifies the Netezza server name or the IP address of the server host.

USER=*user*
specifies the Netezza user name (also called the user ID) that is used to connect to the database.

PASSWORD=*password*
specifies the password that is associated with your Netezza user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error to occur.

DATABASE=*database*
specifies the Netezza database that contains the tables and views that you want to access.

TIP The INDCONN macro variable is not passed as an argument to the %INDNZ_PUBLISH_MODEL macro. This information can be concealed in your SAS job. For example, you can place it in an autoexec file and apply permissions to the file so others cannot access the user credentials.

%INDNZ_PUBLISH_MODEL Macro Syntax

%INDNZ_PUBLISH_MODEL

(DIR=*input-directory-path*, MODELNAME=*name*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, DATABASE=*database-name*>
<, DBCOMPILE=*database-name*>
<, DBJAZLIB=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP >
<, MODE=FENCED | UNFENCED>
<, IDCASE=UPPERCASE | LOWERCASE >
<, OUTDIR=*diagnostic-output-directory*>
);

Note: Do not enclose variable arguments in single or double quotation marks. This causes the %INDNZ_PUBLISH_MODEL macro to fail.

Arguments

DIR=*input-directory-path*
specifies the directory where the scoring model program, the properties file, and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node. This directory contains the score.sas file, the score.xml file, and (if user-defined formats were used) the format catalog.

Requirement: You must use a fully qualified pathname.

Interaction: If you do not use the default directory that is created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, and, if needed, FMTCAT= arguments.

See: "Special Characters in Directory Names" on page 11

MODELNAME=*name*
specifies the name that is prepended to each output function to ensure that each scoring function name is unique on the Netezza database.

Restriction: The scoring function name is a combination of the model and output variable names. A scoring function name cannot exceed 128 characters. For more information, see "Scoring Function Names" on page 64.

Requirement: The model name must be a valid SAS name that is ten characters or fewer. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*

Interaction: Only the EM_ output variables are published as Netezza scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19 and "Scoring Function Names" on page 64.

DATASTEP=*score-program-filename*
specifies the name of the scoring model program file that was created by using the SAS Enterprise Miner Score Code Export node.

Default: score.sas

Restriction: Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interaction: If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the DATASTEP= argument.

XML=*xml-filename*
specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

Default: score.xml

Restrictions:
Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

The maximum number of output variables is 128.

Interaction: If you use the default score.xml file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the XML= argument.

DATABASE=*database-name*
specifies the name of a Netezza database to which the scoring functions and formats are published.

Interaction: The database that is specified by the DATABASE argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "%INDNZ_PUBLISH_MODEL Macro Run Process" on page 58.

Tip: You can publish the scoring functions and formats to a shared database where other users can access them.

DBCOMPILE=*database-name*
specifies the name of the database where the SAS_COMPILEUDF function is published.

Default: SASLIB

See: For more information about publishing the SAS_COMPILEUDF function, see the *SAS In-Database Products: Administrator's Guide*.

DBJAZLIB=*database-name*
specifies the name of the database where the SAS formats library is published.

Default: SASLIB

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created by the FORMAT procedure and that are referenced in the DATA step scoring model program.

Default:

Restriction: Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:
If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the FMTCAT= argument.

If you do not use the default catalog name (FORMATS) or the default library (WORK or LIBRARY) when you create user-defined formats, you must use the FMTSEARCH system option to specify the location of the format catalog. For more information, see PROC FORMAT in the *Base SAS Procedures Guide*.

ACTION=CREATE | REPLACE | DROP
specifies one of the following actions that the macro performs:

CREATE
creates a new function.

REPLACE
overwrites the current function, if a function by the same name is already registered.

DROP
causes all functions for this model to be dropped from the Netezza database.

Default: CREATE

Tip: If the function was published previously and you specify ACTION=CREATE, you receive warning messages that the function already exists and you are prompted to use REPLACE. If you specify ACTION=DROP and the function does not exist, an error message is issued.

MODE= FENCED | UNFENCED
specifies whether running the code is isolated in a separate process in the Netezza database so that a program fault does not cause the database to stop.

Default: FENCED

Restriction: The MODE= argument is supported for Netezza 6.0. The MODE argument is ignored for previous versions of Netezza.

See: "Modes of Operation" on page 62

IDCASE= UPPERCASE | LOWERCASE
specifies whether the variable names in the generated sample SQL code (SampleSQL.txt) appear in uppercase or lowercase characters.

Default: UPPERCASE

Tip: When you specify the IDCASE argument, the %INDNZ_PUBLISH_MODEL macro first determines which release of Netezza is being used. If Netezza release 5.0 or later is being used, the macro then checks to see whether the LOWERCASE or UPPERCASE option is set for the database by using SQL statement SELECT IDENTIFIER_CASE. If the value of the IDCASE argument is different from the case configuration of the database, the macro overwrites the value of the IDCASE option and uses the case configuration of the database. If an earlier release of Netezza is being used, the macro uses the value of the IDCASE argument.

See: "Using the Scoring Functions" on page 65 for more information about the SampleSQL.txt file

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process and sample SQL code (SampleSQL.txt). For more information about the SampleSQL.txt file, see "Scoring Function Names" on page 64.

Tip: This argument is useful when testing your scoring models.

See: "Special Characters in Directory Names" on page 11

Modes of Operation

The %INDNZ_PUBLISH_MODEL macro has two modes of operation: fenced and unfenced. You specify the mode by setting the MODE= argument.

The default mode of operation is fenced. Fenced mode means that the scoring function that is published is isolated in a separate process in the Netezza database when it is invoked, and an error does not cause the database to stop. It is recommended that you publish the scoring functions in fenced mode during acceptance tests.

When the scoring function is ready for production, you can run the macro to publish the scoring function in unfenced mode. You could see a performance advantage if the scoring function is published in unfenced mode.

Note: The MODE= argument is supported for Netezza 6.0. The MODE argument is ignored for previous versions of Netezza.

Model Publishing Macro Example

```
%indnzpm;
%let indconn = server=netezbase user=user1 password=open1 database=mydb;
%indnz_publish_model(dir=C:\SASIN\baseball1, modelname=baseball1);
```

This sequence of macros generates a separate .c file for each output parameter of interest. Each output stub calls into a shared scoring main that is compiled first. The %INDNZ_PUBLISH_MODEL macro also produces a text file of Netezza CREATE FUNCTION commands as shown in the following example.

Note: This example file is shown for illustrative purposes. The text file that is created by the %INDNZ_PUBLISH_MODEL macro cannot be viewed and is deleted after the macro is complete.

```
CREATE FUNCTION baseball1_EM_eventprobablility
(
"CR_ATBAT" float,
"CR_BB" float,
"CR_HITS" float,
"CR_HOME" float,
"CR_RBI" float,
"CR_RUNS" float,
"DIVISION" varchar(31),
"LEAGUE" varchar(31),
"NO_ASSTS" float,
"NO_ATBAT" float,
"NO_BB" float,
"NO_ERROR" float,
"NO_HITS" float,
"NO_HOME" float,
"NO_OUTS" float,
"NO_RBI" float,
"NO_RUNS" float,
"YR_MAJOR" float
)
RETURNS float
LANGUAGE CPP
PARAMETER STYLE npsgeneric
CALLED ON NULL INPUT
EXTERNAL CLASS NAME 'Cbaseball1_em_eventprobability'
EXTERNAL HOST OBJECT '/tmp/tempdir_20090506T113450_316550/baseball1.o_x86'
EXTERNAL NSPU OBJECT '/tmp/tempdir_20090506T113450_316550/baseball1.o_diab_ppc';
DEPENDENCIES dbjazlib..sas_jazlib  /* if TwinFin system */
```

After the scoring functions are installed, they can be invoked in Netezza using SQL, as illustrated in the following example. Each output value is created as a separate function call in the select list.

```
select baseball1_EM_eventprobability
(
"CR_ATBAT",
"CR_BB",
"CR_HITS",
"CR_HOME",
"CR_RBI",
"CR_RUNS",
"DIVISION",
"LEAGUE",
"NO_ASSTS",
"NO_ATBAT",
"NO_BB",
"NO_ERROR",
"NO_HITS",
"NO_HOME",
"NO_OUTS"
) as homeRunProb from MLBNetz;
```

Netezza Permissions

You must have permission to create scoring functions and tables in the Netezza database. You must also have permission to execute the SAS_COMPILEUDF, SAS_DIRECTORYUDF, and SAS_HEXTOTEXTUDF functions in either the SASLIB database or the database specified in lieu of SASLIB where these functions are published.

Without these permissions, the publishing of a scoring function fails. To obtain these permissions, contact your database administrator.

For more information about specific permissions, see the *SAS In-Database Products: Administrator's Guide*.

Scoring Functions inside the Netezza Data Warehouse

Scoring Function Names

The names of the scoring functions that are built in Netezza have the following format:

`modelname_EM_outputvarname`

modelname is the name that was specified in the MODELNAME argument of the %INDNZ_PUBLISH_MODEL macro. *modelname* is always followed by _EM_ in the scoring function name. For more information about the MODELNAME argument, see "Running the %INDNZ_PUBLISH_MODEL Macro" on page 58.

outputvarname is derived from the names of the EM_ output variables in the score.xml file that is generated from the SAS Enterprise Miner Score Code Export node. For more information about the score.xml file, see "Fixed Variable Names" on page 19.

One scoring function is created for each EM_ output variable in the score.xml file. For example, if the scoring model DATA step program takes ten inputs and creates three new variables, then three scoring functions are defined. Each scoring function has the name of an output variable. For example, if you set MODELNAME=credit in the %INDNZ_PUBLISH_MODEL macro, and the EM_ output variables are "EM_PREDICTION", "EM_PROBABILITY", and "EM_DECISION", then the name of the scoring functions that are created would be "credit_EM_PREDICTION", "credit_EM_PROBABILITY", and "credit_EM_DECISION".

Note: Scoring function names cannot exceed 128 characters.

CAUTION:

When the scoring function is generated, the names are case insensitive. Consequently, if you have model names "Model01" and "model01", and you create two scoring functions, the second scoring function overwrites the first scoring function.

Using the Scoring Functions

The scoring functions are available to use in any SQL expression in the same way that Netezza built-in functions are used. For an example, see "Model Publishing Macro Example" on page 62.

There are four ways to see the scoring functions that are created:

- From Netezza, log on to the database using a client tool such as NZSQL and submit an SQL statement. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
nzsql database username password

 select function,createdate,functionsignature from _v_function where
   function like '%MYMODEL%'
```

- From SAS, use SQL procedure code that produces output in the LST file. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
proc sql noerrorstop;
  connect to netezza (server=servername database=database
    username=username password=password);
  select *
     from connection to netezza
     (select function,createdate,functionsignature
      from _v_function where
       function like '%MYMODEL%');
   disconnect from netezza;
quit;
```

  You can also use the SASTRACE and SASTRACELOC system options to generate tracing information. For more information about these system options, see the *SAS System Options: Reference*.

- Look at the SampleSQL.txt file that is produced when the %INDNZ_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the macro.

  The SampleSQL.txt file contains basic code that, with modifications, can be used to run your score code inside Netezza.

  For example, the SampleSQL.txt file refers to an ID column in **allmush1_intab** that is populated with a unique integer from 1 to *n*, with *n* being the number of rows in the table. The ID column uniquely identifies each row. You would replace the ID column with your own primary key column.

  Note: The function and table names must be fully qualified if the function and table are not in the same database.

  The following example assumes that the model name that you used to create the scoring functions is **allmush1**.

```
drop table allmush1_outtab;
create table allmush1_outtab(
 id integer
,"EM_CLASSIFICATION" varchar(33)
,"EM_EVENTPROBABILITY" float
,"EM_PROBABILITY" float
```

```
);
insert into allmush1_outtab(
 id
,"EM_CLASSIFICATION"
,"EM_EVENTPROBABILITY"
,"EM_PROBABILITY"
)
select id,
 allmush1_em_classification("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_CLASSIFICATION",
 allmush1_em_eventprobability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_EVENTPROBABILITY",
 allmush1_em_probability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
```

```
  as "EM_PROBABILITY"
from allmush1_intab ;
```

- You can look at the SAS log. A message that indicates whether a scoring function is successfully or not successfully created or replaced is printed to the SAS log.

Chapter 8

SAS Scoring Accelerator for Teradata

Publishing Scoring Model Files in Teradata **69**

Running the %INDTD_PUBLISH_MODEL Macro **70**
%INDTD_PUBLISH_MODEL Macro Run Process 70
%INDTDPM Macro 70
INDCONN Macro Variable 70
%INDTD_PUBLISH_MODEL Macro Syntax 71
Modes of Operation 73
Model Publishing Example 73
Teradata Permissions 75

Scoring Functions inside the Teradata EDW **75**
Scoring Function Names 75
Using the Scoring Functions 76

Publishing Scoring Model Files in Teradata

The %INDTD_PUBLISH_MODEL macro creates the files that are needed to build the scoring functions and publishes these files to a specified database in the Teradata EDW. Only the EM_ output variables are published as Teradata scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19.

The %INDTD_PUBLISH_MODEL macro uses some of the files that are created by the SAS Enterprise Miner Score Code Export node: the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog.

The %INDTD_PUBLISH_MODEL macro takes the score.sas and score.xml files and produces the set of .c and .h files. These .c and .h files are necessary to build separate scoring functions for each of a fixed set of quantities that can be computed by the scoring model code.

If a format catalog is available, the %INDTD_PUBLISH_MODEL macro processes the format catalog and creates an .h file with C structures. This file is also necessary to build the scoring functions.

This macro also produces a script of the Teradata commands that are used to register the scoring functions on the Teradata EDW.

After this script is created, the macro uses SAS/ACCESS Interface to Teradata to run the script and publish the scoring model files to the Teradata EDW.

Running the %INDTD_PUBLISH_MODEL Macro

%INDTD_PUBLISH_MODEL Macro Run Process

To run the %INDTD_PUBLISH_MODEL macro, complete the following steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Use the SAS Enterprise Miner Score Code Export node to create a score output directory and populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Test your connection to Teradata with a local utility such as BTEQ.
4. Start SAS 9.3 and submit the following commands in the Program Editor or Enhanced Editor:

   ```
   %indtdpm;
   %let indconn = server="myserver" user="myuserid" password="xxxx"
       database="mydb";
   ```

   For more information, see "%INDTDPM Macro" on page 70 and the "INDCONN Macro Variable" on page 70.
5. Run the %INDTD_PUBLISH_MODEL macro.

   Messages are written to the SAS log that indicate the success or failure of the creation of the scoring functions.

   For more information, see "%INDTD_PUBLISH_MODEL Macro Syntax" on page 71.

Note: USER librefs that are not assigned to the WORK library might cause unexpected or unsuccessful behavior.

%INDTDPM Macro

The %INDTDPM macro searches the autocall library for the indtdpm.sas file. The indtdpm.sas file contains all the macro definitions that are used in conjunction with the %INDTD_PUBLISH_MODEL macro. The indtdpm.sas file should be in one of the directories listed in the SASAUTOS= system option in your configuration file. If the indtdpm.sas file is not present, the %INDTDPM macro call (%INDTDPM; statement) issues the following message:

```
macro indtdpm not defined
```

INDCONN Macro Variable

The INDCONN macro variable is used to provide the credentials to connect to Teradata. You must specify server, user, password, and database to access the machine on which you have installed the Teradata EDW. You must assign the INDCONN macro variable before the %INDTD_PUBLISH_MODEL macro is invoked.

Here is the syntax for the value of the INDCONN macro variable:

SERVER="*server* " USER="*user* " PASSWORD="*password*"
DATABASE="*database*";

Arguments

SERVER="*server*"
specifies the Teradata server name or the IP address of the server host.

USER="*user*
specifies the Teradata user name (also called the user ID) that is used to connect to the database.

PASSWORD="*password*
specifies the password that is associated with your Teradata user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error to occur.

DATABASE="*database*"
specifies the Teradata database that contains the tables and views that you want to access.

TIP The INDCONN macro variable is not passed as an argument to the %INDTD_PUBLISH_MODEL macro. This information can be concealed in your SAS job. For example, you can place it in an autoexec file and apply permissions on that file so that others cannot access the user credentials.

%INDTD_PUBLISH_MODEL Macro Syntax

%INDTD_PUBLISH_MODEL

(DIR=*input-directory-path*, MODELNAME=*name*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, DATABASE=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
<, MODE=PROTECTED | UNPROTECTED>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DIR=*input-directory-path*
specifies the directory where the scoring model program, the properties file, and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node. This directory contains the score.sas file, the score.xml file, and (if user-defined formats were used) the format catalog.

Restriction: You must use a fully qualified pathname.

Interaction: If you do not use the default directory that is created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, and (if needed) FMTCAT= arguments.

See: "Special Characters in Directory Names" on page 11

MODELNAME=*name*
specifies the name that is prepended to each output function to ensure that each scoring function name is unique on the Teradata database.

Restriction: The scoring function name is a combination of the model and the output variable names. The scoring function name cannot exceed 30 characters. For more information, see "Scoring Function Names" on page 75.

Requirement: The model name must be a valid SAS name that is ten characters or fewer. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*.

Interaction: Only the EM_ output variables are published as Teradata scoring functions. For more information about the EM_ output variables, see "Fixed Variable Names" on page 19 and "Scoring Function Names" on page 75.

DATASTEP=*score-program-filename*
specifies the name of the scoring model program file that was created by using the SAS Enterprise Miner Score Code Export node.

Default: score.sas

Restriction: Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interaction: If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the DATASTEP= argument.

XML=*xml-filename*
specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

Default: score.xml

Restrictions:
Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

The maximum number of output variables is 128.

Interaction: If you use the default score.xml file that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the XML= argument.

DATABASE=*database-name*
specifies the name of a Teradata database to which the scoring functions and formats are published.

Requirement:

Interaction: The database that is specified by the DATABASE argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "%INDTD_PUBLISH_MODEL Macro Run Process" on page 70.

Tip: You can publish the scoring functions and formats to a shared database where other users can access them.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created by the FORMAT procedure and that are referenced in the DATA step scoring model program.

Restriction: Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.

Interactions:
If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node, you do not need to specify the FMTCAT= argument.

If you do not use the default catalog name (FORMATS) or the default library (WORK or LIBRARY) when you create user-defined formats, you must use the

FMTSEARCH system option to specify the location of the format catalog. For more information, see PROC FORMAT in the *Base SAS Procedures Guide*.

ACTION=CREATE | REPLACE | DROP
specifies one of the following actions that the macro performs:

CREATE
creates a new function.

REPLACE
overwrites the current function, if a function with the same name is already registered.

DROP
causes all functions for this model to be dropped from the Teradata database.

Default: CREATE

Tip: If the function has been previously defined and you specify ACTION=CREATE, you will receive warning messages from Teradata. If the function has been previously defined and you specify ACTION=REPLACE, no warnings are issued.

MODE=PROTECTED | UNPROTECTED
specifies whether the running code is isolated in a separate process in the Teradata database so that a program fault does not cause the database to stop.

Default: PROTECTED

Tip: After a function is validated in PROTECTED mode, it can be republished in UNPROTECTED mode. This could result in a significant performance gain.

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process and sample SQL code (SampleSQL.txt). For more information about the SampleSQL.txt file, see "Scoring Function Names" on page 75.

Tip: This argument is useful when testing your scoring models.

See: "Special Characters in Directory Names" on page 11

Modes of Operation

The %INDTD_PUBLISH_MODEL macro has two modes of operation: protected and unprotected. You specify the mode by setting the MODE= argument.

The default mode of operation is protected. Protected mode means that the macro code is isolated in a separate process in the Teradata database, and any error does not cause the database to stop. It is recommended that you run the %INDTD_PUBLISH_MODEL macro in protected mode during acceptance tests.

When the %INDTD_PUBLISH_MODEL macro is ready for production, you can run the macro in unprotected mode. Note that you could see a performance advantage when you run in unprotected mode.

Model Publishing Example

```
%indtdpm;
%let indconn = server="terabase" user="user1" password="open1" database="mydb";
%indtd_publish_model( dir=C:\SASIN\baseball1, modelname=baseball1);
```

This sequence of macros generates a separate .c file for each output parameter of interest. Each output stub calls into a shared scoring main that is compiled first. The %INDTD_PUBLISH_MODEL macro also produces a text file of Teradata CREATE FUNCTION commands as shown in the following example.

Note: This file is shown for illustrative purposes. The text file that is created by the %INDTD_PUBLISH_MODEL macro cannot be viewed and is deleted after the macro is complete.

```
CREATE FUNCTION baseball1_EM_eventprobablility
(
"CR_ATBAT" float,
"CR_BB" float,
"CR_HITS" float,
"CR_HOME" float,
"CR_RBI" float,
"CR_RUNS" float,
"DIVISION" varchar(31),
"LEAGUE" varchar(31),
"NO_ASSTS" float,
"NO_ATBAT" float,
"NO_BB" float,
"NO_ERROR" float,
"NO_HITS" float,
"NO_HOME" float,
"NO_OUTS" float,
"NO_RBI" float,
"NO_RUNS" float,
"YR_MAJOR" float
)
RETURNS float
LANGUAGE C
NO SQL
PARAMETER STYLE SQL
NOT DETERMINISTIC
CALLED ON NULL INPUT
EXTERNAL NAME 'SL!"jazxfbrs"'
'!CI!tkcsparm!c:\SASIN\baseball1\tkcsparm.h'
'!CS!baseball1_EM_eventprobability!c:\SASIN\baseball1\EM_eventprobability.c';
```

After the scoring functions are installed, they can be invoked in Teradata using SQL, as illustrated in the following example. Each output value is created as a separate function call in the select list.

```
select baseball1_EM_eventprobability
(
"CR_ATBAT",
"CR_BB",
"CR_HITS",
"CR_HOME",
"CR_RBI",
"CR_RUNS",
"DIVISION",
"LEAGUE",
"NO_ASSTS",
"NO_ATBAT",
"NO_BB",
"NO_ERROR",
```

```
"NO_HITS",
"NO_HOME",
"NO_OUTS"
) as homeRunProb from MLBTera;
```

Teradata Permissions

Because functions are associated with a database, the functions inherit the access rights of that database. It could be useful to create a separate shared database for scoring functions so that access rights can be customized as needed.

In addition, to publish the scoring functions in Teradata, you must have the following permissions on the database where the functions are published:

CREATE FUNCTION
DROP FUNCTION
EXECUTE FUNCTION
ALTER FUNCTION

To obtain database permissions, contact your database administrator.

Scoring Functions inside the Teradata EDW

Scoring Function Names

The names of the scoring functions that are built in Teradata have the following format:

*modelname*_EM_*outputvarname*

modelname is the name that was specified in the MODELNAME argument of the %INDTD_PUBLISH_MODEL macro. *modelname* is always followed by _EM_ in the scoring function name. For more information about the MODELNAME argument, see "Running the %INDTD_PUBLISH_MODEL Macro" on page 70.

outputvarname is derived from the names of the EM_ output variables in the score.xml file that is generated from the SAS Enterprise Miner Score Code Export node. For more information about the score.xml file, see "Fixed Variable Names" on page 19.

One scoring function is created for each EM_ output variable in the score.xml file. For example, if the scoring model DATA step program takes ten inputs and creates three new variables, then three scoring functions are defined, each with the name of an output variable. For example, if you set MODELNAME=credit in the %INDTD_PUBLISH_MODEL macro, and the EM_ output variables are "EM_PREDICTION", "EM_PROBABILITY", and "EM_DECISION", then the name of the scoring functions that are created would be "credit_EM_PREDICTION", "credit_EM_PROBABILITY", and "credit_EM_DECISION".

Note: The scoring function name cannot exceed 30 characters.

CAUTION:

When the scoring function is generated, the names are case insensitive. Consequently, if you have model names "Model01" and "model01", and you create two scoring functions, the second scoring function overwrites the first scoring function.

Using the Scoring Functions

The scoring functions are available to use in any SQL expression in the same way that Teradata built-in functions are used. For an example, see "Model Publishing Example" on page 73.

There are four ways to see the scoring functions that are created:

- From Teradata, log on to the database using a client tool such as BTEQ and submit an SQL statement. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
bteq .logon myserver/myuserid,mypassword;
  select * from dbc.tables where tablename like '%mymodel%';
```

- From SAS, use SQL procedure code that produces output in the LST file. The following example assumes that the model name that you used to create the scoring functions is **mymodel**.

```
proc sql noerrorstop;
  connect to teradata (user=user password=pass server=server);
  select *
     from connection to teradata
     (select tablename,tablekind,databasename,LastALterTimeStamp
      from dbc.tables where
      databasename='sas' and tablename like '%mymodel%'
      and tablekind='F');

disconnect teradata;
quit;
```

  You can also use the SASTRACE and SASTRACELOC system options to generate tracing information. For more information about these system options, see the *SAS System Options: Reference*.

- Look at the SampleSQL.txt file that is produced when the %INDTD_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the macro.

  The SampleSQL.txt file contains basic code that, with modifications, can be used to run your score code inside Teradata.

  For example, the SampleSQL.txt file refers to an ID column in **allmush1_intab** that is populated with a unique integer from 1 to *n*, with *n* being the number of rows in the table. The ID column uniquely identifies each row. You would replace the ID column with your own primary key column.

  Note: The function and table names must be fully qualified if the functions and tables are not in the same database.

  The following example assumes that the model name that you used to create the scoring functions is **allmush1**.

```
drop table allmush1_outtab;
create table allmush1_outtab(
 id integer
,"EM_CLASSIFICATION" varchar(33)
,"EM_EVENTPROBABILITY" float
,"EM_PROBABILITY" float
);
```

```
insert into allmush1_outtab(
 id
,"EM_CLASSIFICATION"
,"EM_EVENTPROBABILITY"
,"EM_PROBABILITY"
)
select id,
 allmush1_em_classification("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_CLASSIFICATION",
 allmush1_em_eventprobability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_EVENTPROBABILITY",
 allmush1_em_probability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCBR"
,"STALKROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_PROBABILITY"
from allmush1_intab ;
```

- You can look at the SAS log. A message is printed to the SAS log that states whether a scoring function is successfully or not successfully created or replaced.

Chapter 9

SAS Scoring Accelerator and SAS Model Manager

Using the SAS Scoring Accelerator with SAS Model Manager 79

Using the SAS Scoring Accelerator with SAS Model Manager

You can use SAS Scoring Accelerator in conjunction with the SAS Model Manager to manage and deploy scoring models in DB2, Netezza, and Teradata.

The **Publish Scoring Function** of SAS Model Manager enables you to publish classification and prediction model types to the database. When you publish a DB2, Netezza, or Teradata scoring function for a project, SAS Model Manager exports the project's champion model to the SAS Metadata Repository and calls the SAS Scoring Accelerator to create the scoring functions. The scoring functions are deployed inside the database based on the project's champion model score code. The scoring function is then validated automatically against a default train table to ensure that the scoring results are correct. A scoring application (for example, a call center application that calls the SAS Model Manager Java Scoring API) can then execute the scoring functions in the database. The scoring functions extend the database's SQL language and can be used on SQL statements like other database functions.

For more information, see the *SAS Model Manager: User's Guide*.

Part 3

Format Publishing and the SAS_PUT() Function

Chapter 10
Deploying and Using SAS Formats inside the Database *83*

Chapter 11
Deploying and Using SAS Formats in DB2 under UNIX *91*

Chapter 12
Deploying and Using SAS Formats in Netezza *103*

Chapter 13
Deploying and Using SAS Formats in Teradata *113*

Chapter 10

Deploying and Using SAS Formats inside the Database

Using SAS Formats and the SAS_PUT() Function **83**
How It Works **84**
Special Characters in Directory Names **86**
Considerations and Limitations with User-Defined Formats **87**
Tips for Using the Format Publishing Macros **88**
Tips for Using the SAS_PUT() Function **88**
Determining Format Publish Dates **88**

Using SAS Formats and the SAS_PUT() Function

SAS formats are basically mapping functions that change an element of data from one format to another. For example, some SAS formats change numeric values to various currency formats or date-and-time formats.

SAS supplies many formats. You can also use the SAS FORMAT procedure to define custom formats that replace raw data values with formatted character values. For example, this PROC FORMAT code creates a custom format called $REGION that maps ZIP codes to geographic regions.

```
proc format;
   value $region
   '02129', '03755', '10005' = 'Northeast'
   '27513', '27511', '27705' = 'Southeast'
   '92173', '97214', '94105' = 'Pacific';
run;
```

SAS programs, including in-database procedures, frequently use both user-defined formats and formats that SAS supplies. Although they are referenced in numerous ways, using the PUT function in the SQL procedure is of particular interest for SAS In-Database processing.

The PUT function takes a format reference and a data item as input and returns a formatted value. This SQL procedure query uses the PUT function to summarize sales by region from a table of all customers:

```
select put(zipcode,$region.) as region,
   sum(sales) as sum_sales from sales.customers
   group by region;
```

The SAS SQL processor knows how to process the PUT function. Currently, SAS/ACCESS Interface to the database returns all rows of unformatted data in the SALES.CUSTOMERS table in the database to the SAS System for processing.

The SAS In-Database technology deploys, or publishes, the PUT function implementation to the database as a new function named SAS_PUT(). Similar to any other programming language function, the SAS_PUT() function can take one or more input parameters and return an output value.

The SAS_PUT() function supports use of SAS formats. You can specify the SAS_PUT() function in SQL queries that SAS submits to the database in one of two ways:

- implicitly by enabling SAS to automatically map PUT function calls to SAS_PUT() function calls
- explicitly by using the SAS_PUT() function directly in your SAS program

If you used the SAS_PUT() function in the previous SELECT statement, the database formats the ZIP code values with the $REGION format and processes the GROUP BY clause using the formatted values.

By publishing the PUT function implementation to the database as the SAS_PUT() function, you can realize these advantages:

- You can process the entire SQL query inside the database, which minimizes data transfer (I/O).
- The SAS format processing leverages the scalable architecture of the DBMS.
- The results are grouped by the formatted data and are extracted from the database.

Deploying SAS formats to execute inside a database can enhance performance and exploit the database's parallel processing.

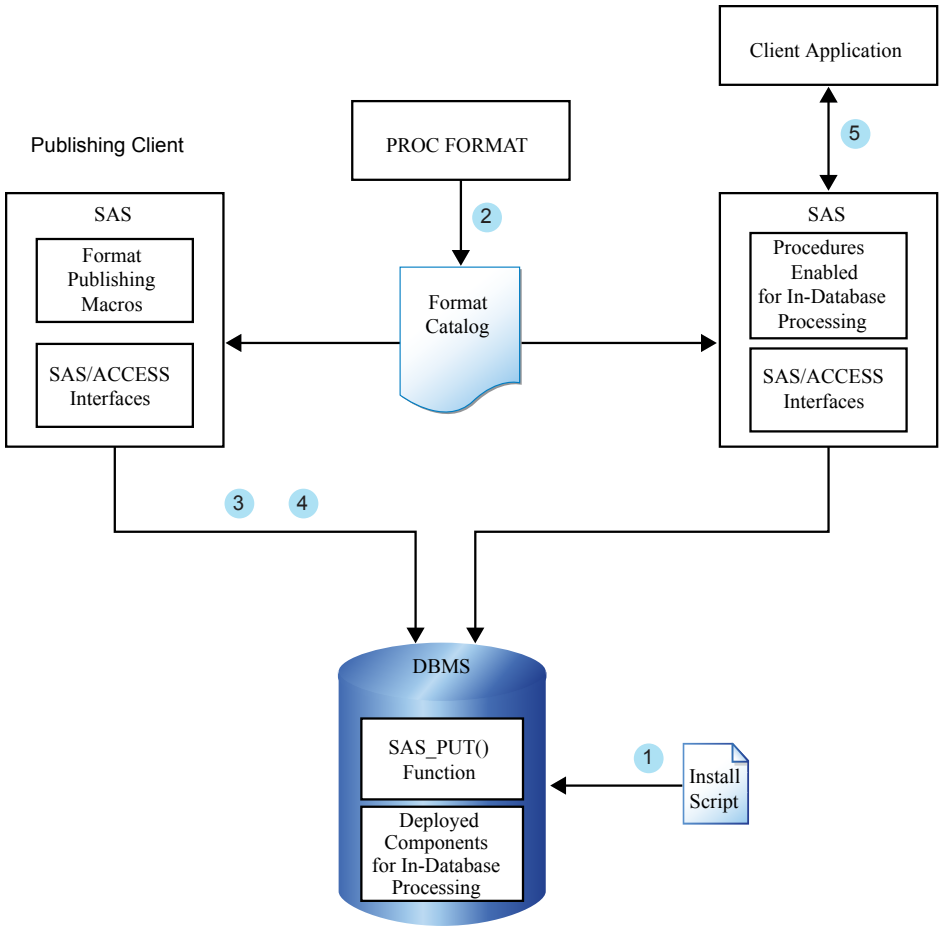
Note: SAS formats and the SAS_PUT() functionality is available in DB2, Netezza, and Teradata.

How It Works

By using the SAS formats publishing macro, you can generate a SAS_PUT() function that enables you to execute PUT function calls inside the database. You can reference the formats that SAS supplies and most custom formats that you create by using PROC FORMAT.

The SAS formats publishing macro takes a SAS format catalog and publishes it to the database. Inside the database, a SAS_PUT() function, which emulates the PUT function, is created and registered for use in SQL queries.

Figure 10.1 *Process Flow Diagram*



Here is the basic process flow.

1 Install the components that are necessary for in–database processing.

   For more information, see "Deployed Components for In-Database Processing" on page 4.

   Note: This is a one-time installation process.

2 If necessary, create your custom formats by using PROC FORMAT and create a permanent catalog by using the LIBRARY= option.

   For more information, see the topic on user-defined formats in the section for your database.

3 Start SAS 9.3 and run the format publishing macro. This macro creates the files that are needed to build the SAS_PUT() function and publishes those files to the database.

   For more information, see the topic on publishing SAS formats in the section for your database.

4 After the format publishing macro creates the script, SAS/ACCESS Interface to Teradata executes the script and publishes the files to the database.

   For more information, see the topic on publishing SAS formats in the section for your database.

5 The SAS_PUT() function is available to use in any SQL expression and to use typically wherever you use your database's built-in functions.

For more information, see the topic on using the SAS_PUT() function in the section for your database.

Special Characters in Directory Names

If the directory names that are used in the macros contain any of the following special characters, you must mask the characters by using the %STR macro quoting function. For more information, see the %STR function and macro string quoting topic in *SAS Macro Language: Reference*.

Table 10.1 *Special Characters in Directory Names*

Character	How to Represent
blank*	%str()
***	%str(*)
;	%str(;)
, (comma)	%str(,)
=	%str(=)
+	%str(+)
-	%str(–)
>	%str(>)
<	%str(<)
^	%str(^)
|	%str(|)
&	%str(&)
#	%str(#)
/	%str(/)
~	%str(~)
%	%str(%%)
'	%str(%')
"	%str(%")
(	%str(%()

Character	How to Represent
)	%str(%))
¬	%str(¬)

* Only leading blanks require the %STR function, but you should avoid using leading blanks in directory names.

** Asterisks (*) are allowed in UNIX directory names. Asterisks are not allowed in Windows directory names. In general, avoid using asterisks in directory names.

Here are some examples of directory names with special characters:

Table 10.2 *Examples of Special Characters in Directory Names*

Directory	Code Representation
`c:\temp\Sales(part1)`	`c:\temp\Sales%str(%()part1%str(%))`
`c:\temp\Drug "trial" X`	`c:\temp\Drug %str(%")trial(%str(%") X`
`c:\temp\Disc's 50% Y`	`c:\temp\Disc%str(%')s 50%str(%%) Y`
`c:\temp\Pay,Emp=Z`	`c:\temp\Pay%str(,)Emp%str(=)Z`

Considerations and Limitations with User-Defined Formats

- If you create a local user-defined format with the same name but a different value than a user-defined format that was published previously to the database a `check sum ERROR` warning occurs and the local format is used. This warning indicates that the local and published formats differ. The query is processed by SAS instead of inside the database.

  If you want the query to be processed inside the database, you need to redefine the local format to match the published version and rerun the query.
- Trailing blanks in PROC FORMAT labels are lost when publishing a picture format.
- Avoid using PICTURE formats with the MULTILABEL option. You cannot successfully create a CNTLOUT= data set when PICTURE formats are present. This a known problem in PROC FORMAT.
- If you use the MULTILABEL option, only the first label that is found is returned. For more information, see the PROC FORMAT MULTILABEL option in the *Base SAS Procedures Guide*.
- The format publishing macros reject a format unless the LANGUAGE= option is set to English or is not specified.
- Although the format catalog can contain informats, the format publishing macros ignore the informats.
- User-defined formats that include a format that SAS supplies are not supported.

Tips for Using the Format Publishing Macros

- Use the ACTION=CREATE option only the first time that you run the format publishing macro. After that, use ACTION=REPLACE or ACTION=DROP.
- The format publishing macro does not require a format catalog. If you do not have any custom formats, only the formats that SAS supplies are published. However, you can use this code to create an empty format catalog in your WORK directory before you publish the PUT function and the formats that SAS supplies:

```
proc format;
run;
```

- If you modify any PROC FORMAT entries in the source catalog, you must republish the entire catalog.
- If the format publishing macro is executed between two procedure calls, the page number of the last query output is increased by two.

Tips for Using the SAS_PUT() Function

- When SAS parses the PUT function, SAS checks to make sure that the format is a known format name. SAS looks for the format in the set of formats that are defined in the scope of the current SAS session. If the format name is not defined in the context of the current SAS session, the SAS_PUT() is returned to the local SAS session for processing.
- Using both the SQLREDUCEPUT= system option (or the PROC SQL REDUCEPUT= option) and SQLMAPPUTTO= can result in a significant performance boost. First, SQLREDUCEPUT= works to reduce as many PUT functions as possible. Then, using SQLMAPPUTTO= with the format publishing macro changes the remaining PUT functions to SAS_PUT() functions.

  For more information, see the "SQLMAPPUTTO= System Option" on page 140 and the "SQLREDUCEPUT= System Option" on page 141.
- To turn off automatic translation of the PUT function to the SAS_PUT() function, set the SQLMAPPUTTO= system option to NONE.
- The format of the SAS_PUT() function parallels that of the PUT function:

```
SAS_PUT(source, 'format.')
```

Determining Format Publish Dates

You might need to know when user-defined formats or formats that SAS supplies were published. SAS supplies two special formats that return a datetime value that indicates when this occurred.

- The INTRINSIC-CRDATE format returns a datetime value that indicates when the SAS formats library was published.

- The UFMT-CRDATE format returns a datetime value that indicates when the user-defined formats were published.

Note: You must use the SQL pass-through facility to return the datetime value associated with the INTRINSIC-CRDATE and UFMT-CRDATE formats, as illustrated in this example:

```
proc sql noerrorstop;
   connect to
&tera (
&connopt);

title 'Publish date of SAS Format Library';
select * from connection to
&tera
   (
      select sas_put(1, 'intrinsic-crdate.')
         as sas_fmts_datetime;
   );
title 'Publish date of user-defined formats';
select * from connection to
&tera
   (
      select sas_put(1, 'ufmt-crdate.')
         as my_formats_datetime;
   );

disconnect from teradata;
quit;
```

Chapter 11

Deploying and Using SAS Formats in DB2 under UNIX

User-Defined Formats in the DB2 Database **91**

Publishing SAS Formats in DB2 . . . . . **91**
Overview of the Publishing Process 91
Running the %INDB2_PUBLISH_FORMATS Macro 92
%INDB2PF Macro 93
INDCONN Macro Variable 93
%INDB2_PUBLISH_FORMATS Macro Syntax 94
Modes of Operation 96
Format Publishing Macro Example 97

Using the SAS_PUT() Function in the DB2 Database **97**
Implicit Use of the SAS_PUT() Function 97
Explicit Use of the SAS_PUT() Function 99

DB2 Permissions **100**

User-Defined Formats in the DB2 Database

You can use PROC FORMAT to create user-defined formats and store them in a format catalog. You can then use the %INDB2_PUBLISH_FORMATS macro to export the user-defined format definitions to the DB2 database where the SAS_PUT() function can reference them.

For more information about the %INDB2_PUBLISH_FORMATS macro, see"Publishing SAS Formats in DB2" on page 91. For more information about the SAS_PUT() function, see "Using the SAS_PUT() Function in the DB2 Database" on page 97.

Publishing SAS Formats in DB2

Overview of the Publishing Process

The SAS publishing macros are used to publish formats and the SAS_PUT() function in DB2.

Note: SFTP is used to transfer the source files to the DB2 server during the publishing process. Certain software products that support SSH-2 or SFTP protocols must be

installed before you can use the publishing macros. For more information, see the *SAS In-Database Products: Administrator's Guide*.

The %INDB2_PUBLISH_FORMATS macro creates the files that are needed to build the SAS_PUT() function and publishes those files to the DB2 database.

This macro also makes many formats that SAS supplies available inside DB2. In addition to formats that SAS supplies, you can also publish the PROC FORMAT definitions that are contained in a single SAS format catalog by using the FMTCAT= option. The process of publishing a PROC FORMAT catalog entry converts the range label pairs into embedded data in DB2.

The %INDB2_PUBLISH_FORMATS macro performs the following tasks:

- produces the set of .c and .h files that are necessary to build the SAS_PUT() function
- produces a script of the DB2 commands that are necessary to register the SAS_PUT() function in the DB2 database
- transfers the .c and .h files to DB2 using SFTP
- calls the SAS_COMPILEUDF function to compile the source files into object files and to link to the SAS Formats Library for DB2
- calls the SAS_DELETEUDF function to remove existing object files and then replaces them with the new object files
- uses the SAS/ACCESS Interface to DB2 to run the script and publish the SAS_PUT() function to the DB2 database

The SAS_PUT() function is registered in DB2 with shared object files that are loaded at run time. These functions must be stored in a permanent location. The SAS object files and the SAS Formats Library for DB2 are stored in the ***db2path*/SQLLIB/FUNCTION/SAS** directory where you supply the ***db2path***. This directory is accessible to all database partitions.

DB2 caches the object files after they are loaded. Each time the updated objects are used, you must either stop and restart the database to clean up the cache, or rename the object files and register the functions with the new object filenames. The SAS publishing process automatically handles the renaming to avoid stopping and restarting the database.

Running the %INDB2_PUBLISH_FORMATS Macro

To run the %INDB2_PUBLISH_FORMATS macro, follow these steps:

1. Start SAS 9.3, and submit these commands in the Program Editor or the Enhanced Editor:

   ```
   %indb2pf;
   %let indconn = server=yourserver user=youruserid password=yourpwd
      database=yourdb schema=yourschema serveruserid=yourserveruserid;
   ```

   For more information, see "%INDB2PF Macro" on page 93 and "INDCONN Macro Variable" on page 93.

2. Run the %INDB2_PUBLISH_FORMATS macro.

   For more information, see "%INDB2_PUBLISH_FORMATS Macro Syntax" on page 94.

   Messages are written to the SAS log that indicate whether the SAS_PUT() function was successfully created.

%INDB2PF Macro

The %INDB2PF macro is an autocall library that initializes the format publishing software.

INDCONN Macro Variable

The INDCONN macro variable is used as credentials to connect to DB2. You must specify the server, user, password, and database. The schema name and server user ID are optional. You must assign the INDCONN macro variable before the %INDB2_PUBLISH_FORMATS macro is invoked.

Here is the syntax for the value of the INDCONN macro variable:

SERVER=*server* USER=*userid* PASSWORD=*password*
DATABASE=*database* <SCHEMA=*schemaname*> <SERVERUSERID=*serveruserid*>

Arguments

SERVER=*server*
specifies the DB2 server name or the IP address of the server host. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Requirement: The name must be consistent with the way the host name was cached when PSFTP *server* was run from the command window. If the full server name was cached, you must use the full server name in the SERVER argument. If the short server name was cached, you must use the short server name. For example, if the long name, *disk3295.unx.comp.com*, is used when PSFTP was run, then *server=disk3295.unx.comp.com* must be specified. If the short name, *disk3295*, was used, then *server=disk3295* must be specified. For more information about running the PSFTP command, see *DB2 Installation and Configuration Steps* in the *SAS In-Database Products: Administrator's Guide*.

USER=*userid*
specifies the DB2 user name (also called the user ID) that is used to connect to the database.

PASSWORD=*password*
specifies the password that is associated with your DB2 user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error.

DATABASE=*database*
specifies the DB2 database that contains the tables and views that you want to access.

Requirement: The format functions are created as Unicode functions. If the database is not a Unicode database, then the alternate collating sequence must be configured to use **`identity_16bit`**.

SCHEMA=*schema*
specifies the schema name for the database.

Default: If you do not specify a value for the SCHEMA argument, the value of the USER argument is used as the schema name.

SERVERUSERID=*serveruserid*
specifies the user ID for SAS SFTP and enables you to access the machine on which you have installed the DB2 database.

Default: If you do not specify a value for the SERVERUSERID argument, the value of the USER argument is used as the user ID for SAS SFTP.

Note: The person who installed and configured the SSH software can provide you with the SERVERUSERID (SFTP user ID) and the private key that need to be added to the pageant.exe (Windows) or SSH agent (UNIX). Pageant must be running for the SFTP process to be successful.

TIP The INDCONN macro variable is not passed as an argument to the %INDB2_PUBLISH_FORMATS macro. This information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDB2_PUBLISH_FORMATS Macro Syntax

%INDB2_PUBLISH_FORMATS (
<DATABASE=*database–name*>
<, FMTCAT=*format-catalog-filename*>
<, FMTTABLE=*format-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
<, MODE=PROTECTED | UNPROTECTED>
<, INITIAL_WAIT=*wait-time*>
<, FTPTIMEOUT=*timeout-time*>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DATABASE=*database-name*
specifies the name of a DB2 database to which the SAS_PUT() function and the formats are published. This argument lets you publish the SAS_PUT() function and the formats to a shared database where other users can access them.

Requirement: The format functions are created as Unicode functions. If the database is not a Unicode database, then the alternate collating sequence must be configured to use `identity_16bit`.

Interaction: The database that is specified by the DATABASE= argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "Running the %INDB2_PUBLISH_FORMATS Macro" on page 92.

Tip: It is not necessary that the format definitions and the SAS_PUT() function reside in the same database as the one that contains the data that you want to format. You can use the SQLMAPPUTO= system option to specify the database where the format definitions and the SAS_PUT() function have been published.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created with the FORMAT procedure and will be made available in DB2.

Default: If you do not specify a value for FMTCAT= and you have created user-defined formats in your SAS session, the default is WORK.FORMATS. If you do not specify a value for FMTCAT= and you have not created any user-defined formats in your SAS session, only the formats that SAS supplies are available in DB2.

Interaction: If the format definitions that you want to publish exist in multiple catalogs, you must copy them into a single catalog for publishing. If you specify more than one format catalog using the FMTCAT argument, only the last catalog that you specify is published.

See: "Considerations and Limitations with User-Defined Formats" on page 87

FMTTABLE=*format-table-name*
specifies the name of the DB2 table that contains all formats that the %INDB2_PUBLISH_FORMATS macro creates and that the SAS_PUT() function supports. The table contains the columns shown in Table 2.1.

Table 11.1 *Format Table Columns*

Column Name	Description
FMTNAME	specifies the name of the format.
SOURCE	specifies the origin of the format. SOURCE can contain one of these values: SAS supplied by SAS PROCFMT User–defined with PROC FORMAT

Default: If FMTTABLE is not specified, no table is created. You can see only the SAS_PUT() function. You cannot see the formats that are published by the macro.

Interaction: If ACTION=CREATE or ACTION=DROP is specified, messages are written to the SAS log that indicate the success or failure of the table creation or drop.

ACTION=CREATE | REPLACE | DROP
specifies that the macro performs one of these actions:

CREATE
creates a new SAS_PUT() function.

REPLACE
overwrites the current SAS_PUT() function, if a SAS_PUT() function is already registered or creates a new SAS_PUT() function if one is not registered.

DROP
causes the SAS_PUT() function to be dropped from the DB2 database.

Interaction: If FMTTABLE= is specified, both the SAS_PUT() function and the format table are dropped. If the table name cannot be found or is incorrect, only the SAS_PUT() function is dropped.

Default: CREATE

Tip: If the SAS_PUT() function was defined previously and you specify ACTION=CREATE, you receive warning messages from DB2. If the SAS_PUT() function was defined previously and you specify ACTION=REPLACE, a message is written to the SAS log indicating that the SAS_PUT() function has been replaced.

MODE=FENCED | UNFENCED
specifies whether the running code is isolated in a separate process in the DB2 database so that a program fault does not cause the database to stop.

Default: FENCED

Tip: Once the SAS formats are validated in fenced mode, you can republish them in unfenced mode for a significant performance gain.

INITIAL_WAIT=*wait-time*
specifies the initial wait time in seconds for SAS SFTP to parse the responses and complete the SFTP batchfile process.

Default: 15 seconds

Interactions:
The INITIAL_WAIT= argument works in conjunction with the FTPTIMEOUT= argument. Initially, SAS SFTP waits the amount of time specified by the INITIAL_WAIT= argument. If the SFTP batchfile process is not complete after the initial wait time, retries occur until the wait time is equal to or greater than the time-out value specified by the FTPTIMEOUT= argument. All retries double the previous wait time. SAS SFTP fails after the time-out value is reached or exceeded. An error message is written to the SAS log.

For example, assume that you use the default values. The initial wait time is 15 seconds. The first retry waits 30 seconds. The second retry waits 60 seconds. The third retry waits 120 seconds, which is the default time-out value. So the default initial wait time and time-out values enable four possible tries: the initial try, and three retries.

See: FTPTIMEOUT= argument

FTPTIMEOUT=*time-out-value*
specifies the time-out value in seconds if SAS SFTP fails to transfer the files.

Default: 120 seconds

Interactions:
The FTPTIMEOUT= argument works in conjunction with the INITIAL_WAIT= argument. Initially, SAS SFTP waits the amount of time specified by the INITIAL_WAIT= argument. If the SFTP batchfile process is not complete after the initial wait time, retries occur until the wait time is equal to or greater than the time-out value specified by the FTPTIMEOUT= argument. All retries double the previous wait time. SAS SFTP fails after the time-out value is reached or exceeded and an error message is written to the SAS log.

For example, assume you use the default values. The initial wait time is 15 seconds. The first retry waits 30 seconds. The second retry waits 60 seconds. The third retry waits 120 seconds, which is the default time-out value. So the default initial wait time and time-out values enable four possible tries: the initial try, and three retries.

Tip: Use this argument to control how long SAS SFTP waits to complete a file transfer before timing out. A time-out failure could indicate a network or key authentication problem.

See: INITIAL_WAIT argument

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process.

See: "Special Characters in Directory Names" on page 86

Modes of Operation

There are two modes of operation when executing the %INDB2_PUBLISH_FORMATS macro: fenced and unfenced. You specify the mode by setting the MODE= argument.

The default mode of operation is fenced. Fenced mode means that the macro code is isolated in a separate process in the DB2 database, and an error does not cause the

database to stop. It is recommended that you run the %INDB2_PUBLISH_FORMATS macro in fenced mode during acceptance tests.

When the %INDB2_PUBLISH_FORMATS macro is ready for production, you can rerun the macro in unfenced mode. Note that you should see a significant performance advantage when you republish the formats in unfenced mode

Format Publishing Macro Example

```
%indb2pf;
%let indconn = server=db2base user=user1 password=open1
database=mydb
   schema=myschema;
%indb2_publish_formats(fmtcat= fmtlib.fmtcat);
```

This sequence of macros generates .c and .h files for each data type. The format data types that are supported are numeric (FLOAT, INT), character, date, time, and timestamp (DATETIME). The %INDB2_PUBLISH_FORMATS macro also produces a text file of DB2 CREATE FUNCTION commands that are similar to these:

```
CREATE FUNCTION sas_put(float , varchar(256))
RETURNS VARCHAR(256)
LANGUAGE C
PARAMETER STYLE npsgeneric
CALLED ON NULL INPUT
EXTERNAL CLASS NAME 'Csas_putn'
EXTERNAL HOST OBJECT '/tmp/tempdir_20090528T135753_616784/formal5.o_x86'
EXTERNAL NSPU OBJECT '/tmp/tempdir_20090528T135753_616784/formal5.o_diab_ppc'
```

After it is installed, you can call the SAS_PUT() function in DB2 by using SQL. For more information, see “Using the SAS_PUT() Function in the DB2 Database” on page 97.

Using the SAS_PUT() Function in the DB2 Database

Implicit Use of the SAS_PUT() Function

After you install the formats that SAS supplies in libraries inside the DB2 database and publish any custom format definitions that you created in SAS, you can access the SAS_PUT() function with your SQL queries.

If the SQLMAPPUTTO= system option is set to SAS_PUT and you submit your program from a SAS session, the SAS SQL processor maps PUT function calls to SAS_PUT() function references that DB2 understands.

This example illustrates how the PUT function is mapped to the SAS_PUT() function using implicit pass-through.

```
options sqlmapputto=sas_put;

%put &mapconn;
user=dbitest password=dbigrp1 server=spubox database=TESTDB
   sql_functions="EXTERNAL_APPEND=WORK.dbfuncext" sql_functions_copy=saslog
```

```
libname dblib DB2 &mapconn;

   /*-- Set SQL debug global options --*/
   /*----------------------------------*/
   options sastrace=',,,d' sastraceloc=saslog;

   /*-- Execute SQL using Implicit Passthru --*/
   /*----------------------------------------*/
   proc sql noerrorstop;
      title1 'Test SAS_PUT using Implicit Passthru ';
      select distinct
         PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo;

   quit;
```

These lines are written to the SAS log.

```
options sqlmapputto=sas_put;

%put &mapconn;
user=dbitext password=dbigrp1 server=spubox database=TESTDB
   sql_functions="EXTERNAL_APPEND=WORK.dbfuncext" sql_functions_copy=saslog;

libname dblib DB2 &mapconn;

NOTE: Libref DBLIB was successfully assigned, as follows:
      Engine:        DB2
      Physical Name: spubox

         /*-- Set SQL debug global options --*/
         /*----------------------------------*/
        options sastrace=',,,d' sastraceloc=saslog;

         /*-- Execute SQL using Implicit Passthru --*/
         /*----------------------------------------*/
         proc sql noerrorstop;
           title1 'Test SAS_PUT using Implicit Passthru ';
           select distinct
              PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo
                ;
DB2: AUTOCOMMIT is NO for connection 1
DB2: AUTOCOMMIT turned ON for connection id 1

DB2_1: Prepared: on connection 1
SELECT * FROM mailorderdemo

DB2: AUTOCOMMIT is NO for connection 2
DB2: AUTOCOMMIT turned ON for connection id 2

DB2_2: Prepared: on connection 2
 select distinct cast(sas_put(mailorderdemo."PRICE", 'DOLLAR8.2') as char(8))
        as PRICE_C from mailorderdemo

DB2_3: Executed: on connection 2
Prepared statement DB2_2
```

```
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.

                Test SAS_PUT using Implicit Passthru                    9
                                                  13:42 Thursday, May 7, 2011

                                PRICE_C
                                _______
                                 $10.00
                                 $12.00
                                 $13.59
                                 $48.99
                                 $54.00
                                  $8.00
                                 $14.00
                                 $27.98
                                 $13.99

          quit;
```

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT to ensure that the SQL processor maps your PUT functions to the SAS_PUT() function and the SAS_PUT() reference is passed through to DB2.
- The SAS SQL processor translates the PUT function in the SQL SELECT statement into a reference to the SAS_PUT() function.

  ```
  select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
     as char(8)) as "PRICE_C" from "sas"."mailorderdemo"
  ```

  A large value, VARCHAR(*n*), is always returned because one function prototype accesses all formats. Use the CAST expression to reduce the width of the returned column to be a character width that is reasonable for the format that is being used.

  The return text cannot contain a binary zero value (hexadecimal 00) because the SAS_PUT() function always returns a VARCHAR(*n*) data type and a DB2 VARCHAR(*n*) is defined to be a null-terminated string.

The SELECT DISTINCT clause executes inside DB2, and the processing is distributed across all available data nodes. DB2 formats the price values with the $DOLLAR8.2 format and processes the SELECT DISTINCT clause using the formatted values.

Explicit Use of the SAS_PUT() Function

If you use explicit pass-through (direct connection to DB2), you can use the SAS_PUT() function call in your SQL program.

This example shows the same query from "Implicit Use of the SAS_PUT() Function" on page 97 and explicitly uses the SAS_PUT() function call.

```
options sqlmapputto=sas_put sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
   title1 'Test SAS_PUT using Explicit Passthru';
   connect to DB2 (user=dbitest password=XXXXXXX database=testdb
      server=spubox);

   select * from connection to DB2
```

```
      (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
         "PRICE_C" from mailorderdemo);

disconnect from DB2;
quit;
```

The following lines are written to the SAS log.

```
options sqlmapputto=sas_put sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
 title1 'Test SAS_PUT using Explicit Passthru';
 connect to DB2 (user=dbitest password=XXXXXXX database=testdb server=spubox);

 select * from connection to DB2
    (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
        "PRICE_C" from mailorderdemo);

           Test SAS_PUT using Explicit Passthru            2
                                      17:13 Thursday, May 7, 2011

                        PRICE_C
                        _______
                         $27.98
                         $10.00
                         $12.00
                         $13.59
                         $48.99
                         $54.00
                         $13.98
                          $8.00
                         $14.00

 disconnect from DB2;
 quit;
```

Note: If you explicitly use the SAS_PUT() function in your code, it is recommended that you use double quotation marks around a column name to avoid any ambiguity with the keywords. For example, if you did not use double quotation marks around the column name, DATE, in this example, all date values would be returned as today's date.

```
select distinct
   cast(sas_put("price", 'dollar8.2') as char(8)) as "price_c",
   cast(sas_put("date", 'date9.1') as char(9)) as "date_d",
   cast(sas_put("inv", 'best8.') as char(8))  as "inv_n",
   cast(sas_put("name", '$32.') as char(32)) as "name_n"
from mailorderdemo;
```

DB2 Permissions

You must have DB2 user permissions to execute the SAS publishing macros to publish the SAS_PUT() and format functions. Some of these permissions are as follows.

- EXECUTE user permission for functions that were published by another user

- READ user permission to read the SASUDF_COMPILER_PATH and SASUDF_DB2PATH global variables
- CREATE_EXTERNAL_ROUTINE user permission to the database to create functions
- CREATEIN user permission for the schema in which the SAS_PUT() and format functions are published if a nondefault schema is used
- CREATE_NOT_FENCED_ROUTINE user permission to create functions that are not fenced

Permissions must be granted for each user that needs to publish the SAS_PUT() and format functions and for each database that the format publishing uses. Without these permissions, publishing of the SAS_PUT() and format functions fail.

The person who can grant the permissions and the order in which permissions are granted is important. For complete information and examples, see the installation and configuration instructions in the *SAS In-Database Products: Administrator's Guide*.

Chapter 12

Deploying and Using SAS Formats in Netezza

User-Defined Formats in the Netezza Data Warehouse **103**
Introduction to User-Defined Formats in Netezza 103
Netezza Considerations and Limitations When Using the FMTCAT= Options . . 104

Publishing SAS Formats in Netezza **104**
Overview of the Publishing Process 104
Running the %INDNZ_PUBLISH_FORMATS Macro 104
%INDNZPF Macro 105
INDCONN Macro Variable 105
%INDNZ_PUBLISH_FORMATS Macro Syntax 106
Modes of Operation 108
Format Publishing Macro Example 108

Using the SAS_PUT() Function in the Netezza Data Warehouse **109**
Implicit Use of the SAS_PUT() Function 109
Explicit Use of the SAS_PUT() Function 111

Netezza Permissions **112**

User-Defined Formats in the Netezza Data Warehouse

Introduction to User-Defined Formats in Netezza

You can use PROC FORMAT to create user-defined formats and store them in a format catalog. You can then use the %INDNZ_PUBLISH_FORMATS macro to export the user-defined format definitions to the Netezza data warehouse where the SAS_PUT() function can reference them.

For more information about the %INDNZ_PUBLISH_FORMATS macro, see "Publishing SAS Formats in Netezza" on page 104. For more information about the SAS_PUT() function, see "Using the SAS_PUT() Function in the Netezza Data Warehouse " on page 109.

Netezza Considerations and Limitations When Using the FMTCAT= Options

If you use the FMTCAT= option to specify a format catalog in the %INDNZ_PUBLISH_FORMATS macro, the following limitations apply if you are using a character set encoding other than Latin1:

- Picture formats are not supported. The picture format supports only Latin1 characters.
- If the format value's encoded string is longer than 256 bytes, the string is truncated and a warning is printed to the SAS log.

Publishing SAS Formats in Netezza

Overview of the Publishing Process

The SAS publishing macros are used to publish formats and the SAS_PUT() function in Netezza.

The %INDNZ_PUBLISH_FORMATS macro creates the files that are needed to build the SAS_PUT() function and publishes those files to the Netezza data warehouse.

This macro also makes many formats that SAS supplies available inside Netezza. In addition to formats that SAS supplies, you can also publish the PROC FORMAT definitions that are contained in a single SAS format catalog by using the FMTCAT= option. The process of publishing a PROC FORMAT catalog entry converts the range label pairs into embedded data in Netezza.

The %INDNZ_PUBLISH_FORMATS macro performs the following tasks:

- produces the set of .c, .cpp, and .h files that are necessary to build the SAS_PUT() function
- produces a script of the Netezza commands that are necessary to register the SAS_PUT() function on the Netezza data warehouse
- transfers the .c, .cpp, and .h files to Netezza using the Netezza External Table interface
- calls the SAS_COMPILEUDF function to compile the source files into object files and to access the SAS Formats Library for Netezza
- uses SAS/ACCESS Interface to Netezza to run the script to create the SAS_PUT() function with the object files

Running the %INDNZ_PUBLISH_FORMATS Macro

To run the %INDNZ_PUBLISH_FORMATS macro, complete the following steps:

1. Start SAS 9.3 and submit these commands in the Program or Enhanced Editor:

```
%indnzpf;
%let indconn = server=myserver user=myuserid password=XXXX
   database=mydb <serveruserid=myserveruserid>;
```

For more information, see " %INDNZPF Macro" on page 105 and the "INDCONN Macro Variable" on page 105.

2. Run the %INDNZ_PUBLISH_FORMATS macro.

   For more information, see "%INDNZ_PUBLISH_FORMATS Macro Syntax" on page 106.

   Messages are written to the SAS log that indicate whether the SAS_PUT() function was successfully created.

%INDNZPF Macro

The %INDNZPF macro is an autocall library that initializes the format publishing software.

INDCONN Macro Variable

The INDCONN macro variable is used as credentials to connect to Netezza. You must specify the server, user, password, database, and (as an option) the server user ID information to access the machine on which you have installed the Netezza data warehouse. You must assign the INDCONN macro variable before the %INDNZ_PUBLISH_FORMATS macro is invoked.

Here is the syntax for the value of the INDCONN macro variable:

SERVER=*server* USER=*userid* PASSWORD=*password* DATABASE=*database*

Arguments

SERVER=*server*
specifies the Netezza server name or the IP address of the server host.

USER=*user*
specifies the Netezza user name (also called the user ID) that is used to connect to the database.

PASSWORD=*password*
specifies the password that is associated with your Netezza user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error to occur.

DATABASE=*database*
specifies the Netezza database that contains the tables and views that you want to access.

TIP The INDCONN macro variable is not passed as an argument to the %INDNZ_PUBLISH_FORMATS macro. This information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDNZ_PUBLISH_FORMATS Macro Syntax

%INDNZ_PUBLISH_FORMATS (
<, DATABASE=*database-name*>
<. DBCOMPILE=*database-name*>
<, DBJAZLIB=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, FMTTABLE=*format-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
<, MODE=FENCED | UNFENCED>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DATABASE=*database-name*
specifies the name of a Netezza database to which the SAS_PUT() function and the formats are published. This argument lets you publish the SAS_PUT() function and the formats to a shared database where other users can access them.

Interaction: The database that is specified by the DATABASE= argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "Running the %INDNZ_PUBLISH_FORMATS Macro" on page 104.

Tip: It is not necessary that the format definitions and the SAS_PUT() function reside in the same database as the one that contains the data that you want to format. You can use the SQLMAPPUTO= system option to specify the database where the format definitions and the SAS_PUT() function have been published.

DBCOMPILE=*database-name*
specifies the name of the database where the SAS_COMPILEUDF function was published.

Default: SASLIB

See: For more information about the publishing the SAS_COMPILEUDF function, see the *SAS In-Database Products: Administrator's Guide*.

DBJAZLIB=*database-name*
specifies the name of the database where the SAS Formats Library for Netezza was published.

Default: SASLIB

Restriction: This argument is supported only on TwinFin systems.

See: For more information about publishing the SAS Formats Library for Netezza, see the *SAS In-Database Products: Administrator's Guide*.

FMTCAT=*format-catalog-filename*
specifies the name of the format catalog file that contains all user-defined formats that were created with the FORMAT procedure and are made available in Netezza.

Default: If you do not specify a value for FMTCAT= and you have created user-defined formats in your SAS session, the default is WORK.FORMATS. If you do not specify a value for FMTCAT= and you have not created any user-defined formats in your SAS session, only the formats that SAS supplies are available in Netezza.

Interaction: If the format definitions that you want to publish exist in multiple catalogs, you must copy them into a single catalog for publishing.

See: "Netezza Considerations and Limitations When Using the FMTCAT= Options" on page 104

FMTTABLE=*format-table-name*
specifies the name of the Netezza table that contains all formats that the %INDNZ_PUBLISH_FORMATS macro creates and that the SAS_PUT() function supports. The table contains the columns shown in Table 2.1.

Table 12.1 *Format Table Columns*

Column Name	Description
FMTNAME	specifies the name of the format.
SOURCE	specifies the origin of the format. SOURCE can contain one of these values: SAS supplied by SAS PROCFMT User-defined with PROC FORMAT

Default: If FMTTABLE is not specified, no table is created. You can see only the SAS_PUT() function. You cannot see the formats that are published by the macro.

Interaction: If ACTION=CREATE or ACTION=DROP is specified, messages are written to the SAS log that indicate the success or failure of the table creation or drop.

ACTION=CREATE | REPLACE | DROP
specifies that the macro performs one of these actions:

CREATE
creates a new SAS_PUT() function.

REPLACE
overwrites the current SAS_PUT() function, if a SAS_PUT() function is already registered or creates a new SAS_PUT() function if one is not registered.

DROP
causes the SAS_PUT() function to be dropped from the Netezza database.

Interaction: If FMTTABLE= is specified, both the SAS_PUT() function and the format table are dropped. If the table name cannot be found or is incorrect, only the SAS_PUT() function is dropped.

Default: CREATE

Tip: If the SAS_PUT() function was published previously and you specify ACTION=CREATE, you receive warning messages that the function already exists and you are prompted to use REPLACE. If you specify ACTION=DROP and the function does not exist, an error message is issued.

MODE= FENCED | UNFENCED
specifies whether running the code is isolated in a separate process in the Netezza database so that a program fault does not cause the database to stop.

Default: FENCED

Restriction: The MODE= argument is supported for Netezza 6.0. The MODE argument is ignored for previous versions of Netezza.

See: "Modes of Operation" on page 108

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process.

See: "Special Characters in Directory Names" on page 86

Modes of Operation

The %INDNZ_PUBLISH_FORMATS macro has two modes of operation: fenced and unfenced. You specify the mode by setting the MODE= argument.

The default mode of operation is fenced. Fenced mode means that the scoring function that is published is isolated in a separate process in the Netezza database when it is invoked, and an error does not cause the database to stop. It is recommended that you publish the scoring functions in fenced mode during acceptance tests.

When the scoring function is ready for production, you can run the macro to publish the scoring function in unfenced mode. You could see a performance advantage if the scoring function is published in unfenced mode.

Note: The MODE= argument is supported for Netezza 6.0. The MODE argument is ignored for previous versions of Netezza.

Format Publishing Macro Example

```
%indnzpf;
%let indconn = server=netezbase user=user1 password=open1
database=mydb;
%indnz_publish_formats(fmtcat= fmtlib.fmtcat);
```

This sequence of macros generates .c, .cpp, and .h files for each data type. The format data types that are supported are numeric (FLOAT, INT), character, date, time, and timestamp (DATETIME). The %INDNZ_PUBLISH_FORMATS macro also produces a text file of Netezza CREATE FUNCTION commands that are similar to these:

```
CREATE FUNCTION sas_put(float , varchar(256))
RETURNS VARCHAR(256)
LANGUAGE CPP
PARAMETER STYLE npsgeneric
CALLED ON NULL INPUT
EXTERNAL CLASS NAME 'Csas_putn'
EXTERNAL HOST OBJECT '/tmp/tempdir_20090528T135753_616784/formal5.o_x86'
EXTERNAL NSPU OBJECT '/tmp/tempdir_20090528T135753_616784/formal5.o_diab_ppc'
```

After it is installed, you can call the SAS_PUT() function in Netezza by using SQL. For more information, see "Using the SAS_PUT() Function in the Netezza Data Warehouse " on page 109.

Using the SAS_PUT() Function in the Netezza Data Warehouse

Implicit Use of the SAS_PUT() Function

After you install the formats that SAS supplies in libraries inside the Netezza data warehouse and publish any custom format definitions that you created in SAS, you can access the SAS_PUT() function with your SQL queries.

If the SQLMAPPUTTO= system option is set to SAS_PUT and you submit your program from a SAS session, the SAS SQL processor maps PUT function calls to SAS_PUT() function references that Netezza understands.

This example illustrates how the PUT function is mapped to the SAS_PUT() function using implicit pass-through.

```
options sqlmapputto=sas_put;

%put &mapconn;

libname dblib netezza &mapconn;

   /*-- Set SQL debug global options --*/
   /*----------------------------------*/
   options sastrace=',,,d' sastraceloc=saslog;

   /*-- Execute SQL using Implicit Passthru --*/
   /*----------------------------------------*/
   proc sql noerrorstop;
      title1 'Test SAS_PUT using Implicit Passthru ';
      select distinct
         PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo;

   quit;
```

These lines are written to the SAS log.

```
options sqlmapputto=sas_put;

%put &mapconn;
user=dbitext password=dbigrp1 server=spubox database=TESTDB
   sql_functions="EXTERNAL_APPEND=WORK.dbfuncext" sql_functions_copy=saslog;

libname dblib netezza &mapconn;

NOTE: Libref DBLIB was successfully assigned, as follows:
      Engine:        NETEZZA
      Physical Name: spubox

         /*-- Set SQL debug global options --*/
         /*----------------------------------*/
        options sastrace=',,,d' sastraceloc=saslog;
```

```
         /*-- Execute SQL using Implicit Passthru --*/
         /*----------------------------------------*/
         proc sql noerrorstop;
           title1 'Test SAS_PUT using Implicit Passthru ';
           select distinct
              PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo
                ;
NETEZZA: AUTOCOMMIT is NO for connection 1
NETEZZA: AUTOCOMMIT turned ON for connection id 1

NETEZZA_1: Prepared: on connection 1
SELECT * FROM mailorderdemo

NETEZZA: AUTOCOMMIT is NO for connection 2
NETEZZA: AUTOCOMMIT turned ON for connection id 2

NETEZZA_2: Prepared: on connection 2
 select distinct cast(sas_put(mailorderdemo."PRICE", 'DOLLAR8.2') as char(8))
        as PRICE_C from mailorderdemo

NETEZZA_3: Executed: on connection 2
Prepared statement NETEZZA_2

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.

              Test SAS_PUT using Implicit Passthru                        9
                                                 13:42 Thursday, May 7, 2011

                                PRICE_C
                                _______
                                 $10.00
                                 $12.00
                                 $13.59
                                 $48.99
                                 $54.00
                                  $8.00
                                 $14.00
                                 $27.98
                                 $13.99

           quit;
```

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT to ensure that the SQL processor maps your PUT functions to the SAS_PUT() function and the SAS_PUT() reference is passed through to Netezza.
- The SAS SQL processor translates the PUT function in the SQL SELECT statement into a reference to the SAS_PUT() function.

  ```
  select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
     as char(8)) as "PRICE_C" from "sas"."mailorderdemo"
  ```

  A large value, VARCHAR(*n*), is always returned because one function prototype accesses all formats. Use the CAST expression to reduce the width of the returned column to be a character width that is reasonable for the format that is being used.

The return text cannot contain a binary zero value (hexadecimal 00) because the SAS_PUT() function always returns a VARCHAR(*n*) data type and a Netezza VARCHAR(*n*) is defined to be a null-terminated string.

The SELECT DISTINCT clause executes inside Netezza, and the processing is distributed across all available data nodes. Netezza formats the price values with the $DOLLAR8.2 format and processes the SELECT DISTINCT clause using the formatted values.

Explicit Use of the SAS_PUT() Function

If you use explicit pass-through (direct connection to Netezza), you can use the SAS_PUT() function call in your SQL program.

This example shows the same query from "Implicit Use of the SAS_PUT() Function" on page 109 and explicitly uses the SAS_PUT() function call.

```
options sqlmapputto=sas_put sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
   title1 'Test SAS_PUT using Explicit Passthru';
   connect to netezza (user=dbitest password=XXXXXXX database=testdb
      server=spubox);

   select * from connection to netezza
      (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
         "PRICE_C" from mailorderdemo);

disconnect from netezza;
quit;
```

The following lines are written to the SAS log.

```
options sqlmapputto=sas_put sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
 title1 'Test SAS_PUT using Explicit Passthru';
 connect to netezza (user=dbitest password=XXXXXXX database=testdb server=spubox);

 select * from connection to netezza
    (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
        "PRICE_C" from mailorderdemo);

           Test SAS_PUT using Explicit Passthru            2
                                        17:13 Thursday, May 7, 2011

                          PRICE_C
                          _______
                           $27.98
                           $10.00
                           $12.00
                           $13.59
                           $48.99
                           $54.00
                           $13.98
                            $8.00
                           $14.00
```

```
disconnect from netezza;
quit;
```

Note: If you explicitly use the SAS_PUT() function in your code, it is recommended that you use double quotation marks around a column name to avoid any ambiguity with the keywords. For example, if you did not use double quotation marks around the column name, DATE, in this example, all date values would be returned as today's date.

```
select distinct
   cast(sas_put("price", 'dollar8.2') as char(8)) as "price_c",
   cast(sas_put("date", 'date9.1') as char(9)) as "date_d",
   cast(sas_put("inv", 'best8.') as char(8))  as "inv_n",
   cast(sas_put("name", '$32.') as char(32)) as "name_n"
from mailorderdemo;
```

Netezza Permissions

You must have permission to create the SAS_PUT() function and formats, and tables in the Netezza database. You must also have permission to execute the SAS_COMPILEUDF, SAS_DictionaryUDF, and SAS_HexToTextUDF functions in either the SASLIB database or the database specified in lieu of SASLIB where these functions are published.

Without these permissions, the publishing of the SAS_PUT() function and formats fails. To obtain these permissions, contact your database administrator.

For more information about specific permissions, see the *SAS In-Database Products: Administrator's Guide*.

Chapter 13

Deploying and Using SAS Formats in Teradata

User-Defined Formats in the Teradata EDW **113**
Introduction to User-Defined Formats in Teradata 113
Teradata Limitations and Restrictions When Using the FMTCAT= Option 113

Publishing SAS Formats in Teradata **114**
Overview of the Publishing Process 114
Running the %INDTD_PUBLISH_FORMATS Macro 114
%INDTDPF Macro 115
INDCONN Macro Variable 115
%INDTD_PUBLISH_FORMATS Macro Syntax 115
Modes of Operation 117
Format Publishing Macro Example 118

Data Types and the SAS_PUT() Function **118**

Using the SAS_PUT() Function in the Teradata EDW **120**
Implicit Use of the SAS_PUT() Function 120
Explicit Use of the SAS_PUT() Function 122
Tips When Using the SAS_PUT() Function in Teradata 123

Teradata Permissions **123**

User-Defined Formats in the Teradata EDW

Introduction to User-Defined Formats in Teradata

You can use PROC FORMAT to create user-defined formats and store them in a format catalog. You can then use the %INDTD_PUBLISH_FORMATS macro to export the user-defined format definitions to the Teradata EDW where the SAS_PUT() function can reference them.

For more information about %INDTD_PUBLISH_FORMATS, see "Publishing SAS Formats in Teradata" on page 114. For more information about the SAS_PUT() function, see "Using the SAS_PUT() Function in the Teradata EDW " on page 120.

Teradata Limitations and Restrictions When Using the FMTCAT= Option

If you use the FMTCAT= option to specify a format catalog in the %INDTD_PUBLISH_FORMATS macro and if you are using a character set encoding

other than Latin1, picture formats are not supported. The picture format supports only Latin1 characters.

Publishing SAS Formats in Teradata

Overview of the Publishing Process

The SAS publishing macros are used to publish formats and the SAS_PUT() function in the Teradata EDW.

The %INDTD_PUBLISH_FORMATS macro creates the files that are needed to build the SAS_PUT() function and publishes these files to the Teradata EDW.

The %INDTD_PUBLISH_FORMATS macro also publishes the formats that are included in the SAS formats library. This makes many formats that SAS supplies available inside Teradata. see For more information on the SAS formats library, see "Deployed Components for Teradata" on page 6.

In addition to formats that SAS supplies, you can also publish the PROC FORMAT definitions that are contained in a single SAS format catalog by using the FMTCAT= option. The process of publishing a PROC FORMAT catalog entry converts the range label pairs into embedded data in Teradata. For more information about value-range-sets, see PROC FORMAT in the *Base SAS Procedures User's Guide*.

Note: If you specify more than one format catalog using the FMTCAT= option, the last format that you specify is the one that is published. You can have only one formats library active in the Teradata database.

The %INDTD_PUBLISH_FORMATS macro performs the following tasks:

- creates .h and .c files, which are necessary to build the SAS_PUT() function
- produces a script of Teradata commands that are necessary to register the SAS_PUT() function in the Teradata EDW
- uses SAS/ACCESS Interface to Teradata to execute the script and publish the files to the Teradata EDW

Running the %INDTD_PUBLISH_FORMATS Macro

Follow these steps to run the %INDTD_PUBLISH_FORMATS macro.

1. Start SAS 9.3 and submit these commands in the Program or Enhanced Editor:

```
%indtdpf;
%let indconn = server="myserver" user="myuserid" password="xxxx"
   database="mydb";
```

   For more information, see " %INDTDPF Macro" on page 115 and the "INDCONN Macro Variable" on page 115.

2. Run the %INDTD_PUBLISH_FORMATS macro. For more information, see "%INDTD_PUBLISH_FORMATS Macro Syntax" on page 115.

   Messages are written to the SAS log that indicate whether the SAS_PUT() function was successfully created.

%INDTDPF Macro

The %INDTDPF macro is an autocall library that initializes the format publishing software.

INDCONN Macro Variable

The INDCONN macro variable is used as credentials to connect to Teradata. You must specify the server, user, password, and database information to access the machine on which you have installed the Teradata EDW. You must assign the INDCONN macro variable before the %INDTD_PUBLISH_FORMATS macro is invoked.

Here is the syntax for the value of the INDCONN macro variable:

SERVER="*server*" USER="*userid*" PASSWORD="*password*" DATABASE="*database*"

Arguments

SERVER="*server*"
specifies the Teradata server name or the IP address of the server host.

USER="*user*
specifies the Teradata user name (also called the user ID) that is used to connect to the database.

PASSWORD="*password*
specifies the password that is associated with your Teradata user ID.

Tip: Use only PASSWORD=, PASS=, or PW= for the password argument. PWD= is not supported and causes an error to occur.

DATABASE="*database*"
specifies the Teradata database that contains the tables and views that you want to access.

TIP The INDCONN macro variable is not passed as an argument to the %INDTD_PUBLISH_FORMATS macro. Consequently, this information can be concealed in your SAS job. You might want to place it in an autoexec file and set the permissions on the file so that others cannot access the user ID and password.

%INDTD_PUBLISH_FORMATS Macro Syntax

%INDTD_PUBLISH_FORMATS (
<DATABASE=*database–name*>
<, FMTCAT=*format-catalog-filename*>
<, FMTTABLE=*format-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
<, MODE=PROTECTED | UNPROTECTED>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DATABASE=*database-name*
specifies the name of a Teradata database to which the SAS_PUT() function and the formats are published. This argument lets you publish the SAS_PUT() function and the formats to a shared database where other users can access them.

Interaction: The database that is specified by the DATABASE= argument takes precedence over the database that you specify in the INDCONN macro variable. For more information, see "Running the %INDTD_PUBLISH_FORMATS Macro" on page 114.

Tip: It is not necessary that the format definitions and the SAS_PUT() function reside in the same database as the one that contains the data that you want to format. You can use the SQLMAPPUTTO= system option to specify where the format definitions and the SAS_PUT() function are published. For more information, see "SQLMAPPUTTO= System Option" on page 140.

FMTCAT=*format-catalog-filename*

specifies the name of the format catalog file that contains all user-defined formats that were created with the FORMAT procedure and are made available in Teradata.

Default: If you do not specify a value for FMTCAT= and you have created user-defined formats in your SAS session, the default is WORK.FORMATS. If you do not specify a value for FMTCAT= and you have not created any user-defined formats in your SAS session, only the formats that SAS supplies are available in Teradata.

Interaction: If the format definitions that you want to publish exist in multiple catalogs, you must copy them into a single catalog for publishing.

FMTTABLE=*format–table–name*

specifies the name of the Teradata table that contains all formats that the %INDTD_PUBLISH_FORMATS macro creates and that the SAS_PUT() function supports. The table contains the columns in the following table.

Table 13.1 *Format Table Columns*

Column Name	Description
FMTNAME	specifies the name of the format.
SOURCE	specifies the origin of the format. SOURCE can contain one of these values: SAS supplied by SAS. PROCFMT User-defined with PROC FORMAT.
PROTECTED	specifies whether the format is protected. PROTECTED can contain one of these values: YES Format was created with the MODE= option set to PROTECTED. NO Format was created with the MODE= option set to UNPROTECTED.

Default: If FMTTABLE is not specified, no table is created. You can see only the SAS_PUT() function. You cannot see the formats that are published by the macro.

Interaction: If ACTION=CREATE or ACTION=DROP is specified, messages are written to the SAS log that indicate the success or failure of the table creation or drop.

ACTION=CREATE | REPLACE | DROP
specifies that the macro performs one of these actions:

CREATE
creates a new SAS_PUT() function.

REPLACE
overwrites the current SAS_PUT() function, if a SAS_PUT() function is already registered or creates a new SAS_PUT() function if one is not registered.

DROP
causes the SAS_PUT() function to be dropped from the Teradata database.

Interaction: If FMTTABLE= is specified, both the SAS_PUT() function and the format table are dropped. If the table name cannot be found or is incorrect, only the SAS_PUT() function is dropped.

Default: CREATE.

Tip: If the SAS_PUT() function was defined previously and you specify ACTION=CREATE, you receive warning messages from Teradata. If the SAS_PUT() function was defined previously and you specify ACTION=REPLACE, a message is written to the SAS log indicating that the SAS_PUT() function has been replaced.

MODE=PROTECTED | UNPROTECTED
specifies whether the running code is isolated in a separate process in the Teradata database so that a program fault does not cause the database to stop.

Default: PROTECTED

Tip: Once the SAS formats are validated in PROTECTED mode, you can republish them in UNPROTECTED mode for a performance gain.

See: "Modes of Operation" on page 117

OUTDIR=*diagnostic-output-directory*
specifies a directory that contains diagnostic files.

Files that are produced include an event log that contains detailed information about the success or failure of the publishing process.

See: "Special Characters in Directory Names" on page 86

Modes of Operation

There are two modes of operation when executing the %INDTD_PUBLISH_FORMATS macro: protected and unprotected. You specify the mode by setting the MODE= argument.

The default mode of operation is protected. Protected mode means that the macro code is isolated in a separate process in the Teradata database, and an error does not cause the database to stop. It is recommended that you run the %INDTD_PUBLISH_FORMATS macro in protected mode during acceptance tests.

When the %INDTD_PUBLISH_FORMATS macro is ready for production, you can rerun the macro in unprotected mode. Note that you could see a performance advantage when you republish the formats in unprotected mode.

Format Publishing Macro Example

```
%indtdpf;
%let indconn server="terabase" user="user1" password="open1" database="mydb";
%indtd_publish_formats(fmtcat= fmtlib.fmtcat);
```

This sequence of macros generates a .c and a .h file for each data type. The format data types that are supported are numeric (FLOAT, INT), character, date, time, and timestamp (DATETIME). The %INDTD_PUBLISH_FORMATS macro also produces a text file of Teradata CREATE FUNCTION commands that are similar to these:

```
CREATE FUNCTION sas_put
(d float, f varchar(64))
RETURNS varchar(256)
SPECIFIC sas_putn
LANGUAGE C
NO SQL
PARAMETER STYLE SQL
NOT DETERMINISTIC
CALLED ON NULL INPUT
EXTERNAL NAME
'SL!"jazxfbrs"'
'!CI!ufmt!C:\file-path\'
'!CI!jazz!C:\file-path\'
'!CS!formn!C:\file-path\';
```

After it is installed, you can call the SAS_PUT() function in Teradata by using SQL. For more information, see “Using the SAS_PUT() Function in the Teradata EDW ” on page 120.

Data Types and the SAS_PUT() Function

The SAS_PUT() function supports direct use of the Teradata data types shown in Table 13.2 on page 118. In some cases, the Teradata database performs an implicit conversion of the input data to the match the input data type that is defined for the SAS_PUT() function. For example, all compatible numeric data types are implicitly converted to FLOAT before they are processed by the SAS_PUT() function.

Table 13.2 *Teradata Data Types Supported by the SAS_PUT() Function*

Type of Data	Data Type
Numeric	BYTEINT
	SMALLINT
	INTEGER
	BIGINT*
	DECIMAL (ANSI NUMERIC)*
	FLOAT (ANSI REAL or DOUBLE PRECISION)

Type of Data	Data Type
Date and time	DATE
	TIME
	TIMESTAMP
Character*****	CHARACTER†
	VARCHAR
	LONG VARCHAR

* Numeric precision might be lost when inputs are implicitly converted to FLOAT before they are processed by the SAS_PUT() function.

** Only the Latin-1 character set is supported for character data. UNICODE is not supported at this time.

*** When character inputs are larger than 256 characters, the results depend on the session mode associated with the Teradata connection.

† The SAS_PUT() function has a VARCHAR data type for its first argument when the value passed has a data type of CHARACTER. Therefore, columns with a data type of CHARACTER have their trailing blanks trimmed when converting to a VARCHAR data type.

The SAS_PUT() function does not support direct use of the Teradata data types shown in Table 13.3 on page 119. In some cases, unsupported data types can be explicitly converted to a supported type by using SAS or SQL language constructs. For information about performing explicit data conversions, see the topic on data types for Teradata in *SAS/ACCESS for Relational Databases: Reference* and your Teradata documentation.

Table 13.3 *Teradata Data Types Not Supported by the SAS_PUT() Function*

Type of Data	Data Type
ANSI date and time	INTERVAL
	TIME WITH TIME ZONE
	TIMESTAMP WITH TIME ZONE
GRAPHIC server character set	GRAPHIC
	VARGRAPHIC
	LONG VARGRAPHIC
Binary and large object	CLOB
	BYTE
	VARBYTE
	BLOB

If an incompatible data type is passed to the SAS_PUT() function, various error messages can appear in the SAS log including the following messages:

- Function SAS_PUT does not exist
- Data truncation
- SQL syntax error near the location of the first argument in the SAS_PUT function call

Using the SAS_PUT() Function in the Teradata EDW

Implicit Use of the SAS_PUT() Function

After you install the formats that SAS supplies in libraries inside the Teradata EDW and publish any custom format definitions that you created in SAS, you can access the SAS_PUT() function with your SQL queries.

If the SQLMAPPUTTO= system option is set to SAS_PUT and you submit your program from a SAS session, the SAS SQL processor maps PUT function calls to SAS_PUT() function references that Teradata understands.

This example illustrates how the PUT function is mapped to the SAS_PUT() function using implicit pass-through.

```
options sqlmapputto=sas_put;

libname dblib teradata user="sas" password="sas" server="sl96208"
   database=sas connection=shared;

   /*-- Set SQL debug global options --*/
   /*-----------------------------------*/
   options sastrace=',,,d' sastraceloc=saslog;

   /*-- Execute SQL using Implicit Passthru --*/
   /*-----------------------------------------*/
   proc sql noerrorstop;
      title1 'Test SAS_PUT using Implicit Passthru ';
      select distinct
         PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo;
   quit;
```

These lines are written to the SAS log.

```
libname dblib teradata user="sas" password="sas" server="sl96208"
   database=sas connection=shared;

NOTE: Libref DBLIB was successfully assigned, as follows:
      Engine:        TERADATA
      Physical Name: sl96208

         /*-- Set SQL debug global options --*/
         /*-----------------------------------*/
         options sastrace=',,,d' sastraceloc=saslog;

         /*-- Execute SQL using Implicit Passthru --*/
         /*-----------------------------------------*/
         proc sql noerrorstop;
           title1 'Test SAS_PUT using Implicit Passthru ';
           select distinct
              PUT(PRICE,Dollar8.2) AS PRICE_C
            from dblib.mailorderdemo
```

```
               ;

TERADATA_0: Prepared: on connection 0
SELECT * FROM sas."mailorderdemo"


TERADATA_1: Prepared: on connection 0
 select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
 as char(8)) as "PRICE_C" from "sas"."mailorderdemo"

TERADATA: trforc: COMMIT WORK
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.

TERADATA_2: Executed: on connection 0
 select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
 as char(8)) as "PRICE_C" from "sas"."mailorderdemo"

TERADATA: trget - rows to fetch: 9
TERADATA: trforc: COMMIT WORK

              Test SAS_PUT using Implicit Passthru                        9
                                    3:42 Thursday, September 25, 2010

                                PRICE_C
                                _______
                                  $8.00
                                 $10.00
                                 $12.00
                                 $13.59
                                 $13.99
                                 $14.00
                                 $27.98
                                 $48.99
                                 $54.00

        quit;
```

Be aware of these factors:

- The SQLMAPPUTTO= system option must be set to SAS_PUT to ensure that the SQL processor maps your PUT functions to the SAS_PUT() function and the SAS_PUT() reference is passed through to Teradata.
- The SAS SQL processor translates the PUT function in the SQL SELECT statement into a reference to the SAS_PUT() function.

  ```
  select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
     as char(8)) as "PRICE_C" from "sas"."mailorderdemo"
  ```

  A large value, VARCHAR(*n*), is always returned because one function prototype accesses all formats. Use the CAST expression to reduce the width of the returned column to be a character width that is reasonable for the format that is being used.

  The return text cannot contain a binary zero value (hexadecimal 00) because the SAS_PUT() function always returns a VARCHAR(*n*) data type and a Teradata VARCHAR(*n*) is defined to be a null-terminated string.

The SELECT DISTINCT clause executes inside Teradata, and the processing is distributed across all available data nodes. Teradata formats the price values with the

$DOLLAR8.2 format and processes the SELECT DISTINCT clause using the formatted values.

Explicit Use of the SAS_PUT() Function

If you use explicit pass-through (a direct connection to Teradata), you can use the SAS_PUT() function call in your SQL program.

This example shows the same query from "Implicit Use of the SAS_PUT() Function" on page 120 and explicitly uses the SAS_PUT() function call.

```
proc sql noerrorstop;
   title1 'Test SAS_PUT using Explicit Passthru';
   connect to teradata (user=sas password=XXX database=sas server=sl96208);

   select * from connection to teradata
      (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
         "PRICE_C" from mailorderdemo);

disconnect from teradata;
quit;
```

The following lines are written to the SAS log.

```
 proc sql noerrorstop;
 title1 'Test SAS_PUT using Explicit Passthru ';
 connect to teradata (user=sas password=XXX database=sas server=sl96208);

 select * from connection to teradata
    (select distinct cast(sas_put("PRICE",'DOLLAR8.2') as char(8)) as
        "PRICE_C" from mailorderdemo);

            Test SAS_PUT using Explicit Passthru                        10
                                        13:42 Thursday, September 25, 2010

                          PRICE_C
                          _______
                            $8.00
                           $10.00
                           $12.00
                           $13.59
                           $13.99
                           $14.00
                           $27.98
                           $48.99
                           $54.00

disconnect from teradata;
quit;
```

Note: If you explicitly use the SAS_PUT() function in your code, it is recommended that you use double quotation marks around a column name to avoid any ambiguity with the keywords. For example, if you did not use double quotation marks around the column name, DATE, in this example, all date values would be returned as today's date.

```
select distinct
   cast(sas_put("price", 'dollar8.2') as char(8)) as "price_c",
```

```
   cast(sas_put("date", 'date9.1') as char(9)) as "date_d",
   cast(sas_put("inv", 'best8.') as char(8))  as "inv_n",
   cast(sas_put("name", '$32.') as char(32)) as "name_n"
from mailorderdemo;
```

Tips When Using the SAS_PUT() Function in Teradata

- Format widths greater than 256 can cause unexpected or unsuccessful behavior.
- If a variable is associated with a $HEXw. format, SAS/ACCESS creates the DBMS table, and the PUT function is being mapped to the SAS_PUT() function, SAS/ACCESS assumes that variable is binary and assigns a data type of BYTE to that column. The SAS_PUT() function does not support the BYTE data type. Teradata reports an error that the SAS_PUT() function is not found instead of reporting that an incorrect data type was passed to the function. To avoid this error, variables that are processed by the SAS_PUT() function implicitly should not have the $HEXw. format associated with them. For more information, see "Data Types and the SAS_PUT() Function" on page 118.

  If you use the $HEXw. format in an explicit SAS_PUT() function call, this error does not occur.
- If you use the $HEXw. format in an explicit SAS_PUT() function call, blanks in the variable are converted to "20" but trailing blanks (blanks that occur when using a format width greater than the variable width) are trimmed. For example, the value "A " ("A" with a single blank) with a $HEX4. format is written as 4120. The value "A" ("A" with no blanks) with a $HEX4. format is written as 41 with no blanks.

Teradata Permissions

Because functions are associated with a database, the functions inherit the access rights of that database. It could be useful to create a separate shared database for scoring functions so that access rights can be customized as needed. In addition, you must have the following permissions to publish the scoring functions in Teradata:

- CREATE FUNCTION
- DROP FUNCTION
- EXECUTE FUNCTION
- ALTER FUNCTION

Without these permissions, the publishing of the SAS_PUT() function and formats fails. To obtain these permissions, contact your database administrator.

For more information about specific permissions, see the *SAS In-Database Products: Administrator's Guide*.

Part 4

In-Database Procedures

Chapter 14
Running SAS Procedures inside the Database *127*

Chapter 14

Running SAS Procedures inside the Database

Introduction to In-Database Procedures **127**

Running In-Database Procedures **128**

In-Database Procedures in DB2 under UNIX and PC Hosts, Netezza, and Oracle **129**

In-Database Procedures in Teradata **129**

In-Database Procedure Considerations and Limitations **130**
- Overview 130
- User-Defined Formats 130
- Row Order 131
- BY-Groups 131
- LIBNAME Statement 131
- Data Set-related Options 132
- Column Names in Netezza 132
- Miscellaneous Items 132

Using the MSGLEVEL Option to Control Messaging **133**

Introduction to In-Database Procedures

Using conventional processing, a SAS procedure (by means of the SAS/ACCESS engine) receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred.

Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries. These queries allow the aggregations and analytics to be run inside the database. Some of the in-database procedures generate SQL procedure syntax and use implicit pass-through to generate the native SQL. Other in-database procedures generate native SQL and use explicit pass-through. For more information about how a specific procedure works inside the database, see the documentation for that procedure.

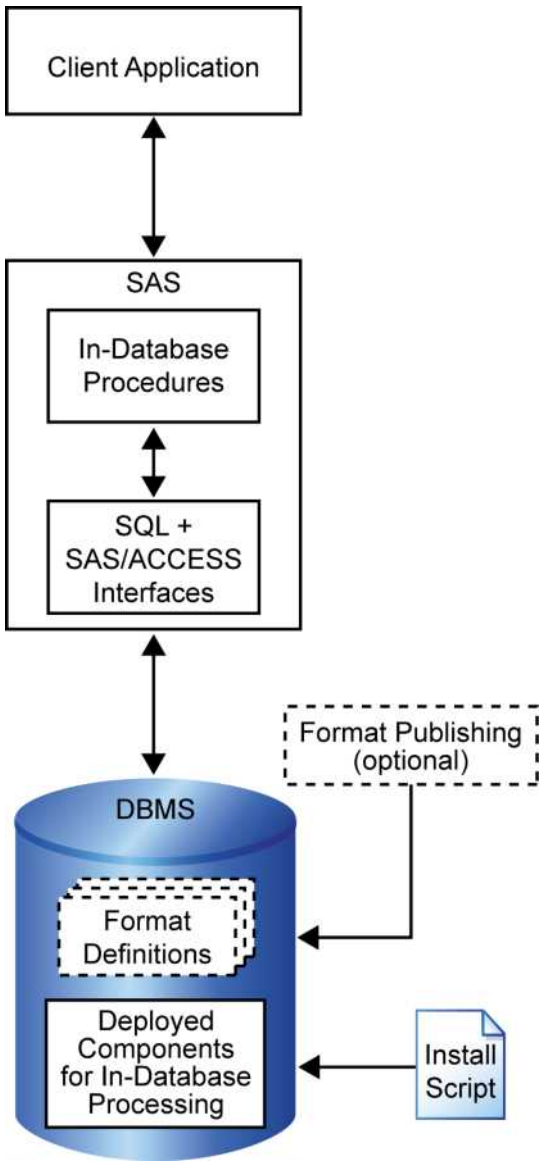
The queries submitted by SAS in-database procedures reference DBMS SQL functions and, in some cases, the special SAS functions that are deployed inside the database. One example of a special SAS function is the SAS_PUT() function that enables you to execute PUT function calls inside the database. Other examples are SAS functions for computing sum-of-squares-and-crossproducts (SSCP) matrices.

For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database

procedures, more work is done inside the database, and less data movement can occur. This could result in significant performance improvements.

This diagram illustrates the in-database procedure process.

Figure 14.1 *Process Flow Diagram*



Running In-Database Procedures

To run in-database procedures, these actions must be taken:

- The SQLGENERATION system option or the SQLGENERATION LIBNAME option must be set to DBMS or DBMS='*database-name*'.

  Note: By default, the SQLGENERATION system option is "on" for DB2, Netezza, Oracle, and Teradata.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the SQLGENERATION system option is set to NONE DBMS='DB2, NETEZZA ORACLE TERADATA' and the in-database procedures are run using conventional SAS processing for databases other than DB2, Netezza, Oracle, and Teradata. The procedures are not run inside the database.

Conventional SAS processing is also used when specific procedure statements and options do not support in-database processing. For complete information, see the "SQLGENERATION= System Option" on page 137 or the SQLGENERATION LIBNAME option in *SAS/ACCESS for Relational Databases: Reference*.

- The LIBNAME statement must point to a valid version of the DBMSs:
  - DB2 UDB9.5 Fixpack 7 running only on AIX or LINUX x64
  - Netezza 5.0 or higher
  - Teradata server running version 12 or higher for Linux
  - Oracle 9i

In-Database Procedures in DB2 under UNIX and PC Hosts, Netezza, and Oracle

The following Base SAS procedures have been enhanced for in-database processing inside DB2 under UNIX and PC Hosts, Netezza, and Oracle.

- FREQ
- RANK
- REPORT
- SORT
- SUMMARY/MEANS
- TABULATE

For more information about running a specific procedure inside the database, see the documentation for that procedure.

In-Database Procedures in Teradata

The following Base SAS, SAS Enterprise Miner, SAS/ETS, and SAS/STAT procedures have been enhanced for in-database processing.

- CORR*
- CANCORR*
- DMDB*
- DMINE*
- DMREG*
- FACTOR*

- FREQ
- PRINCOMP*
- RANK
- REG*
- REPORT
- SCORE*
- SORT
- SUMMARY/MEANS
- TABULATE
- TIMESERIES*
- VARCLUS*

*SAS Analytics Accelerator is required to run these procedures inside the database. For more information, see the *SAS Analytics Accelerator for Teradata: Guide*.

For more information about running a specific procedure inside the database, see the documentation for that procedure.

In-Database Procedure Considerations and Limitations

Overview

The considerations and limitations in the following sections apply to both Base SAS and SAS/STAT in-database procedures.

Note: Each in-database procedure has its own specific considerations and limitations. For more information, see the documentation for the procedure.

User-Defined Formats

If you are using in-database procedures with user-defined formats that were published in the database, you must have a local copy of the user-defined formats. Without the local copy, the procedure fails.

Note: The local copy of the user-defined format must be identical in both name and function to the format that is published to the database. If they are not identical, the following actions occur.

- A “check sum ERROR” warning is produced. The warning indicates that the local and published formats differ.
- The local format is used, and the query is processed by SAS instead of inside the database.

If this occurs, you can redefine the local format to match the published version and rerun the procedure inside the database.

For more information about publishing user-defined formats, see the section on deploying and using formats for your database in Part 3, “Format Publishing and the SAS_PUT() Function”.

Row Order

- DBMS tables have no inherent order for the rows. Therefore, the BY statement with the OBS option and the FIRSTOBS option prevents in-database processing.
- The order of rows written to a database table from a SAS procedure is not likely to be preserved. For example, the SORT procedure can output a SAS data set that contains ordered observations. If the results are written to a database table, the order of rows within that table might not be preserved because the DBMS has no obligation to maintain row order.
- You can print a table using the SQL procedure with an ORDER BY clause to get consistent row order. Another option is to use the SORT procedure to create an ordinary SAS data set and use the PRINT procedure on that SAS data set.

BY-Groups

BY-group processing is handled by SAS for Base SAS procedures. Raw results are returned from the DBMS, and SAS BY-group processing applies formats as necessary to create the BY group.

For SAS/STAT procedures, formats can be applied, and BY-group processing can occur inside the DBMS if the SAS_PUT() function and formats are published to the DBMS. For more information, see the *SAS Analytics Accelerator for Teradata: Guide*.

These BY statement option settings apply to the in-database procedures:

- The DESCENDING option is supported.
- The NOTSORTED option is ignored because the data is always returned in sorted order.

By default, when SAS/ACCESS creates a database table, SAS/ACCESS uses the SAS formats that are assigned to variables to decide which DBMS data types to assign to the DBMS columns. If you specify the DBFMTIGNORE system option for numeric formats, SAS/ACCESS creates DBMS columns with a DOUBLE PRECISION data type. For more information, see the LIBNAME Statement for Relational Databases, “LIBNAME Statement Data Conversions,” and the DBFMTIGNORE system option in *SAS/ACCESS for Relational Databases: Reference*.

LIBNAME Statement

- These LIBNAME statement options and settings prevent in-database processing:
  - DBMSTEMP=YES
  - DBCONINIT
  - DBCONTERM
  - DBGEN_NAME=SAS
  - PRESERVE_NAMES=NO
  - MODE=TERADATA
- LIBNAME concatenation prevents in-database processing.

Data Set-related Options

These data set options and settings prevent in-database processing:

- RENAME= on a data set.
- OUT= data set on DBMS and DATA= data set not on DBMS.

  For example, you can have *data=td.foo* and *out=work.fooout* where WORK is the Base SAS engine.
- DATA= and OUT= data sets are the same DBMS table.
- OBS= and FIRSTOBS= on DATA= data set.

Column Names in Netezza

Column names that start with an underscore are not allowed in Netezza.

An error occurs if you try to create an output table in Netezza that contains a column whose name starts with an underscore. The workaround for this is to send the output table to the SAS Work directory.

Miscellaneous Items

These items prevent in-database processing:

- DBMSs do not support SAS passwords.
- SAS encryption requires passwords that are not supported.
- Teradata does not support generation options that are explicitly specified in the procedure step, and the procedure does not know whether a generation number is explicit or implicit.
- When the database resolves function references. the database searches in this order:

  1. fully qualified object name
  2. current database
  3. SYSLIB

  If you need to reference functions that are published in a nonsystem, nondefault database, you must use one of these methods:

  - explicit SQL
  - DATABASE= LIBNAME option
  - map the fully qualified name (*schema*.sas_put) in the external mapping

Using the MSGLEVEL Option to Control Messaging

The MSGLEVEL system option specifies the level of detail in messages that are written to the SAS log. When the MSGLEVEL option is set to N (the default value) these messages are printed to the SAS log:

- a note that says SQL is used for in-database computations when in-database processing is performed
- error messages if something goes wrong with the SQL commands that are submitted for in-database computations
- if there are SQL error messages, a note that says whether SQL is used

When the MSGLEVEL option is set to I, all the messages that are printed when MSGLEVEL=N are printed to the SAS log.

These messages are also printed to the SAS log:

- A note that explains why SQL was not used for in-database computations, if SQL is not used.

  Note: No note is printed if you specify SQLGENERATION=NONE.

- A note that says that SQL cannot be used because there are no observations in the data source.

  Note: This information is not always available to the procedure.

- If you try to create a special SAS data set as a DBMS table for PROC MEANS or PROC SUMMARY, a note that says that the TYPE= attribute is not stored in DBMS tables.
- If you are using a format that SAS supplies or a user-defined format, a note that says if the format was or was not found in the database.

Part 5

System Options Reference

Chapter 15
System Options That Affect In-Database Processing *137*

Chapter 15

System Options That Affect In-Database Processing

Dictionary **137**
SQLGENERATION= System Option 137
SQLMAPPUTTO= System Option 140
SQLREDUCEPUT= System Option 141

Dictionary

SQLGENERATION= System Option

Specifies whether and when SAS procedures generate SQL for in-database processing of source data.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Default: NONE DBMS='Teradata DB2 ORACLE NETEZZA'

Restriction: For DBMS= and EXCLUDEDB= values, the maximum length of an engine name is eight characters. For the EXCLUDEPROC= value, the maximum length of a procedure name is 16 characters. An engine can appear only once, and a procedure can appear only once for a given engine.

Data source: DB2 under UNIX and PC Hosts, Netezza, Oracle, Teradata

Syntax

SQLGENERATION=<(>NONE | DBMS <DBMS='*engine1... enginen* '>
<EXCLUDEDB=*engine* | '*engine1... enginen*'>
<EXCLUDEPROC="*engine*='*proc1... procn*' *enginen*='*proc1... procn*' ">)>

SQLGENERATION=" "

Syntax Description

NONE
prevents those SAS procedures that are enabled for in-database processing from generating SQL for in-database processing. This is a primary state.

DBMS
allows SAS procedures that are enabled for in-database processing to generate SQL for in-database processing of DBMS tables through supported SAS/ACCESS engines. This is a primary state.

DBMS='*engine1... enginen*'
specifies one or more SAS/ACCESS engines. It modifies the primary state.

EXCLUDEDB=*engine* | '*engine1... enginen*'
prevents SAS procedures from generating SQL for in-database processing for one or more specified SAS/ACCESS engines.

EXCLUDEPROC="*engine*='*proc1... procn*' *enginen*='*proc1... procn*' "
identifies engine-specific SAS procedures that do not support in-database processing.

" "
resets the value to the default that was shipped.

Details

Use this option with such procedures as PROC FREQ to indicate what SQL is generated for in-database processing based on the type of subsetting that you need and the SAS/ACCESS engines that you want to access the source table.

You must specify NONE (which indicates the primary state) and DBMS.

The maximum length of the option value is 4096. Also, parentheses are required when this option value contains multiple keywords.

Not all procedures support SQL generation for in-database processing for every engine type. If you specify a setting that is not supported, an error message indicates the level of SQL generation that is not supported, and the procedure can reset to the default so that source table records can be read and processed within SAS. If this is not possible, the procedure ends and sets SYSERR= as needed.

You can specify different SQLGENERATION= values for the DATA= and OUT= data sets by using different LIBNAME statements for each of these data sets.

Here is how SAS/ACCESS handles precedence.

Table 15.1 *Precedence of Values for SQLGENERATION= LIBNAME and System Options*

LIBNAME Option	PROC EXCLUDE on System Option?	Engine Type	Engine Specified on System Option	Resulting Value	From (option)
not set NONE DBMS	yes	database interface	NONE DBMS	NONE EXCLUDEDB	system
NONE	no			NONE	LIBNAME
DBMS				DBMS	
not set			NONE	NONE	system
NONE DBMS			DBMS	DBMS	LIBNAME
		no SQL generated for this database host or database version	NONE DBMS	NONE	
not set		Base			system
NONE DBMS					LIBNAME

Example

Here is the default that is shipped with the product.

```
options sqlgeneration='';
proc options option=sqlgeneration;
 run;
```

SAS procedures generate SQL for in-database processing for all databases except DB2 in this example.

```
options sqlgeneration='';
options sqlgeneration=(DBMS EXCLUDEDB='DB2');
proc options option=sqlgeneration;
run;
```

In this example, in-database processing occurs only for Teradata, but SAS procedures generate no SQL for in-database processing.

```
options sqlgeneration='';
options SQLGENERATION = (NONE DBMS='Teradata');
proc options option=sqlgeneration;
run;
```

In this next example, SAS procedures do not generate SQL for in-database processing even though in-database processing occurs only for Teradata.

```
options sqlgeneration='';
Options SQLGENERATION = (NONE DBMS='Teradata' EXCLUDEDB='DB2');
proc options option=sqlgeneration;
run;
```

For this example, PROC1 and PROC2 for Oracle are not processed inside the database. SAS procedures for Oracle that support in-database processing do not generate SQL for in-database processing, and in-database processing occurs only for Teradata.

```
options sqlgeneration='';
Options SQLGENERATION = (NONE EXCLUDEPROC="oracle='proc1,proc2'"
        DBMS='Teradata' EXCLUDEDB='ORACLE');
proc options option=sqlgeneration;
run;
```

See Also

- "Running In-Database Procedures" on page 128

LIBNAME Options:

- SQLGENERATION= LIBNAME option in *SAS/ACCESS for Relational Databases: Reference*

SQLMAPPUTTO= System Option

Specifies whether the PUT function is mapped to the SAS_PUT() function for a database. This is also possible where the SAS_PUT() function is mapped.

Valid in: configuration file, SAS invocation, OPTIONS statement

Default: SAS_PUT

Data source: DB2 under UNIX, Netezza, Teradata

Syntax

SQLMAPPUTTO= NONE | SAS_PUT | (*database*.SAS_PUT)

Syntax Description

NONE
specifies to PROC SQL that no PUT mapping is to occur.

SAS_PUT
specifies that the PUT function be mapped to the SAS_PUT() function.

***database*.SAS_PUT**
specifies the database name.

Requirement: If you specify a database name, you must enclose the entire argument in parentheses.

Tips:
It is not necessary that the format definitions and the SAS_PUT() function reside in the same database as the one that contains the data that you want to format. You can use the *database*.SAS_PUT argument to specify the database where the format definitions and the SAS_PUT() function have been published.

The database name can be a multilevel name and it can include blanks.

Details

The format publishing macros deploy, or publish, the PUT function implementation to the database as a new function named SAS_PUT(). The format publishing macros also publish both user-defined formats that you create using PROC FORMAT and most formats that SAS supplies. The SAS_PUT() function supports the use of SAS formats. You can use the function in SQL queries that SAS submits to the database so that the entire SQL query can be processed inside the database. You can also use it in conjunction with in-database procedures.

You can use this option with the SQLREDUCEPUT=, SQLREDUCEPUTOBS, and SQLREDUCEPUTVALUES= system options. For more information about these options, see the *SAS SQL Procedure User's Guide*.

See Also

- Chapter 11, "Deploying and Using SAS Formats in DB2 under UNIX," on page 91
- Chapter 12, "Deploying and Using SAS Formats in Netezza," on page 103
- Chapter 13, "Deploying and Using SAS Formats in Teradata," on page 113
- "BY-Groups" on page 131

LIBNAME Options:

- SQLMAPPUTTO= LIBNAME option in *SAS/ACCESS for Relational Databases: Reference*

SQLREDUCEPUT= System Option

For the SQL procedure, specifies the engine type that a query uses for which optimization is performed by replacing a PUT function in a query with a logically equivalent expression.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Categories: Files: SAS Files
System administration: SQL
System administration: Performance

PROC OPTIONS GROUP= SASFILES
SQL
PERFORMANCE

Note: This option can be restricted by a site administrator. For more information, see the section on restricted options in *SAS System Options: Reference*.

Syntax

SQLREDUCEPUT= ALL | NONE | DBMS | BASE

Syntax Description

ALL
specifies that optimization is performed on all PUT functions regardless of any engine that is used by the query to access the data.

NONE
specifies that no optimization is to be performed.

DBMS
specifies that optimization is performed on all PUT functions whose query is performed by a SAS/ACCESS engine. This is the default.

Requirement: The first argument to the PUT function must be a variable obtained by a table that is accessed using a SAS/ACCESS engine.

BASE
specifies that optimization is performed on all PUT functions whose query is performed by a SAS/ACCESS engine or a Base SAS engine.

Details

If you specify the SQLREDUCEPUT= system option, SAS optimizes the PUT function as much as possible before the query is executed. If the query also contains a WHERE clause, the evaluation of the WHERE clause is simplified. The following SELECT statements are examples of queries that would be reduced if this option was set to any value other than **none**:

```
select x, y from &lib..b where (PUT(x, abc.) in ('yes', 'no'));
select x from &lib..a where (PUT(x, udfmt.) = trim(left('small')));
```

If both the SQLREDUCEPUT= system option and the SQLCONSTDATETIME system option are specified, PROC SQL replaces the DATE, TIME, DATETIME, and TODAY functions with their respective values to determine the PUT function value before the query executes.

The following two SELECT clauses show the original query and the optimized query:

```
select x from &lib..c where (put(bday, date9.) = put(today(), date9.));
```

Here, the SELECT clause is optimized.

```
select x from &lib..c where (x = '17Mar2011'D);
```

If a query does not contain the PUT function, it is not optimized.

Note: The value that is specified in the SQLREDUCEPUT= system option is in effect for all SQL procedure statements, unless the PROC SQL REDUCEPUT= option is set. The value of the REDUCEPUT= option takes precedence over the SQLREDUCEPUT= system option. However, changing the value of the REDUCEPUT= option does not change the value of the SQLREDUCEPUT= system option.

See Also

- "Improving Query Performance" in the *SAS SQL Procedure User's Guide*

Procedure Statement Options:

- PROC SQL Statement REDUCEPUT= option in the *SAS SQL Procedure User's Guide*

System Options:

- SQLCONSTDATETIME and SQLREDUCEPUTOBS in the *SAS SQL Procedure User's Guide*

Part 6

Appendixes

Appendix 1
Scoring File Examples for Aster nCluster *147*

Appendix 1

Scoring File Examples for Aster *n*Cluster

Example of a .ds2 Scoring File 147
Example of an Input and Output Variables Scoring File 167
Example of a User-Defined Formats Scoring File 174

Example of a .ds2 Scoring File

This is an example of a .ds2 scoring file. The filename is sasscore_score.ds2.

```
data &ASTER_OUTPUT;
 #_local _LPMAX;
 #_local _P4;
 #_local _P3;
 #_local _P2;
 #_local _P1;
 #_local _P0;
 #_local _IY;
 #_local _MAXP;
 #_local _LP3;
 #_local _LP2;
 #_local _LP1;
 #_local _LP0;
 #_local _TEMP;
 #_local _7_1;
 #_local _7_0;
 #_local _6_2;
 #_local _6_1;
 #_local _6_0;
 #_local _5_14;
 #_local _5_13;
 #_local _5_12;
 #_local _5_11;
 #_local _5_10;
 #_local _5_9;
 #_local _5_8;
 #_local _5_7;
 #_local _5_6;
 #_local _5_5;
 #_local _5_4;
```

```
 #_local _5_3;
 #_local _5_2;
 #_local _5_1;
 #_local _5_0;
 #_local _3_10;
 #_local _3_9;
 #_local _3_8;
 #_local _3_7;
 #_local _3_6;
 #_local _3_5;
 #_local _3_4;
 #_local _3_3;
 #_local _3_2;
 #_local _3_1;
 #_local _3_0;
 #_local _2_12;
 #_local _2_11;
 #_local _2_10;
 #_local _2_9;
 #_local _2_8;
 #_local _2_7;
 #_local _2_6;
 #_local _2_5;
 #_local _2_4;
 #_local _2_3;
 #_local _2_2;
 #_local _2_1;
 #_local _2_0;
 #_local _DM_FIND;
 #_local _1_14;
 #_local _1_13;
 #_local _1_12;
 #_local _1_11;
 #_local _1_10;
 #_local _1_9;
 #_local _1_8;
 #_local _1_7;
 #_local _1_6;
 #_local _1_5;
 #_local _1_4;
 #_local _1_3;
 #_local _1_2;
 #_local _1_1;
 #_local _1_0;
 #_local _DM_BAD;
dcl char(4) _WARN_;
dcl char(6) I_ATTACK;
dcl char(6) U_ATTACK;
dcl char(32) EM_CLASSIFICATION;
dcl double COUNT;
dcl double DIF_SRVR;
dcl char(32) FLAG;
dcl double HOT;
dcl double SAM_SRAT;
dcl char(32) SERVICE;
dcl double SRV_CNT;
```

```
method run();
 dcl char(8) _NORM8;
 dcl char(8) _NORM8;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(6) REGDRU[5];
 dcl char(6) REGDRF[5];
REGDRU:=('u2r   ', 'r2l   ', 'probe ', 'normal', 'dos   ');
REGDRF:=('U2R', 'R2L', 'PROBE', 'NORMAL', 'DOS');
set &ASTER_INPUT;
_WARN_ = ' ';
if (COUNT = .) then AOV16_COUNT = 8.0;
 else if (COUNT <= 31.9375) then AOV16_COUNT = 1.0;
 else if (COUNT <= 63.875) then AOV16_COUNT = 2.0;
 else if (COUNT <= 95.8125) then AOV16_COUNT = 3.0;
 else if (COUNT <= 127.75) then AOV16_COUNT = 4.0;
 else if (COUNT <= 159.6875) then AOV16_COUNT = 5.0;
 else if (COUNT <= 191.625) then AOV16_COUNT = 6.0;
 else if (COUNT <= 223.5625) then AOV16_COUNT = 7.0;
 else if (COUNT <= 255.5) then AOV16_COUNT = 8.0;
 else if (COUNT <= 287.4375) then AOV16_COUNT = 9.0;
 else if (COUNT <= 319.375) then AOV16_COUNT = 10.0;
 else if (COUNT <= 351.3125) then AOV16_COUNT = 11.0;
 else if (COUNT <= 383.25) then AOV16_COUNT = 12.0;
 else if (COUNT <= 415.1875) then AOV16_COUNT = 13.0;
 else if (COUNT <= 447.125) then AOV16_COUNT = 14.0;
 else if (COUNT <= 479.0625) then AOV16_COUNT = 15.0;
 else AOV16_COUNT = 16.0;
if (SRV_CNT = .) then AOV16_SRV_CNT = 7.0;
 else if (SRV_CNT <= 31.9375) then AOV16_SRV_CNT = 1.0;
 else if (SRV_CNT <= 63.875) then AOV16_SRV_CNT = 2.0;
 else if (SRV_CNT <= 95.8125) then AOV16_SRV_CNT = 3.0;
 else if (SRV_CNT <= 127.75) then AOV16_SRV_CNT = 4.0;
 else if (SRV_CNT <= 159.6875) then AOV16_SRV_CNT = 5.0;
 else if (SRV_CNT <= 191.625) then AOV16_SRV_CNT = 6.0;
 else if (SRV_CNT <= 223.5625) then AOV16_SRV_CNT = 7.0;
 else if (SRV_CNT <= 255.5) then AOV16_SRV_CNT = 8.0;
 else if (SRV_CNT <= 287.4375) then AOV16_SRV_CNT = 9.0;
 else if (SRV_CNT <= 319.375) then AOV16_SRV_CNT = 10.0;
 else if (SRV_CNT <= 351.3125) then AOV16_SRV_CNT = 11.0;
 else if (SRV_CNT <= 383.25) then AOV16_SRV_CNT = 12.0;
 else if (SRV_CNT <= 415.1875) then AOV16_SRV_CNT = 13.0;
 else if (SRV_CNT <= 447.125) then AOV16_SRV_CNT = 14.0;
 else if (SRV_CNT <= 479.0625) then AOV16_SRV_CNT = 15.0;
 else AOV16_SRV_CNT = 16.0;
if (SAM_SRAT = .) then AOV16_SAM_SRAT = 14.0;
 else if (SAM_SRAT <= 0.0625) then AOV16_SAM_SRAT = 1.0;
 else if (SAM_SRAT <= 0.125) then AOV16_SAM_SRAT = 2.0;
 else if (SAM_SRAT <= 0.1875) then AOV16_SAM_SRAT = 3.0;
 else if (SAM_SRAT <= 0.25) then AOV16_SAM_SRAT = 4.0;
 else if (SAM_SRAT <= 0.3125) then AOV16_SAM_SRAT = 5.0;
 else if (SAM_SRAT <= 0.375) then AOV16_SAM_SRAT = 6.0;
```

```
 else if (SAM_SRAT <= 0.4375) then AOV16_SAM_SRAT = 7.0;
 else if (SAM_SRAT <= 0.5) then AOV16_SAM_SRAT = 8.0;
 else if (SAM_SRAT <= 0.5625) then AOV16_SAM_SRAT = 9.0;
 else if (SAM_SRAT <= 0.625) then AOV16_SAM_SRAT = 10.0;
 else if (SAM_SRAT <= 0.6875) then AOV16_SAM_SRAT = 11.0;
 else if (SAM_SRAT <= 0.75) then AOV16_SAM_SRAT = 12.0;
 else if (SAM_SRAT <= 0.8125) then AOV16_SAM_SRAT = 13.0;
 else if (SAM_SRAT <= 0.875) then AOV16_SAM_SRAT = 14.0;
 else if (SAM_SRAT <= 0.9375) then AOV16_SAM_SRAT = 15.0;
 else AOV16_SAM_SRAT = 16.0;
if (DIF_SRVR = .) then AOV16_DIF_SRVR = 1.0;
 else if (DIF_SRVR <= 0.0625) then AOV16_DIF_SRVR = 1.0;
 else if (DIF_SRVR <= 0.125) then AOV16_DIF_SRVR = 2.0;
 else if (DIF_SRVR <= 0.1875) then AOV16_DIF_SRVR = 3.0;
 else if (DIF_SRVR <= 0.25) then AOV16_DIF_SRVR = 4.0;
 else if (DIF_SRVR <= 0.3125) then AOV16_DIF_SRVR = 5.0;
 else if (DIF_SRVR <= 0.375) then AOV16_DIF_SRVR = 6.0;
 else if (DIF_SRVR <= 0.4375) then AOV16_DIF_SRVR = 7.0;
 else if (DIF_SRVR <= 0.5) then AOV16_DIF_SRVR = 8.0;
 else if (DIF_SRVR <= 0.5625) then AOV16_DIF_SRVR = 9.0;
 else if (DIF_SRVR <= 0.625) then AOV16_DIF_SRVR = 10.0;
 else if (DIF_SRVR <= 0.6875) then AOV16_DIF_SRVR = 11.0;
 else if (DIF_SRVR <= 0.75) then AOV16_DIF_SRVR = 12.0;
 else if (DIF_SRVR <= 0.8125) then AOV16_DIF_SRVR = 13.0;
 else if (DIF_SRVR <= 0.875) then AOV16_DIF_SRVR = 14.0;
 else if (DIF_SRVR <= 0.9375) then AOV16_DIF_SRVR = 15.0;
 else AOV16_DIF_SRVR = 16.0;
if (HOT = .) then AOV16_HOT = 1.0;
 else if (HOT <= 1.875) then AOV16_HOT = 1.0;
 else if (HOT <= 3.75) then AOV16_HOT = 2.0;
 else if (HOT <= 5.625) then AOV16_HOT = 3.0;
 else if (HOT <= 7.5) then AOV16_HOT = 4.0;
 else if (HOT <= 9.375) then AOV16_HOT = 5.0;
 else if (HOT <= 11.25) then AOV16_HOT = 6.0;
 else if (HOT <= 13.125) then AOV16_HOT = 7.0;
 else if (HOT <= 15.0) then AOV16_HOT = 8.0;
 else if (HOT <= 16.875) then AOV16_HOT = 9.0;
 else if (HOT <= 18.75) then AOV16_HOT = 10.0;
 else if (HOT <= 20.625) then AOV16_HOT = 11.0;
 else if (HOT <= 22.5) then AOV16_HOT = 12.0;
 else if (HOT <= 24.375) then AOV16_HOT = 13.0;
 else if (HOT <= 26.25) then AOV16_HOT = 14.0;
 else if (HOT <= 28.125) then AOV16_HOT = 15.0;
 else AOV16_HOT = 16.0;
_NORM8 = DMNORM(SERVICE, 32.0);
select (_NORM8);
when ('IRC     ') G_SERVICE = 2.0;
when ('X11     ') G_SERVICE = 2.0;
when ('Z39_50  ') G_SERVICE = 1.0;
when ('AUTH    ') G_SERVICE = 2.0;
when ('BGP     ') G_SERVICE = 0.0;
when ('COURIER ') G_SERVICE = 1.0;
when ('CSNET_NS') G_SERVICE = 1.0;
when ('CTF     ') G_SERVICE = 0.0;
when ('DAYTIME ') G_SERVICE = 1.0;
when ('DISCARD ') G_SERVICE = 0.0;
```

```
when ('DOMAIN  ') G_SERVICE = 1.0;
when ('DOMAIN_U') G_SERVICE = 2.0;
when ('ECHO    ') G_SERVICE = 0.0;
when ('ECO_I   ') G_SERVICE = 2.0;
when ('ECR_I   ') G_SERVICE = 0.0;
when ('EFS     ') G_SERVICE = 1.0;
when ('EXEC    ') G_SERVICE = 0.0;
when ('FINGER  ') G_SERVICE = 2.0;
when ('FTP     ') G_SERVICE = 2.0;
when ('FTP_DATA') G_SERVICE = 2.0;
when ('GOPHER  ') G_SERVICE = 1.0;
when ('HOSTNAME') G_SERVICE = 0.0;
when ('HTTP    ') G_SERVICE = 2.0;
when ('HTTP_443') G_SERVICE = 0.0;
when ('IMAP4   ') G_SERVICE = 1.0;
when ('ISO_TSAP') G_SERVICE = 0.0;
when ('KLOGIN  ') G_SERVICE = 0.0;
when ('KSHELL  ') G_SERVICE = 0.0;
when ('LDAP    ') G_SERVICE = 0.0;
when ('LINK    ') G_SERVICE = 1.0;
when ('LOGIN   ') G_SERVICE = 0.0;
when ('MTP     ') G_SERVICE = 1.0;
when ('NAME    ') G_SERVICE = 0.0;
when ('NETBIOS_') G_SERVICE = 0.0;
when ('NETSTAT ') G_SERVICE = 0.0;
when ('NNSP    ') G_SERVICE = 0.0;
when ('NNTP    ') G_SERVICE = 1.0;
when ('NTP_U   ') G_SERVICE = 2.0;
when ('OTHER   ') G_SERVICE = 2.0;
when ('POP_2   ') G_SERVICE = 0.0;
when ('POP_3   ') G_SERVICE = 2.0;
when ('PRINTER ') G_SERVICE = 1.0;
when ('PRIVATE ') G_SERVICE = 1.0;
when ('RED_I   ') G_SERVICE = 2.0;
when ('REMOTE_J') G_SERVICE = 1.0;
when ('RJE     ') G_SERVICE = 1.0;
when ('SHELL   ') G_SERVICE = 0.0;
when ('SMTP    ') G_SERVICE = 2.0;
when ('SQL_NET ') G_SERVICE = 0.0;
when ('SSH     ') G_SERVICE = 1.0;
when ('SUNRPC  ') G_SERVICE = 1.0;
when ('SUPDUP  ') G_SERVICE = 1.0;
when ('SYSTAT  ') G_SERVICE = 1.0;
when ('TELNET  ') G_SERVICE = 2.0;
when ('TFTP_U  ') G_SERVICE = 2.0;
when ('TIM_I   ') G_SERVICE = 1.0;
when ('TIME    ') G_SERVICE = 2.0;
when ('URH_I   ') G_SERVICE = 2.0;
when ('URP_I   ') G_SERVICE = 2.0;
when ('UUCP    ') G_SERVICE = 1.0;
when ('UUCP_PAT') G_SERVICE = 0.0;
when ('VMNET   ') G_SERVICE = 1.0;
when ('WHOIS   ') G_SERVICE = 1.0;
otherwise _WARN_ = 'U';
end;
_NORM8 = DMNORM(FLAG, 32.0);
```

```
select (_NORM8);
when ('OTH     ') G_FLAG = 3.0;
when ('REJ     ') G_FLAG = 2.0;
when ('RSTO    ') G_FLAG = 2.0;
when ('RSTOS0  ') G_FLAG = 3.0;
when ('RSTR    ') G_FLAG = 3.0;
when ('S0      ') G_FLAG = 0.0;
when ('S1      ') G_FLAG = 3.0;
when ('S2      ') G_FLAG = 3.0;
when ('S3      ') G_FLAG = 3.0;
when ('SF      ') G_FLAG = 1.0;
when ('SH      ') G_FLAG = 3.0;
otherwise _WARN_ = 'U';
end;
_DM_BAD = 0.0;
_1_0 = 0.0;
_1_1 = 0.0;
_1_2 = 0.0;
_1_3 = 0.0;
_1_4 = 0.0;
_1_5 = 0.0;
_1_6 = 0.0;
_1_7 = 0.0;
_1_8 = 0.0;
_1_9 = 0.0;
_1_10 = 0.0;
_1_11 = 0.0;
_1_12 = 0.0;
_1_13 = 0.0;
_1_14 = 0.0;
if MISSING(AOV16_COUNT) then do ;
_1_0 = .;
_1_1 = .;
_1_2 = .;
_1_3 = .;
_1_4 = .;
_1_5 = .;
_1_6 = .;
_1_7 = .;
_1_8 = .;
_1_9 = .;
_1_10 = .;
_1_11 = .;
_1_12 = .;
_1_13 = .;
_1_14 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_COUNT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
_DM_FIND = 0.0;
if _DM12 <= '16' then do ;
if _DM12 <= '12' then do ;
if _DM12 <= '10' then do ;
```

```
if _DM12 = '1' then do ;
_1_0 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '10' then do ;
_1_9 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '11' then do ;
_1_10 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '12' then do ;
_1_11 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
 else do ;
if _DM12 <= '14' then do ;
if _DM12 = '13' then do ;
_1_12 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '14' then do ;
_1_13 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '15' then do ;
_1_14 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '16' then do ;
_1_0 = -1.0;
_1_1 = -1.0;
_1_2 = -1.0;
_1_3 = -1.0;
_1_4 = -1.0;
_1_5 = -1.0;
_1_6 = -1.0;
_1_7 = -1.0;
_1_8 = -1.0;
_1_9 = -1.0;
_1_10 = -1.0;
_1_11 = -1.0;
```

```
_1_12 = -1.0;
_1_13 = -1.0;
_1_14 = -1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
end;
 else do ;
if _DM12 <= '5' then do ;
if _DM12 <= '3' then do ;
if _DM12 = '2' then do ;
_1_1 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '3' then do ;
_1_2 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '4' then do ;
_1_3 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '5' then do ;
_1_4 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
 else do ;
if _DM12 <= '7' then do ;
if _DM12 = '6' then do ;
_1_5 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '7' then do ;
_1_6 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '8' then do ;
_1_7 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '9' then do ;
```

```
_1_8 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
end;
if ^_DM_FIND then do ;
_1_0 = .;
_1_1 = .;
_1_2 = .;
_1_3 = .;
_1_4 = .;
_1_5 = .;
_1_6 = .;
_1_7 = .;
_1_8 = .;
_1_9 = .;
_1_10 = .;
_1_11 = .;
_1_12 = .;
_1_13 = .;
_1_14 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_2_0 = 0.0;
_2_1 = 0.0;
_2_2 = 0.0;
_2_3 = 0.0;
_2_4 = 0.0;
_2_5 = 0.0;
_2_6 = 0.0;
_2_7 = 0.0;
_2_8 = 0.0;
_2_9 = 0.0;
_2_10 = 0.0;
_2_11 = 0.0;
_2_12 = 0.0;
if MISSING(AOV16_DIF_SRVR) then do ;
_2_0 = .;
_2_1 = .;
_2_2 = .;
_2_3 = .;
_2_4 = .;
_2_5 = .;
_2_6 = .;
_2_7 = .;
_2_8 = .;
_2_9 = .;
_2_10 = .;
_2_11 = .;
_2_12 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
```

```
end;
 else do ;
_DM12 = put(AOV16_DIF_SRVR, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_2_0 = 1.0;
end;
 else if _DM12 = '2' then do ;
_2_1 = 1.0;
end;
 else if _DM12 = '16' then do ;
_2_0 = -1.0;
_2_1 = -1.0;
_2_2 = -1.0;
_2_3 = -1.0;
_2_4 = -1.0;
_2_5 = -1.0;
_2_6 = -1.0;
_2_7 = -1.0;
_2_8 = -1.0;
_2_9 = -1.0;
_2_10 = -1.0;
_2_11 = -1.0;
_2_12 = -1.0;
end;
 else if _DM12 = '11' then do ;
_2_10 = 1.0;
end;
 else if _DM12 = '8' then do ;
_2_7 = 1.0;
end;
 else if _DM12 = '10' then do ;
_2_9 = 1.0;
end;
 else if _DM12 = '3' then do ;
_2_2 = 1.0;
end;
 else if _DM12 = '7' then do ;
_2_6 = 1.0;
end;
 else if _DM12 = '4' then do ;
_2_3 = 1.0;
end;
 else if _DM12 = '9' then do ;
_2_8 = 1.0;
end;
 else if _DM12 = '5' then do ;
_2_4 = 1.0;
end;
 else if _DM12 = '12' then do ;
_2_11 = 1.0;
end;
 else if _DM12 = '6' then do ;
_2_5 = 1.0;
end;
 else if _DM12 = '13' then do ;
```

```
_2_12 = 1.0;
end;
 else do ;
_2_0 = .;
_2_1 = .;
_2_2 = .;
_2_3 = .;
_2_4 = .;
_2_5 = .;
_2_6 = .;
_2_7 = .;
_2_8 = .;
_2_9 = .;
_2_10 = .;
_2_11 = .;
_2_12 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_3_0 = 0.0;
_3_1 = 0.0;
_3_2 = 0.0;
_3_3 = 0.0;
_3_4 = 0.0;
_3_5 = 0.0;
_3_6 = 0.0;
_3_7 = 0.0;
_3_8 = 0.0;
_3_9 = 0.0;
_3_10 = 0.0;
if MISSING(AOV16_HOT) then do ;
_3_0 = .;
_3_1 = .;
_3_2 = .;
_3_3 = .;
_3_4 = .;
_3_5 = .;
_3_6 = .;
_3_7 = .;
_3_8 = .;
_3_9 = .;
_3_10 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_HOT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_3_0 = 1.0;
end;
 else if _DM12 = '2' then do ;
_3_1 = 1.0;
end;
 else if _DM12 = '15' then do ;
```

```
_3_10 = 1.0;
end;
 else if _DM12 = '3' then do ;
_3_2 = 1.0;
end;
 else if _DM12 = '4' then do ;
_3_3 = 1.0;
end;
 else if _DM12 = '11' then do ;
_3_7 = 1.0;
end;
 else if _DM12 = '12' then do ;
_3_8 = 1.0;
end;
 else if _DM12 = '10' then do ;
_3_6 = 1.0;
end;
 else if _DM12 = '8' then do ;
_3_5 = 1.0;
end;
 else if _DM12 = '16' then do ;
_3_0 = -1.0;
_3_1 = -1.0;
_3_2 = -1.0;
_3_3 = -1.0;
_3_4 = -1.0;
_3_5 = -1.0;
_3_6 = -1.0;
_3_7 = -1.0;
_3_8 = -1.0;
_3_9 = -1.0;
_3_10 = -1.0;
end;
 else if _DM12 = '13' then do ;
_3_9 = 1.0;
end;
 else if _DM12 = '7' then do ;
_3_4 = 1.0;
end;
 else do ;
_3_0 = .;
_3_1 = .;
_3_2 = .;
_3_3 = .;
_3_4 = .;
_3_5 = .;
_3_6 = .;
_3_7 = .;
_3_8 = .;
_3_9 = .;
_3_10 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_5_0 = 0.0;
```

```
_5_1 = 0.0;
_5_2 = 0.0;
_5_3 = 0.0;
_5_4 = 0.0;
_5_5 = 0.0;
_5_6 = 0.0;
_5_7 = 0.0;
_5_8 = 0.0;
_5_9 = 0.0;
_5_10 = 0.0;
_5_11 = 0.0;
_5_12 = 0.0;
_5_13 = 0.0;
_5_14 = 0.0;
if MISSING(AOV16_SRV_CNT) then do ;
_5_0 = .;
_5_1 = .;
_5_2 = .;
_5_3 = .;
_5_4 = .;
_5_5 = .;
_5_6 = .;
_5_7 = .;
_5_8 = .;
_5_9 = .;
_5_10 = .;
_5_11 = .;
_5_12 = .;
_5_13 = .;
_5_14 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_SRV_CNT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_5_0 = 1.0;
end;
 else if _DM12 = '16' then do ;
_5_0 = -1.0;
_5_1 = -1.0;
_5_2 = -1.0;
_5_3 = -1.0;
_5_4 = -1.0;
_5_5 = -1.0;
_5_6 = -1.0;
_5_7 = -1.0;
_5_8 = -1.0;
_5_9 = -1.0;
_5_10 = -1.0;
_5_11 = -1.0;
_5_12 = -1.0;
_5_13 = -1.0;
_5_14 = -1.0;
end;
```

```
 else if _DM12 = '2' then do ;
_5_1 = 1.0;
end;
 else if _DM12 = '15' then do ;
_5_14 = 1.0;
end;
 else if _DM12 = '14' then do ;
_5_13 = 1.0;
end;
 else if _DM12 = '3' then do ;
_5_2 = 1.0;
end;
 else if _DM12 = '4' then do ;
_5_3 = 1.0;
end;
 else if _DM12 = '5' then do ;
_5_4 = 1.0;
end;
 else if _DM12 = '6' then do ;
_5_5 = 1.0;
end;
 else if _DM12 = '8' then do ;
_5_7 = 1.0;
end;
 else if _DM12 = '7' then do ;
_5_6 = 1.0;
end;
 else if _DM12 = '9' then do ;
_5_8 = 1.0;
end;
 else if _DM12 = '10' then do ;
_5_9 = 1.0;
end;
 else if _DM12 = '12' then do ;
_5_11 = 1.0;
end;
 else if _DM12 = '11' then do ;
_5_10 = 1.0;
end;
 else if _DM12 = '13' then do ;
_5_12 = 1.0;
end;
 else do ;
_5_0 = .;
_5_1 = .;
_5_2 = .;
_5_3 = .;
_5_4 = .;
_5_5 = .;
_5_6 = .;
_5_7 = .;
_5_8 = .;
_5_9 = .;
_5_10 = .;
_5_11 = .;
_5_12 = .;
```

```
_5_13 = .;
_5_14 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if MISSING(G_FLAG) then do ;
_6_0 = .;
_6_1 = .;
_6_2 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(G_FLAG, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_6_0 = 0.0;
_6_1 = 1.0;
_6_2 = 0.0;
end;
 else if _DM12 = '0' then do ;
_6_0 = 1.0;
_6_1 = 0.0;
_6_2 = 0.0;
end;
 else if _DM12 = '2' then do ;
_6_0 = 0.0;
_6_1 = 0.0;
_6_2 = 1.0;
end;
 else if _DM12 = '3' then do ;
_6_0 = -1.0;
_6_1 = -1.0;
_6_2 = -1.0;
end;
 else do ;
_6_0 = .;
_6_1 = .;
_6_2 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if MISSING(G_SERVICE) then do ;
_7_0 = .;
_7_1 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(G_SERVICE, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '2' then do ;
_7_0 = -1.0;
_7_1 = -1.0;
```

```
end;
 else if _DM12 = '0' then do ;
_7_0 = 1.0;
_7_1 = 0.0;
end;
 else if _DM12 = '1' then do ;
_7_0 = 0.0;
_7_1 = 1.0;
end;
 else do ;
_7_0 = .;
_7_1 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if _DM_BAD > 0.0 then do ;
_P0 = 0.0006798097;
_P1 = 0.0153183775;
_P2 = 0.0558123725;
_P3 = 0.3941083163;
_P4 = 0.534081124;
goto REGDR1;
end;
_LP0 = 0.0;
_LP1 = 0.0;
_LP2 = 0.0;
_LP3 = 0.0;
_TEMP = 1.0;
_LP0 = _LP0 + (8.97309749884509) * _TEMP * _1_0;
_LP1 = _LP1 + (9.475456450304) * _TEMP * _1_0;
_LP2 = _LP2 + (0.08183779939133) * _TEMP * _1_0;
_LP3 = _LP3 + (7.91547642280949) * _TEMP * _1_0;
_LP0 = _LP0 + (-7.09311218652648) * _TEMP * _1_1;
_LP1 = _LP1 + (3.42946756538907) * _TEMP * _1_1;
_LP2 = _LP2 + (-1.63736222687037) * _TEMP * _1_1;
_LP3 = _LP3 + (8.60035492871607) * _TEMP * _1_1;
_LP0 = _LP0 + (16.3315840253036) * _TEMP * _1_2;
_LP1 = _LP1 + (-5.85959164693143) * _TEMP * _1_2;
_LP2 = _LP2 + (-2.53740928241609) * _TEMP * _1_2;
_LP3 = _LP3 + (2.62120809028614) * _TEMP * _1_2;
_LP0 = _LP0 + (-22.5615273556858) * _TEMP * _1_3;
_LP1 = _LP1 + (-5.52330111707437) * _TEMP * _1_3;
_LP2 = _LP2 + (-5.33919133360776) * _TEMP * _1_3;
_LP3 = _LP3 + (0.11884727866076) * _TEMP * _1_3;
_LP0 = _LP0 + (-30.2554906468364) * _TEMP * _1_4;
_LP1 = _LP1 + (0.64526397467362) * _TEMP * _1_4;
_LP2 = _LP2 + (-4.40987507627988) * _TEMP * _1_4;
_LP3 = _LP3 + (-1.46254346452609) * _TEMP * _1_4;
_LP0 = _LP0 + (13.4444067104834) * _TEMP * _1_5;
_LP1 = _LP1 + (-15.4359581659106) * _TEMP * _1_5;
_LP2 = _LP2 + (-3.78315830765155) * _TEMP * _1_5;
_LP3 = _LP3 + (-4.74730533646477) * _TEMP * _1_5;
_LP0 = _LP0 + (5.99426137980241) * _TEMP * _1_6;
_LP1 = _LP1 + (3.34304711000097) * _TEMP * _1_6;
_LP2 = _LP2 + (-4.49993737709991) * _TEMP * _1_6;
```

```
_LP3 = _LP3 + (0.39149662840319) * _TEMP * _1_6;
_LP0 = _LP0 + (8.00404660871621) * _TEMP * _1_7;
_LP1 = _LP1 + (3.87729351931859) * _TEMP * _1_7;
_LP2 = _LP2 + (-5.662863418933) * _TEMP * _1_7;
_LP3 = _LP3 + (0.92431512613497) * _TEMP * _1_7;
_LP0 = _LP0 + (9.73639514490121) * _TEMP * _1_8;
_LP1 = _LP1 + (1.66486268124882) * _TEMP * _1_8;
_LP2 = _LP2 + (-5.34790399310294) * _TEMP * _1_8;
_LP3 = _LP3 + (-0.80452936208339) * _TEMP * _1_8;
_LP0 = _LP0 + (-1.18886754533908) * _TEMP * _1_9;
_LP1 = _LP1 + (0.98108751722337) * _TEMP * _1_9;
_LP2 = _LP2 + (-5.09756573837529) * _TEMP * _1_9;
_LP3 = _LP3 + (0.55035390990751) * _TEMP * _1_9;
_LP0 = _LP0 + (3.33003316374041) * _TEMP * _1_10;
_LP1 = _LP1 + (1.28863079547562) * _TEMP * _1_10;
_LP2 = _LP2 + (4.52005620947533) * _TEMP * _1_10;
_LP3 = _LP3 + (-1.88185495205653) * _TEMP * _1_10;
_LP0 = _LP0 + (-1.23514061750629) * _TEMP * _1_11;
_LP1 = _LP1 + (-0.63165164315095) * _TEMP * _1_11;
_LP2 = _LP2 + (4.47980876228159) * _TEMP * _1_11;
_LP3 = _LP3 + (-2.28762571372038) * _TEMP * _1_11;
_LP0 = _LP0 + (3.45175998109795) * _TEMP * _1_12;
_LP1 = _LP1 + (-0.05911640263949) * _TEMP * _1_12;
_LP2 = _LP2 + (3.7133976012504) * _TEMP * _1_12;
_LP3 = _LP3 + (-3.40533163917284) * _TEMP * _1_12;
_LP0 = _LP0 + (-1.79579379752335) * _TEMP * _1_13;
_LP1 = _LP1 + (-0.66575638518718) * _TEMP * _1_13;
_LP2 = _LP2 + (2.46190197688312) * _TEMP * _1_13;
_LP3 = _LP3 + (-3.86144993561858) * _TEMP * _1_13;
_LP0 = _LP0 + (16.2289623285747) * _TEMP * _1_14;
_LP1 = _LP1 + (3.87844062530087) * _TEMP * _1_14;
_LP2 = _LP2 + (13.5342255495752) * _TEMP * _1_14;
_LP3 = _LP3 + (0.11787447033604) * _TEMP * _1_14;
_TEMP = 1.0;
_LP0 = _LP0 + (4.96178727582277) * _TEMP * _2_0;
_LP1 = _LP1 + (7.19423934264755) * _TEMP * _2_0;
_LP2 = _LP2 + (-2.57814751000107) * _TEMP * _2_0;
_LP3 = _LP3 + (-0.41318251862093) * _TEMP * _2_0;
_LP0 = _LP0 + (2.53606187215301) * _TEMP * _2_1;
_LP1 = _LP1 + (1.02456723195019) * _TEMP * _2_1;
_LP2 = _LP2 + (-3.01518942636817) * _TEMP * _2_1;
_LP3 = _LP3 + (-6.42999803474578) * _TEMP * _2_1;
_LP0 = _LP0 + (-17.0716556901489) * _TEMP * _2_2;
_LP1 = _LP1 + (-2.55836176487159) * _TEMP * _2_2;
_LP2 = _LP2 + (-2.66986765613004) * _TEMP * _2_2;
_LP3 = _LP3 + (-3.77427590976266) * _TEMP * _2_2;
_LP0 = _LP0 + (-11.6228431594003) * _TEMP * _2_3;
_LP1 = _LP1 + (-4.42118648129498) * _TEMP * _2_3;
_LP2 = _LP2 + (-2.41006554535669) * _TEMP * _2_3;
_LP3 = _LP3 + (-2.47998713977501) * _TEMP * _2_3;
_LP0 = _LP0 + (-6.65446334079067) * _TEMP * _2_4;
_LP1 = _LP1 + (-4.21089586391698) * _TEMP * _2_4;
_LP2 = _LP2 + (-2.40850931862971) * _TEMP * _2_4;
_LP3 = _LP3 + (-2.46504190674716) * _TEMP * _2_4;
_LP0 = _LP0 + (-3.17136687316047) * _TEMP * _2_5;
_LP1 = _LP1 + (-1.69998112881134) * _TEMP * _2_5;
```

```
_LP2 = _LP2 + (-2.27711189809608) * _TEMP * _2_5;
_LP3 = _LP3 + (-0.56541361679043) * _TEMP * _2_5;
_LP0 = _LP0 + (0.06485838750697) * _TEMP * _2_6;
_LP1 = _LP1 + (2.083825423476) * _TEMP * _2_6;
_LP2 = _LP2 + (4.31755671819224) * _TEMP * _2_6;
_LP3 = _LP3 + (4.36618153369848) * _TEMP * _2_6;
_LP0 = _LP0 + (-3.15969642288067) * _TEMP * _2_7;
_LP1 = _LP1 + (-3.11663563731206) * _TEMP * _2_7;
_LP2 = _LP2 + (-1.93101189518423) * _TEMP * _2_7;
_LP3 = _LP3 + (-0.66813727595772) * _TEMP * _2_7;
_LP0 = _LP0 + (23.3492198306386) * _TEMP * _2_8;
_LP1 = _LP1 + (15.3692429684277) * _TEMP * _2_8;
_LP2 = _LP2 + (19.6299281653522) * _TEMP * _2_8;
_LP3 = _LP3 + (3.7767067535256) * _TEMP * _2_8;
_LP0 = _LP0 + (0.6888846491937) * _TEMP * _2_9;
_LP1 = _LP1 + (0.26881516596812) * _TEMP * _2_9;
_LP2 = _LP2 + (3.49366097063402) * _TEMP * _2_9;
_LP3 = _LP3 + (4.77521196924485) * _TEMP * _2_9;
_LP0 = _LP0 + (6.93832447370645) * _TEMP * _2_10;
_LP1 = _LP1 + (-14.7995817386477) * _TEMP * _2_10;
_LP2 = _LP2 + (-3.17802481741923) * _TEMP * _2_10;
_LP3 = _LP3 + (-0.75953335528334) * _TEMP * _2_10;
_LP0 = _LP0 + (-5.17421740905568) * _TEMP * _2_11;
_LP1 = _LP1 + (-3.50927803184578) * _TEMP * _2_11;
_LP2 = _LP2 + (-0.93991965967767) * _TEMP * _2_11;
_LP3 = _LP3 + (-0.57578867183536) * _TEMP * _2_11;
_LP0 = _LP0 + (-5.40485675039647) * _TEMP * _2_12;
_LP1 = _LP1 + (-3.43007109867235) * _TEMP * _2_12;
_LP2 = _LP2 + (-11.8686117799293) * _TEMP * _2_12;
_LP3 = _LP3 + (-0.57409656273319) * _TEMP * _2_12;
_TEMP = 1.0;
_LP0 = _LP0 + (42.0263556916437) * _TEMP * _3_0;
_LP1 = _LP1 + (1.55172177304255) * _TEMP * _3_0;
_LP2 = _LP2 + (10.9123737543277) * _TEMP * _3_0;
_LP3 = _LP3 + (2.20643367366059) * _TEMP * _3_0;
_LP0 = _LP0 + (35.8542164366111) * _TEMP * _3_1;
_LP1 = _LP1 + (-7.03832251333459) * _TEMP * _3_1;
_LP2 = _LP2 + (-13.5692536842049) * _TEMP * _3_1;
_LP3 = _LP3 + (-11.6512021486838) * _TEMP * _3_1;
_LP0 = _LP0 + (47.2300160154457) * _TEMP * _3_2;
_LP1 = _LP1 + (5.43079775823532) * _TEMP * _3_2;
_LP2 = _LP2 + (-1.76238042211005) * _TEMP * _3_2;
_LP3 = _LP3 + (2.88051687962657) * _TEMP * _3_2;
_LP0 = _LP0 + (32.2801944616028) * _TEMP * _3_3;
_LP1 = _LP1 + (5.10935792540826) * _TEMP * _3_3;
_LP2 = _LP2 + (2.52744460733309) * _TEMP * _3_3;
_LP3 = _LP3 + (2.95088205442946) * _TEMP * _3_3;
_LP0 = _LP0 + (31.6597113950015) * _TEMP * _3_4;
_LP1 = _LP1 + (-9.07258866978128) * _TEMP * _3_4;
_LP2 = _LP2 + (1.62190948241675) * _TEMP * _3_4;
_LP3 = _LP3 + (2.04551962977074) * _TEMP * _3_4;
_LP0 = _LP0 + (31.6597116255105) * _TEMP * _3_5;
_LP1 = _LP1 + (2.67824698076013) * _TEMP * _3_5;
_LP2 = _LP2 + (1.62190948383178) * _TEMP * _3_5;
_LP3 = _LP3 + (1.19293530007666) * _TEMP * _3_5;
_LP0 = _LP0 + (31.6597116340262) * _TEMP * _3_6;
```

```
_LP1 = _LP1 + (2.54298742758522) * _TEMP * _3_6;
_LP2 = _LP2 + (1.62190948388826) * _TEMP * _3_6;
_LP3 = _LP3 + (1.30825513024406) * _TEMP * _3_6;
_LP0 = _LP0 + (-362.950916427088) * _TEMP * _3_7;
_LP1 = _LP1 + (6.17176825281735) * _TEMP * _3_7;
_LP2 = _LP2 + (2.29729057331607) * _TEMP * _3_7;
_LP3 = _LP3 + (1.72970564346861) * _TEMP * _3_7;
_LP0 = _LP0 + (15.9700734501859) * _TEMP * _3_8;
_LP1 = _LP1 + (-9.54799929498259) * _TEMP * _3_8;
_LP2 = _LP2 + (-9.74287510861865) * _TEMP * _3_8;
_LP3 = _LP3 + (1.95662231341111) * _TEMP * _3_8;
_LP0 = _LP0 + (31.6597115840211) * _TEMP * _3_9;
_LP1 = _LP1 + (-9.0725886711641) * _TEMP * _3_9;
_LP2 = _LP2 + (1.62190948358845) * _TEMP * _3_9;
_LP3 = _LP3 + (2.04551963042141) * _TEMP * _3_9;
_LP0 = _LP0 + (31.291502511214) * _TEMP * _3_10;
_LP1 = _LP1 + (20.319207702854) * _TEMP * _3_10;
_LP2 = _LP2 + (1.22785286240863) * _TEMP * _3_10;
_LP3 = _LP3 + (-8.71070773697709) * _TEMP * _3_10;
_TEMP = 1.0;
_LP0 = _LP0 + (39.0432493014866) * _TEMP * _5_0;
_LP1 = _LP1 + (2.41556930669061) * _TEMP * _5_0;
_LP2 = _LP2 + (10.9819053439207) * _TEMP * _5_0;
_LP3 = _LP3 + (-2.4193090445841) * _TEMP * _5_0;
_LP0 = _LP0 + (26.0525989318919) * _TEMP * _5_1;
_LP1 = _LP1 + (-10.8013995852177) * _TEMP * _5_1;
_LP2 = _LP2 + (7.80802468659326) * _TEMP * _5_1;
_LP3 = _LP3 + (-8.37335359162762) * _TEMP * _5_1;
_LP0 = _LP0 + (-91.7996367657177) * _TEMP * _5_2;
_LP1 = _LP1 + (-7.20941847531768) * _TEMP * _5_2;
_LP2 = _LP2 + (6.37205506985912) * _TEMP * _5_2;
_LP3 = _LP3 + (-4.13523264892108) * _TEMP * _5_2;
_LP0 = _LP0 + (-43.2987854849329) * _TEMP * _5_3;
_LP1 = _LP1 + (9.63628678654799) * _TEMP * _5_3;
_LP2 = _LP2 + (15.2260866612625) * _TEMP * _5_3;
_LP3 = _LP3 + (3.41098536758909) * _TEMP * _5_3;
_LP0 = _LP0 + (-92.5078418147566) * _TEMP * _5_4;
_LP1 = _LP1 + (0.92035946274589) * _TEMP * _5_4;
_LP2 = _LP2 + (14.6028124613418) * _TEMP * _5_4;
_LP3 = _LP3 + (4.74556696940043) * _TEMP * _5_4;
_LP0 = _LP0 + (-169.198537792928) * _TEMP * _5_5;
_LP1 = _LP1 + (17.5135430652249) * _TEMP * _5_5;
_LP2 = _LP2 + (-27.5413368656283) * _TEMP * _5_5;
_LP3 = _LP3 + (5.71011491340335) * _TEMP * _5_5;
_LP0 = _LP0 + (29.0429678675398) * _TEMP * _5_6;
_LP1 = _LP1 + (-4.70698581451379) * _TEMP * _5_6;
_LP2 = _LP2 + (2.19747568966552) * _TEMP * _5_6;
_LP3 = _LP3 + (0.25036394861618) * _TEMP * _5_6;
_LP0 = _LP0 + (27.4220001532713) * _TEMP * _5_7;
_LP1 = _LP1 + (-5.62951270960282) * _TEMP * _5_7;
_LP2 = _LP2 + (2.97946845585617) * _TEMP * _5_7;
_LP3 = _LP3 + (0.07300025078033) * _TEMP * _5_7;
_LP0 = _LP0 + (24.9838671156593) * _TEMP * _5_8;
_LP1 = _LP1 + (-4.23916148505361) * _TEMP * _5_8;
_LP2 = _LP2 + (3.42557523365742) * _TEMP * _5_8;
_LP3 = _LP3 + (1.46388562797025) * _TEMP * _5_8;
```

```
_LP0 = _LP0 + (22.8194752422965) * _TEMP * _5_9;
_LP1 = _LP1 + (-4.25224375283395) * _TEMP * _5_9;
_LP2 = _LP2 + (2.49905210556025) * _TEMP * _5_9;
_LP3 = _LP3 + (-0.01709833699071) * _TEMP * _5_9;
_LP0 = _LP0 + (37.114213383863) * _TEMP * _5_10;
_LP1 = _LP1 + (2.9953971574379) * _TEMP * _5_10;
_LP2 = _LP2 + (-4.63754693643679) * _TEMP * _5_10;
_LP3 = _LP3 + (-4.36468726526216) * _TEMP * _5_10;
_LP0 = _LP0 + (34.2320056651284) * _TEMP * _5_11;
_LP1 = _LP1 + (-2.48152127510367) * _TEMP * _5_11;
_LP2 = _LP2 + (-7.20881969172312) * _TEMP * _5_11;
_LP3 = _LP3 + (2.05199646600986) * _TEMP * _5_11;
_LP0 = _LP0 + (34.1979425371632) * _TEMP * _5_12;
_LP1 = _LP1 + (1.32583179116639) * _TEMP * _5_12;
_LP2 = _LP2 + (-1.94011877303868) * _TEMP * _5_12;
_LP3 = _LP3 + (8.74058490108554) * _TEMP * _5_12;
_LP0 = _LP0 + (39.1512435469843) * _TEMP * _5_13;
_LP1 = _LP1 + (1.88577792759584) * _TEMP * _5_13;
_LP2 = _LP2 + (-1.93386166738385) * _TEMP * _5_13;
_LP3 = _LP3 + (0.84886002004651) * _TEMP * _5_13;
_LP0 = _LP0 + (20.9363766085136) * _TEMP * _5_14;
_LP1 = _LP1 + (-2.0647251475618) * _TEMP * _5_14;
_LP2 = _LP2 + (-13.1892422255085) * _TEMP * _5_14;
_LP3 = _LP3 + (-4.52842188369726) * _TEMP * _5_14;
_TEMP = 1.0;
_LP0 = _LP0 + (1.76663561037174) * _TEMP * _6_0;
_LP1 = _LP1 + (-5.40874215787948) * _TEMP * _6_0;
_LP2 = _LP2 + (-6.87281360284862) * _TEMP * _6_0;
_LP3 = _LP3 + (-6.22229997982126) * _TEMP * _6_0;
_LP0 = _LP0 + (21.8797726373068) * _TEMP * _6_1;
_LP1 = _LP1 + (2.87906958740983) * _TEMP * _6_1;
_LP2 = _LP2 + (1.83666665646742) * _TEMP * _6_1;
_LP3 = _LP3 + (4.13135987011355) * _TEMP * _6_1;
_LP0 = _LP0 + (1.73459041116589) * _TEMP * _6_2;
_LP1 = _LP1 + (-0.75352434519744) * _TEMP * _6_2;
_LP2 = _LP2 + (-0.62400019216188) * _TEMP * _6_2;
_LP3 = _LP3 + (0.53569098310408) * _TEMP * _6_2;
_TEMP = 1.0;
_LP0 = _LP0 + (-3.44927846183227) * _TEMP * _7_0;
_LP1 = _LP1 + (-6.37652016665453) * _TEMP * _7_0;
_LP2 = _LP2 + (-4.25904939215537) * _TEMP * _7_0;
_LP3 = _LP3 + (-4.51685639332432) * _TEMP * _7_0;
_LP0 = _LP0 + (-6.43408008433648) * _TEMP * _7_1;
_LP1 = _LP1 + (-0.80236520705753) * _TEMP * _7_1;
_LP2 = _LP2 + (-0.12922463272966) * _TEMP * _7_1;
_LP3 = _LP3 + (-0.63228249961139) * _TEMP * _7_1;
_LPMAX = 0.0;
_LP0 = -123.067467124716 + _LP0;
if _LPMAX < _LP0 then _LPMAX = _LP0;
_LP1 = -23.6221258810818 + _LP1;
if _LPMAX < _LP1 then _LPMAX = _LP1;
_LP2 = -18.5909979689337 + _LP2;
if _LPMAX < _LP2 then _LPMAX = _LP2;
_LP3 = -6.00322742797283 + _LP3;
if _LPMAX < _LP3 then _LPMAX = _LP3;
_LP0 = EXP(_LP0 - _LPMAX);
```

```
_LP1 = EXP(_LP1 - _LPMAX);
_LP2 = EXP(_LP2 - _LPMAX);
_LP3 = EXP(_LP3 - _LPMAX);
_LPMAX = EXP(-_LPMAX);
_P4 = 1.0 / (_LPMAX + _LP0 + _LP1 + _LP2 + _LP3);
_P0 = _LP0 * _P4;
_P1 = _LP1 * _P4;
_P2 = _LP2 * _P4;
_P3 = _LP3 * _P4;
_P4 = _LPMAX * _P4;
REGDR1: P_ATTACKU2R = _P0;
_MAXP = _P0;
_IY = 1.0;
P_ATTACKR2L = _P1;
if (_P1 - _MAXP > 1E-8) then do ;
_MAXP = _P1;
_IY = 2.0;
end;
P_ATTACKPROBE = _P2;
if (_P2 - _MAXP > 1E-8) then do ;
_MAXP = _P2;
_IY = 3.0;
end;
P_ATTACKNORMAL = _P3;
if (_P3 - _MAXP > 1E-8) then do ;
_MAXP = _P3;
_IY = 4.0;
end;
P_ATTACKDOS = _P4;
if (_P4 - _MAXP > 1E-8) then do ;
_MAXP = _P4;
_IY = 5.0;
end;
I_ATTACK = REGDRF[_IY];
U_ATTACK = REGDRU[_IY];
EM_EVENTPROBABILITY = P_ATTACKU2R;
EM_PROBABILITY = MAX(P_ATTACKU2R, P_ATTACKR2L, P_ATTACKPROBE, P_ATTACKNORMAL,
P_ATTACKDOS);
EM_CLASSIFICATION = I_ATTACK;
_return: ;
end;
 enddata;
```

Example of an Input and Output Variables Scoring File

Here is an example of an input and output variables scoring file. The filename is sasscore_score_io.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<Score>
   <Producer>
      <Name> SAS Enterprise Miner </Name>
```

```
      <Version> 1.0 </Version>
   </Producer>
   <TargetList>
   </TargetList>
   <Input>
      <Variable>
         <Name> COUNT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> DIF_SRVR </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[diff_srv_rate]]>
         </Description>
      </Variable>
      <Variable>
         <Name> FLAG </Name>
         <Type> character </Type>
      </Variable>
      <Variable>
         <Name> HOT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> SAM_SRAT </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[same_srv_rate]]>
         </Description>
      </Variable>
      <Variable>
         <Name> SERVICE </Name>
         <Type> character </Type>
      </Variable>
      <Variable>
         <Name> SRV_CNT </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[srv_count]]>
         </Description>
      </Variable>
   </Input>
   <Output>
      <Variable>
         <Name> AOV16_COUNT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_DIF_SRVR </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_HOT </Name>
         <Type> numeric </Type>
      </Variable>
```

```
<Variable>
   <Name> AOV16_SAM_SRAT </Name>
   <Type> numeric </Type>
</Variable>
<Variable>
   <Name> AOV16_SRV_CNT </Name>
   <Type> numeric </Type>
</Variable>
<Variable>
   <Name> EM_CLASSIFICATION </Name>
   <Type> character </Type>
   <Description>
      <![CDATA[Prediction for ATTACK]]>
   </Description>
</Variable>
<Variable>
   <Name> EM_EVENTPROBABILITY </Name>
   <Type> numeric </Type>
   <Description>
      <![CDATA[Probability for level U2R of ATTACK]]>
   </Description>
</Variable>
<Variable>
   <Name> EM_PROBABILITY </Name>
   <Type> numeric </Type>
   <Description>
      <![CDATA[Probability of Classification]]>
   </Description>
</Variable>
<Variable>
   <Name> G_FLAG </Name>
   <Type> numeric </Type>
</Variable>
<Variable>
   <Name> G_SERVICE </Name>
   <Type> numeric </Type>
</Variable>
<Variable>
   <Name> I_ATTACK </Name>
   <Type> character </Type>
   <Description>
      <![CDATA[Into: ATTACK]]>
   </Description>
</Variable>
<Variable>
   <Name> P_ATTACKDOS </Name>
   <Type> numeric </Type>
   <Description>
      <![CDATA[Predicted: ATTACK=dos]]>
   </Description>
</Variable>
<Variable>
   <Name> P_ATTACKNORMAL </Name>
   <Type> numeric </Type>
   <Description>
      <![CDATA[Predicted: ATTACK=normal]]>
```

```
      </Description>
    </Variable>
    <Variable>
      <Name> P_ATTACKPROBE </Name>
      <Type> numeric </Type>
      <Description>
        <![CDATA[Predicted: ATTACK=probe]]>
      </Description>
    </Variable>
    <Variable>
      <Name> P_ATTACKR2L </Name>
      <Type> numeric </Type>
      <Description>
        <![CDATA[Predicted: ATTACK=r2l]]>
      </Description>
    </Variable>
    <Variable>
      <Name> P_ATTACKU2R </Name>
      <Type> numeric </Type>
      <Description>
        <![CDATA[Predicted: ATTACK=u2r]]>
      </Description>
    </Variable>
    <Variable>
      <Name> U_ATTACK </Name>
      <Type> character </Type>
      <Description>
        <![CDATA[Unnormalized Into: ATTACK]]>
      </Description>
    </Variable>
    <Variable>
      <Name> _WARN_ </Name>
      <Type> character </Type>
      <Description>
        <![CDATA[Warnings]]>
      </Description>
    </Variable>
  </Output>
  <C>
    <Function>
      <Name>
        score
      </Name>
      <ParameterList>
        <Parameter>
          <Array length="7">
            <Type>
              Parm
            </Type>
            <DataMap>
              <Element index="0">
                <Value>
                  <Origin> COUNT </Origin>
                  <Type> double </Type>
                </Value>
              </Element>
```

```
            <Element index="1">
               <Value>
                  <Origin> DIF_SRVR </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="2">
               <Value>
                  <Origin> FLAG </Origin>
                  <Array length="33">
                    <Type> char </Type>
                  </Array>
               </Value>
            </Element>
            <Element index="3">
               <Value>
                  <Origin> HOT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="4">
               <Value>
                  <Origin> SAM_SRAT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="5">
               <Value>
                  <Origin> SERVICE </Origin>
                  <Array length="33">
                    <Type> char </Type>
                  </Array>
               </Value>
            </Element>
            <Element index="6">
               <Value>
                  <Origin> SRV_CNT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
         </DataMap>
      </Array>
   </Parameter>

   <Parameter>
      <Array length="18">
         <Type>
            Parm
         </Type>
         <DataMap>
            <Element index="0">
               <Value>
                  <Origin> AOV16_COUNT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
```

```
<Element index="1">
   <Value>
      <Origin> AOV16_DIF_SRVR </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="2">
   <Value>
      <Origin> AOV16_HOT </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="3">
   <Value>
      <Origin> AOV16_SAM_SRAT </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="4">
   <Value>
      <Origin> AOV16_SRV_CNT </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="5">
   <Value>
      <Origin> EM_CLASSIFICATION </Origin>
      <Array length="33">
        <Type> char </Type>
     </Array>
   </Value>
</Element>
<Element index="6">
   <Value>
      <Origin> EM_EVENTPROBABILITY </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="7">
   <Value>
      <Origin> EM_PROBABILITY </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="8">
   <Value>
      <Origin> G_FLAG </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="9">
   <Value>
      <Origin> G_SERVICE </Origin>
      <Type> double </Type>
   </Value>
</Element>
```

```
      <Element index="10">
         <Value>
            <Origin> I_ATTACK </Origin>
            <Array length="7">
              <Type> char </Type>
           </Array>
         </Value>
      </Element>
      <Element index="11">
         <Value>
            <Origin> P_ATTACKDOS </Origin>
            <Type> double </Type>
         </Value>
      </Element>
      <Element index="12">
         <Value>
            <Origin> P_ATTACKNORMAL </Origin>
            <Type> double </Type>
         </Value>
      </Element>
      <Element index="13">
         <Value>
            <Origin> P_ATTACKPROBE </Origin>
            <Type> double </Type>
         </Value>
      </Element>
      <Element index="14">
         <Value>
            <Origin> P_ATTACKR2L </Origin>
            <Type> double </Type>
         </Value>
      </Element>
      <Element index="15">
         <Value>
            <Origin> P_ATTACKU2R </Origin>
            <Type> double </Type>
         </Value>
      </Element>
      <Element index="16">
         <Value>
            <Origin> U_ATTACK </Origin>
            <Array length="7">
              <Type> char </Type>
           </Array>
         </Value>
      </Element>
      <Element index="17">
         <Value>
            <Origin> _WARN_ </Origin>
            <Array length="5">
              <Type> char </Type>
           </Array>
         </Value>
      </Element>
   </DataMap>
</Array>
```

```
            </Parameter>
         </ParameterList>
      </Function>
   </C>
</Score>
```

Example of a User-Defined Formats Scoring File

Here is an example of a user-defined formats scoring file. The filename is sasscore_score_ufmt.xml.

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="SUVformats.xsl"?>
<LIBRARY type="EXPORT" version="SUV">
   <HEADER>
      <Provider>SAS Institute Inc.</Provider>
      <Version>9.2</Version>
      <VersionLong>9.02.02M2D09012009</VersionLong>
      <CreationDateTime>2009-12-14T12:47:03</CreationDateTime>
   </HEADER>

   <TABLE name="sasscore_score_ufmt">
      <TABLE-HEADER>
         <Provider>SAS Institute Inc.</Provider>
         <Version>9.2</Version>
         <VersionLong>9.02.02M2D09012009</VersionLong>
         <CreationDateTime>2009-12-14T12:47:03</CreationDateTime>
         <ModifiedDateTime>2009-12-14T12:47:03</ModifiedDateTime>

         <Protection />
         <DataSetType />
         <DataRepresentation />
         <Encoding>utf-8</Encoding>
         <ReleaseCreated />
         <HostCreated />
         <FileName>sasscore_score_ufmt</FileName>

         <Observations />
         <Compression number="1" />
         <Variables number="21" />
      </TABLE-HEADER>

      <COLUMN name="FMTNAME" label="Format name">
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>32</LENGTH>
         <Offset>32</Offset>
         <SortedBy />
      </COLUMN>

      <COLUMN name="START" label="Starting value for format">
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
```

```
   <LENGTH>16</LENGTH>
   <Offset>16</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="END" label="Ending value for format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>16</LENGTH>
   <Offset>16</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="LABEL" label="Format value label">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MIN" label="Minimum length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MAX" label="Maximum length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DEFAULT" label="Default length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="LENGTH" label="Format length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="FUZZ" label="Fuzz value">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
```

```
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="PREFIX" label="Prefix characters">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>2</LENGTH>
   <Offset>2</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MULT" label="Multiplier">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="FILL" label="Fill character">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="NOEDIT" label="Is picture string noedit?">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="TYPE" label="Type of format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="SEXCL" label="Start exclusion">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="EEXCL" label="End exclusion">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
```

```
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="HLO" label="Additional information">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>11</LENGTH>
   <Offset>11</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DECSEP" label="Decimal separator">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DIG3SEP" label="Three-digit separator">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DATATYPE" label="Date/time/datetime?">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="LANGUAGE" label="Language for date strings">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<ROW>
   <FMTNAME missing=" " />
   <START missing=" " />
   <END missing=" " />
   <LABEL missing=" " />
   <MIN missing=" " />
   <MAX missing=" " />
   <DEFAULT missing=" " />
   <LENGTH missing=" " />
   <FUZZ missing=" " />
   <PREFIX missing=" " />
```

```
   <MULT missing=" " />
   <FILL missing=" " />
   <NOEDIT missing=" " />
   <TYPE missing=" " />
   <SEXCL missing=" " />
   <EEXCL missing=" " />
   <HLO missing=" " />
   <DECSEP missing=" " />
   <DIG3SEP missing=" " />
   <DATATYPE missing=" " />
   <LANGUAGE missing=" " />
</ROW>

<ROW>
   <DELTA-RECORD key="ABC" />
   <FMTNAME>ABC</FMTNAME>
   <START>1</START>
   <END>1</END>
   <LABEL>yes</LABEL>
   <MIN>1</MIN>
   <MAX>40</MAX>
   <DEFAULT>3</DEFAULT>
   <LENGTH>3</LENGTH>
   <FUZZ>1E-12</FUZZ>
   <PREFIX missing=" " />
   <MULT>0</MULT>
   <FILL missing=" " />
   <NOEDIT>0</NOEDIT>
   <TYPE>N</TYPE>
   <SEXCL>N</SEXCL>
   <EEXCL>N</EEXCL>
   <HLO missing=" " />
   <DECSEP missing=" " />
   <DIG3SEP missing=" " />
   <DATATYPE missing=" " />
   <LANGUAGE missing=" " />
</ROW>

<ROW>
   <DELTA-RECORD key="YESNO" />
   <FMTNAME>YESNO</FMTNAME>
   <START>0</START>
   <END>0</END>
   <LABEL>NO</LABEL>
   <MIN>1</MIN>
   <MAX>40</MAX>
   <DEFAULT>3</DEFAULT>
   <LENGTH>3</LENGTH>
   <FUZZ>0</FUZZ>
   <PREFIX missing=" " />
   <MULT>0</MULT>
   <FILL missing=" " />
   <NOEDIT>0</NOEDIT>
   <TYPE>C</TYPE>
   <SEXCL>N</SEXCL>
   <EEXCL>N</EEXCL>
```

```
            <HLO missing=" " />
            <DECSEP missing=" " />
            <DIG3SEP missing=" " />
            <DATATYPE missing=" " />
            <LANGUAGE missing=" " />
         </ROW>

         <ROW>
            <FMTNAME>YESNO</FMTNAME>
            <START>1</START>
            <END>1</END>
            <LABEL>YES</LABEL>
            <MIN>1</MIN>
            <MAX>40</MAX>
            <DEFAULT>3</DEFAULT>
            <LENGTH>3</LENGTH>
            <FUZZ>0</FUZZ>
            <PREFIX missing=" " />
            <MULT>0</MULT>
            <FILL missing=" " />
            <NOEDIT>0</NOEDIT>
            <TYPE>C</TYPE>
            <SEXCL>N</SEXCL>
            <EEXCL>N</EEXCL>
            <HLO missing=" " />
            <DECSEP missing=" " />
            <DIG3SEP missing=" " />
            <DATATYPE missing=" " />
            <LANGUAGE missing=" " />
         </ROW>
      </TABLE>
   </LIBRARY>
```

Index

Special Characters

%INDAC_PUBLISH_MODEL macro
  example 30
  overview 25
  syntax 28
%INDACPM macro 26
%INDB2_PUBLISH_FORMATS macro
  example 97
  modes of operation 96
  running 92
  syntax 94
%INDB2_PUBLISH_MODEL macro 35
  example 41
  modes of operation 41
  running 36
  syntax 38
%INDB2PF macro 93
%INDB2PM macro 37
%INDGP_PUBLISH_MODEL macro 47
  example 52
  running 48
  syntax 50
%INDGPPM macro 49
%INDNZ_PUBLISH_FORMATS macro
  example 108
  modes of operation 108
  running 104
  syntax 106
%INDNZ_PUBLISH_MODEL macro 57
  example 62
  modes of operation 62
  running 58
  syntax 59
%INDNZPF macro 105
%INDNZPM macro 58
%INDTD_PUBLISH_FORMATS macro 113
  example 118
  modes of operation 117
  running 114
  syntax 115
%INDTD_PUBLISH_MODEL macro 69
  arguments 71
  example 73
  modes of operation 73
  syntax 71
%INDTDPF macro 115
%INDTDPM macro 70

A

Aster nCluster
  deployed components for in-database processing 4
  permissions 30
  SAS Embedded Process 4
  SAS System libraries 4
  Scoring Accelerator 25

B

BY-group processing
  in-database procedures and 131

C

case sensitivity 43, 54, 64, 75

D

data mining models 13
data set options
  in-database procedures and 132
data types
  SAS_PUT() function (Teradata) 118
DB2
  deployed components for in-database processing 5
  in-database procedures 129
  permissions 43, 100
  publishing SAS formats 91
  SAS formats library 5

Scoring Accelerator 35
user-defined formats 91
directory names
special characters in 86

E

EM_ output variables 25, 35, 47, 57, 69
extension nodes 20

F

fenced mode 41, 62, 96, 108
fixed variable names 19
format publishing macros
special characters in directory names 86
tips for using 88
formats
determining publish dates 88
publishing (DB2) 91
publishing (Netezza) 104
publishing (Teradata) 114
SAS formats library (DB2) 5
SAS formats library (Greenplum) 5
SAS formats library (Netezza) 6
SAS formats library (Teradata) 6
formats publishing macros
%INDB2_PUBLISH_FORMATS 92
%INDNZ_PUBLISH_FORMATS 104
%INDTD_PUBLISH_FORMATS 114
functions
See SAS_PUT() function

G

Greenplum
deployed components for in-database processing 5
permissions 53
SAS formats library 5
Scoring Accelerator 47

I

in-database procedures 127
BY-groups 131
considerations and limitations 130
controlling messaging with MSGLEVEL option 133
data set options 132
DB2 129
generating SQL for 137
items preventing in-database processing 132
LIBNAME statement 131
Netezza 129
Oracle 129
row order 131
running 128
SAS formats and 113
Teradata 129
in-database processing
deployed components for Aster nCluster 4
deployed components for DB2 5
deployed components for Greenplum 5
deployed components for Netezza 5
deployed components for Teradata 6
INDCONN macro variable
Aster nCluster 27
DB2 37, 93
Greenplum 49
Netezza 59, 105
Teradata 70, 115
INTRINSIC-CRDATE format 88

M

macros
%INDAC_PUBLISH_MODEL 28
%INDACPM 26
%INDB2_PUBLISH_FORMATS 94
%INDB2_PUBLISH_MODEL 38
%INDB2PF Macro 93
%INDB2PM 37
%INDGP_PUBLISH_MODEL 50
%INDGPPM 49
%INDNZ_PUBLISH_FORMATS 106
%INDNZ_PUBLISH_MODEL 59
%INDNZPF 105
%INDNZPM 58
%INDTD_PUBLISH_FORMATS 115
%INDTD_PUBLISH_MODEL 71
%INDTDPF 115
%INDTDPM 70
messaging
controlling with MSGLEVEL option 133
model registration
Score Code Export node compared with SAS Metadata Server 14
MSGLEVEL system option
controlling messaging with 133

N

names
of scoring functions 43, 53, 64, 75
Netezza
deployed components for in-database processing 5
in-database procedures 129

permissions 64, 112
publishing SAS formats 104
SAS formats library 6
Scoring Accelerator 57
user-defined formats 103
nodes
score code created by SAS Enterprise Miner nodes 20
user-defined 20

O
Oracle
in-database procedures 129
output, created by Score Code Export node 16
output files 17
output variables 18
EM_ 25, 35, 47, 57, 69

P
permissions
Aster nCluster 30
DB2 43, 100
Greenplum 53
Netezza 64, 112
Teradata 75, 123
procedures
See in-database procedures
process flow diagrams
using Score Code Export node in 14
properties, Score Code Export node 15
protected mode 117
publishing client 9
publishing macros
%INDAC_PUBLISH_MODEL 28
%INDB2_PUBLISH_FORMATS 92
%INDB2_PUBLISH_MODEL 38
%INDGP_PUBLISH_MODEL 50
%INDNZ_PUBLISH_FORMATS 104
%INDNZ_PUBLISH_MODEL 59
%INDTD_PUBLISH_FORMATS 114
%INDTD_PUBLISH_MODEL 71
publishing process 25, 35, 47, 57, 69
publishing SAS formats
DB2 91
determining format publish dates 88
Netezza 104
special characters in directory names 86
Teradata 114
tips 88
publishing scoring model files
running %INDB2_PUBLISH_MODEL macro 36
running %INDGP_PUBLISH_MODEL macro 48
running the %INDNZ_PUBLISH_MODEL macro 58
PUT function
in-database procedures and 113
mapping to SAS_PUT function 140
reducing, based on engine type 141

R
registering models
Score Code Export node compared with SAS Metadata Server 14
Results window 16
row order
in-database procedures and 131

S
SAS_PUT() function
data types in Teradata 118
DB2 91
explicit use of (DB2) 99
explicit use of (Netezza) 111
explicit use of (Teradata) 122
implicit use of (DB2) 97
implicit use of (Netezza) 109
implicit use of (Teradata) 120
mapping PUT function to 140
Netezza 104
Teradata 114
tips for using 88
tips for using in Teradata 123
SAS_SCORE() function
installation 4
overview 32
using 32
SAS Embedded Process 4, 25
SAS Enterprise Miner
score code created by each node 20
SAS formats
%INDB2_PUBLISH_FORMATS macro 92
%INDNZ_PUBLISH_FORMATS macro 104
%INDTD_PUBLISH_FORMATS macro 114
DB2 91
deploying 113
in-database procedures and 113
Netezza 104
SAS_PUT() function and 113
SAS formats library (DB2) 5
SAS formats library (Greenplum) 5

SAS formats library (Netezza) 6
SAS formats library (Teradata) 6
Teradata 114
SAS formats library (DB2) 5
SAS formats library (Greenplum) 5
SAS formats library (Netezza) 6
SAS formats library (Teradata) 6
SAS Metadata Server
compared with registering models with Score Code Export node 14
SAS Scoring Accelerator
components 9
SAS System libraries 4
SAS/ACCESS LIBNAME statement
in-database procedures and 131
score code
created by each node of SAS Enterprise Miner 20
Score Code Export node 9, 13
compared with registering models on SAS Metadata Server 14
files exported by 13
output created by 16
properties 15
using in process flow diagrams 14
scoring files
creating 25
example 147, 167, 174
viewing 31
scoring functions
names of 43, 53, 64, 75
scoring publishing macro and 9
using 44, 54, 65, 76
viewing 44, 54, 65, 76
scoring publishing macros
%INDAC_PUBLISH_MODEL 28
%INDB2_PUBLISH_MODEL 38
%INDGP_PUBLISH_MODEL 50
%INDNZ_PUBLISH_MODEL 59
%INDTD_PUBLISH_MODEL 71
overview 9
SFTP protocol 35
source data
generating SQL for in-database processing of 137
special characters
in directory names 86
SQL
generating for in-database processing of source data 137
SQLGENERATION= system option 137
SQLMAPPUTTO= system option 140
SQLREDUCEPUT= system option 141
SSH-2 protocol 35

T

Teradata
data types and SAS_PUT() function 118
deployed components for in-database processing 6
in-database procedures 129
permissions 75, 123
publishing SAS formats 114
SAS formats library 6
tips for using the SAS_PUT() function 123
user-defined formats 113

U

UFMT-CRDATE format 89
unfenced mode 41, 62, 96, 108
unprotected mode 117
user-defined formats
DB2 91
determining publish date 88
Netezza 103
SAS_PUT() function 83
Teradata 113
user-defined nodes 20

V

variables
EM_ output variables 35, 47, 57
fixed variable names 19
output variables 18