SAS® 9.4 In-Database Products

User's Guide

Fifth Edition

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *SAS® 9.4 In-Database Products: User's Guide, Fifth Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 In-Database Products: User's Guide, Fifth Edition

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

February 2015

Contents

PART 3 SAS In-Database Code Accelerator 173

PART 4 In-Database DATA Step Processing 189

PART 5 Format Publishing and the SAS_PUT( ) Function 195

What's New in SAS 9.4 In-Database Products

Overview

In SAS 9.4, the following new features and enhancements were added to expand the capabilities of the SAS In-Database products:

- In the February 2015 release of SAS 9.4, the following changes and additions were made:
  - The SAS In-Database Code Accelerator for Hadoop uses HCatalog to process complex, non-delimited files. This enables the SAS In-Database Code Accelerator for Hadoop to support Avro, ORC, RCFile, and Parquet file types.
  - You can now use the DBCREATE_TABLE_OPTS table option to specify the output SerDe, the output delimiter of the Hive table, the output ESCAPED BY character, and any other CREATE TABLE syntax allowed by Hive.
- In the August 2014 release of SAS 9.4, the following changes and additions were made:
  - The SAS Scoring Accelerator and in-database processing of Base SAS procedures is available for SAP HANA.
  - Running limited DATA step scoring programs in Hadoop is now production.
  - Numerous changes were made to the installation and configuration script for the SAS Embedded Process for Hadoop.
- In the April 2014 release of SAS 9.4, documentation enhancements were made in the following areas:
  - installation and configuration of the SAS Embedded Process for Hadoop
  - considerations when creating or modifying DATA step score code
  - semaphore requirements when using the SAS Embedded Process for Greenplum
- In the December 2013 release of SAS 9.4, the following changes and additions were made:
  - Limited DATA step programs can be run inside Hadoop for scoring.
  - New parameters have been added for the Hadoop INDCONN macro variable.
  - New Hadoop JAR files are now tied to the version of Apache Hadoop that you are using.
  - The SAS In-Database Code Accelerator for Teradata now can run the DS2 data program as well as the thread program inside the database.

- DS2ACCEL, a new system option, controls whether the DS2 code is executed inside the database. The default value is NONE, which prevents DS2 code from executing inside the database.
- The PROC DS2 INDB option has changed its name to DS2ACCEL. INDB is still supported. However, the default value for this option has changed from YES to NO, which prevents DS2 code from executing in the database. This is a change in behavior from the initial 9.4 release.

- In the September 2013 release of SAS 9.4, the following changes and additions were made:
  - In-database processing for Hadoop has been enhanced by the addition of the SAS Scoring Accelerator for Hadoop.
  - The autocall macros that initialized the publishing macros are no longer needed for any DBMS. However, they are still supported.
- In the July 2013 release of SAS 9.4, the following changes and additions were made:
  - The SAS Scoring Accelerator for SPD Server is available.
- In the June 2013 release of SAS 9.4, the following changes and additions were made:
  - Greenplum and Teradata in-database processing has been enhanced by the addition of the SAS In-Database Code Accelerators.
  - The SAS In-Database Code Accelerator enables you to publish a DS2 thread program to the database and execute that thread program in parallel inside the database.
  - In-database scoring for Netezza has been enhanced by the addition of the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs within Netezza to read and write data.
  - Two columns have been added to the model table. The ModelUUID and Notes columns assist in processing scoring models when using the SAS Embedded Process.
  - The Hadoop scripts that install, control, and provide status of the SAS Embedded Process have changed. There is now just one script, sasep-server.sh, that installs both the SAS Embedded Process and the Hadoop JAR files.

SAS In-Database Code Accelerator

February 2015 Release of SAS 9.4: Changes and Enhancements

In the February 2015 release of SAS 9.4, the following changes and additions were made:

- The SAS In-Database Code Accelerator for Hadoop supports only Cloudera 5.2 and Hortonworks 2.1 or later.
- The SAS In-Database Code Accelerator for Hadoop uses HCatalog to process complex, non-delimited files.
- The SAS In-Database Code Accelerator for Hadoop now supports Avro, ORC, RCFile, and Parquet file types.

- For the SAS In-Database Code Accelerator for Hadoop, you can use the DBCREATE_TABLE_OPTS table option to specify the output SerDe, the output delimiter of the Hive table, the output escaped by, and any other CREATE TABLE syntax allowed by Hive.

August 2014 Release of SAS 9.4: Changes and Enhancements

Hadoop in-database processing has been enhanced by the addition of the SAS In-Database Code Accelerator. The SAS In-Database Code Accelerator enables you to publish a DS2 thread and data program to the database and execute those programs in parallel inside the database.

December 2013 Release of SAS 9.4: Changes and Enhancements

In the December 2013 release of SAS 9.4, the following changes and additions were made:

- The SAS In-Database Code Accelerator for Teradata now runs the DS2 data program as well as the thread program inside the database.
- DS2ACCEL, a new system option, controls whether the DS2 code is executed inside the database. The default value is NONE, which prevents DS2 code from executing inside the database.
- The PROC DS2 INDB option has changed its name to DS2ACCEL. INDB is still supported. However, the default value for this option has changed from YES to NO. This change prevents DS2 code from executing in the database. This is a change in behavior from the initial SAS 9.4 release.

SAS 9.4: Changes and Enhancements

The SAS In-Database Code Accelerator enables you to publish a DS2 thread program to the database and execute that thread program in parallel inside the database. Examples of thread programs include large transpositions, computationally complex programs, scoring models, and BY-group processing. The SAS In-Database Code Accelerator is available for Greenplum and Teradata.

The SAS In-Database Code Accelerator must be licensed at your site.

Greenplum Changes

April 2014 Release of SAS 9.4: Changes and Enhancements

Information about semaphore requirements when using the SAS Embedded Process was added to *SAS In-Database Products: Administrator's Guide*.

SAS 9.4: Changes and Enhancements

There are several changes for Greenplum:

- Version 1.2 of the Greenplum Partner Connector (GPPC) is now available and should be installed if you use SAS Embedded Process 9.4.
- A new script, UninstallSASEPFiles.sh, is available. This script stops and uninstalls the SAS Embedded Process on each database host node.
- You can now specify a non-default port when you create the connection string to publish formats and models.

Hadoop Changes

August 2014 Release of SAS 9.4: Changes and Enhancements

In the August 2014 release of SAS 9.4, there are several Hadoop changes:

- You can now specify a fixed record format for the output file of the SAS Scoring Accelerator for Hadoop. Previously, all output was delimited.
- SPD file formats are supported by the SAS Embedded Process for Hadoop.
- Instead of manually selecting the Hadoop JAR files to the client machine, the SAS Embedded Process determines which version of the JAR files are required and gathers them into a ZIP file for you to copy to the client machine.
- You now have the option whether to automatically start the SAS Embedded Process when the installation is complete.

April 2014 Release of SAS 9.4: Changes and Enhancements

The documentation about the installation and configuration of the SAS Embedded Process was enhanced.

December 2013 Release of SAS 9.4: Changes and Enhancements

In the December 2013 release of SAS 9.4, there are several Hadoop changes:

- The SAS Embedded Process and the SAS Scoring Accelerator for Hadoop support Kerberos and Hive2 for both Cloudera and Hortonworks.
- The trace log messages for the SAS Embedded Process are now stored in the MapReduce job log.
- A new option, *hdfsuser*, is available in the sasep-servers.sh script. *hdfsuser* specifies the user ID that has Write access to HDFS root directory.
- A new parameter, HADOOP_CFG=, is available for the INDCONN macro variable. The HADOOP_CFG= parameter specifies the location of the Hadoop configuration file that is used with the %INDHD_PUBLISH_MODEL and the %INDHD_RUN_MODEL macros.
- The Cloudera JAR files for the SAS Embedded Process have been replaced by a set of Apache JAR files. The new JAR files are based on a release of the Apache Hadoop instead of a particular Hadoop distributor.

September 2013 Release of SAS 9.4: Changes and Enhancements

In-database scoring for Hadoop is available.

SAS 9.4: Changes and Enhancements

The Hadoop scripts that install, control, and provide status of the SAS Embedded Process have changed. There is now just one script, sasep-servers.sh, that installs both the SAS Embedded Process and the Hadoop JAR files. Running this script also enables you to start, stop, and provide status of the SAS Embedded Process.

DATA Step Processing in Hadoop

August 2014 Release of SAS 9.4: Changes and Enhancements

Running limited DATA step scoring programs in Hadoop is now production.

December 2013 Release of SAS 9.4: Changes and Enhancements

Limited DATA step scoring programs can be run inside Hadoop. This feature is pre-production.

Netezza Changes

SAS 9.4: Changes and Enhancements

In-database scoring for Netezza has been enhanced by the addition of the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs within Netezza to read and write data. The SAS Embedded Process can be used with the SAS Scoring Accelerator for Netezza to run scoring models.

In-Database Processing for SAP HANA

August 2014 Release of SAS 9.4: Changes and Enhancements

In-database scoring for SAP HANA is available. You can also run Base SAS procedures inside SAP HANA.

SAS Scoring Accelerator for SPD Server

July 2013 Release of SAS 9.4: Changes and Enhancements

In-database scoring for the SAS Scalable Performance Data Server is available.

Changes for Running In-Database Procedures

SAS 9.4: Changes and Enhancements

The PRESERVE_NAMES LIBNAME option no longer prevents in-database processing.

SAS Model Manager Changes

April 2014 Release of SAS 9.4: Changes and Enhancements

A new section was added about considerations when creating or modifying DATA step score code.

SAS 9.4: Changes and Enhancements

Two columns have been added to the model table. The ModelUUID and Notes columns assist in processing scoring models when using the SAS Embedded Process.

Autocall Macros

September 2013 Release of SAS 9.4: Changes and Enhancements

The following autocall macros are no longer needed for any DBMS. However, they are still supported. These macros initialized the publishing macros.

%INDACPF	%INDB2PF	%INDGPPM	%INDNZPM
%INDACPM	%INDB2PM	%INDNZPC	%INDORPM
%INDB2PC	%INDGPPC	%INDNZPF	%INDTDPF

%INDB2PD %INDGPPF %INDNZPJ %INDTDPM

Part 1

Introduction

Chapter 1
SAS In-Database Processing

Introduction to SAS In-Database Processing

When using conventional processing to access data inside a data source, 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 data source. 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 data source to SAS increases.

SAS in-database processing integrates SAS solutions, SAS analytic processes, and third-party data provider. Using SAS in-database processing, you can run scoring models, some SAS procedures, DS2 thread programs, and formatted SQL queries inside the data source. The following table lists the SAS products needed to use these features.

In-Database Feature	Software Required	Supported Data Providers
format publishing and the SAS_PUT( ) function	• Base SAS • SAS/ACCESS Interface to the data source	Aster DB2 under UNIX Greenplum Netezza Teradata

In-Database Feature	Software Required	Supported Data Providers
scoring models	• Base SAS • SAS/ACCESS Interface to the data source • SAS Scoring Accelerator • SAS Enterprise Miner • SAS Scalable Performance Data Server (optional) • SAS Model Manager (optional)	Aster DB2 under UNIX Greenplum Hadoop Netezza Oracle SAP HANA SPD Server Teradata
*Base SAS procedures: FREQ RANK REPORT SORT SUMMARY/MEANS TABULATE	• Base SAS • SAS/ACCESS Interface to the data source	Aster DB2 (UNIX and z/OS) Greenplum Hadoop Oracle Netezza SAP HANA 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
DS2 threaded programs	• Base SAS • SAS/ACCESS Interface to the data source • SAS In-Database Code Accelerator	Greenplum Hadoop Teradata
DATA step scoring programs	• Base SAS • SAS/ACCESS Interface to Hadoop	Hadoop

In-Database Feature	Software Required	Supported Data Providers
data quality operations	• Base SAS • SAS/ACCESS Interface to Hadoop • SAS/ACCESS Interface to Teradata • SAS In-Database Code Accelerator • SAS Data Loader for Hadoop • SAS Data Quality Accelerator for Teradata	Hadoop Teradata
extract and transform data	• Base SAS • SAS/ACCESS Interface to Hadoop • SAS/ACCESS Interface to Teradata • SAS Data Loader for Hadoop • SAS Data Loader for Hadoop • SAS Data Quality Accelerator for Teradata	Hadoop Teradata

* In-database processing of PROC RANK and PROC SORT is not supported by Hadoop.

Deployed Components for In-Database Processing

Deployed Components for Aster

Components that are deployed to Aster for in-database processing are contained in a self-extracting archive file (tkindbsrv-9.4_M2-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The archive file is located in the ***`SAS-installation-directory/`*** **`SASTKInDatabaseServer/9.4/AsternClusteronLinuxx64/`** directory.

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

In particular, the SAS System libraries, the SAS_SCORE( ) SQL/MR function, and the SAS_PUT( ) SQL/MR function are installed. The SAS Scoring Accelerator for Aster uses these libraries and the SAS_SCORE( ) SQL/MR function to run scoring models inside the database. The SAS_PUT( ) function executes the format files in Aster. The

SAS_PUT( ) function is deployed and stored in the NC_INSTALLED_FILES table under either the PUBLIC schema (4.5) or the specified schema (4.6).

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 two self-extracting archive files (acceldb2fmt-3.1-*n*_*.sh and tkindbsrv-9.4_M2-*n*_*.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The first self-extracting archive file is located in the ***SAS-installation-directory*/SASFormatsLibraryForDB2/3.1/DB2on<AIX | Linux64>/** directory. The second self-extracting archive file is located in the ***SAS-installation-directory*/SASTKInDatabaseServer/9.4/DB2on<AIX | Linuxx64>/** directory.

- The following components are deployed in the acceldb2fmt-3.1-*n*_*.sh file:
  - 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.
- The SAS Embedded Process is deployed in the tkindbsrv-9.4_M2-*n*_*.sh file. The SAS Embedded Process contains run-time libraries and other software that is installed on your DB2 system. The SAS scoring files created in DB2 access the routines within the run-time libraries.

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 two self-extracting archive files (accelgplmfmt-3.1-*n*_lax.sh and tkindbsrv-9.4_M2-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The first self-extracting archive file is located in the ***SAS-installation-directory*/SASFormatsLibraryforGreenplum/3.1/GreenplumonLinux64/** directory. The second self-extracting archive file is located in the ***SAS-installation-directory*/SASTKInDatabaseServer/9.4/ GreenplumonLinux64/** directory.

- The following components are deployed in the accelgplmfmt-3.1-*n*_lax.sh file:
  - 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 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.
- The SAS Embedded Process is deployed in the tkindbsrv-9.4_M2-*n*_lax.sh file. The SAS Embedded Process contains run-time libraries and other software that is installed on your Greenplum system. The SAS Embedded Process accesses the scoring files when a scoring operation is performed. The SAS Embedded Process also executes the DS2 thread program when using the SAS In-Database Code Accelerator.

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

Deployed Components for Hadoop

Components that are deployed to Hadoop for in-database processing are contained in a self-extracting archive file, tkindbsrv-9.42_M1-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The archive file is located in the ***SAS-installation-directory*/ SASTKInDatabaseServer/9.4/HadooponLinux64** directory.

The SAS Embedded Process and Hadoop JAR file installer are the components that are deployed in Hadoop. The SAS Embedded Process contains run-time libraries, and other software that is installed on your Hadoop system.

For more information, see 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 two self-extracting archive files (accelnetzfmt-3.1-*n*_lax.sh and tkindbsrv-9.4_M2-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The first archive file is located in the ***SAS-installation-directory*/ SASFormatsLibraryforNetezza/3.1/Netezza32bitTwinFin/** directory. The second archive file is located in the ***SAS-installation-directory*/ SASTKInDatabaseServer/9.4/Netezza64bitTwinFin** directory.

The following components are deployed in the accelnetzfmt-3.1-*n*_lax.sh file:

- 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 SAS Embedded Process is deployed in the tkindbsrv-9.4_M2-*n*_lax.sh file. The SAS Embedded Process contains run-time libraries, and other software that is installed on your Netezza system. The SAS Embedded Process accesses the scoring files when a scoring operation is performed.

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 Oracle

Components that are deployed to Oracle for in-database processing are contained in a self-extracting archive file (tkindbsrv-9.4_M2-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The archive file is located in the ***`SAS-installation-directory/`*** **`SASTKInDatabaseServer/9.4/OracleDatabaseonLinuxx64/`** directory.

The SAS Embedded Process is the component that is deployed in Oracle. The SAS Embedded Process contains run-time libraries and other software that is installed on your Oracle system. The SAS scoring files created in Oracle access the routines within the run-time libraries.

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

Deployed Components for SAP HANA

Components that are deployed to SAP HANA for in-database processing are contained in a self-extracting archive file, tkindbsrv-9.4-1-*n*_lax.sh). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The archive file is located in the ***`SAS-installation-directory/`*** **`SASTKInDatabaseServer/9.4/SAPHANAonLinux64`** directory.

The SAS Embedded Process is the component that is deployed in SAP HANA. The SAS Embedded Process contains run-time libraries and other software that is installed on your SAP HANA system. The SAS scoring files created in SAP HANA access the routines within the run-time libraries.

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

Deployed Components for SPD Server

The SAS Scoring Accelerator for SPD Server requires SAS Scalable Performance Data Server version 5.1 and SAS 9.4.

If you have a model that was produced by SAS Enterprise Miner, an active SPD Server, and a license for the SAS Scoring Accelerator for SPD Server, you have everything that is needed to run scoring models in the SPD Server. Installation of an in-database deployment package is not required.

Deployed Components for Teradata

Components that are deployed to Teradata for in-database processing are contained in two RPM files (accelterfmt-3.1-*n*.x86_64.rpm and tkindbsrv-9.42_M1-*n*.x86_64.rpm). *n* is a number that indicates the latest version of the file. If this is the initial installation, *n* has a value of 1. Each time you reinstall or upgrade, *n* is incremented by 1.

The first RPM file is located in the ***`SAS-installation-directory`*****`/SASFormatsLibraryforTeradata/3.1/TeradataonLinux/`** directory. The second RPM file is located in the ***`SAS-installation-directory`*****`/SASTKInDatabaseServer/9.4/TeradataonLinux/`** directory.

The components that are deployed are the SAS formats library and the SAS Embedded Process.

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 can access the routines within its run-time library.

The SAS Embedded Process contains run-time libraries, and other software that is installed on your Teradata system. The SAS scoring files created in Teradata access the routines within the run-time libraries. The SAS Embedded Process also executes the DS2 thread program when using the SAS In-Database Code Accelerator.

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 13
Publish user-defined formats and use the SAS_PUT( ) function.	Chapter 16, “Deploying and Using SAS Formats inside the Database,” on page 197
Run procedures inside the database.	Chapter 22, “Running SAS Procedures inside the Database,” on page 265

In-Database Processing Task	Documentation
Run DS2 thread programs inside the database.	Chapter 14, “Using the SAS In-Database Code Accelerator,” on page 175
Run DATA step scoring programs in Hadoop.	Chapter 15, “DATA Step Processing in Hadoop,” on page 191
Perform data quality operations	*SAS Data Loader for Hadoop: User's Guide*
Extract and transform data	*SAS Data Loader for Hadoop: User's Guide*

Part 2

SAS Scoring Accelerator

Chapter 2

Introduction to the SAS Scoring Accelerator

SAS Scoring Accelerator for SAS/ACCESS Databases

Overview of the SAS Scoring Accelerator for SAS/ACCESS Databases

When using conventional processing to access data inside a data source, 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 data source. 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. This happens because the amount of data that is fetched from the data source to the SAS scoring process increases.

The SAS Scoring Accelerator embeds the robustness of SAS Enterprise Miner scoring models directly in the highly scalable data source. By using the SAS In-Database technology and the SAS Scoring Accelerator, the scoring process is done inside the data source 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 files or functions that can be deployed inside the data source. After the scoring functions are published, the functions extend the data source's SQL language and can be used in SQL statements like other data source functions. After the scoring files are published, they are used by the SAS Embedded Process to run the scoring model.

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 data source to run the scoring model. The publishing client then uses the SAS/ACCESS Interface to the data source to publish the files to the data source.

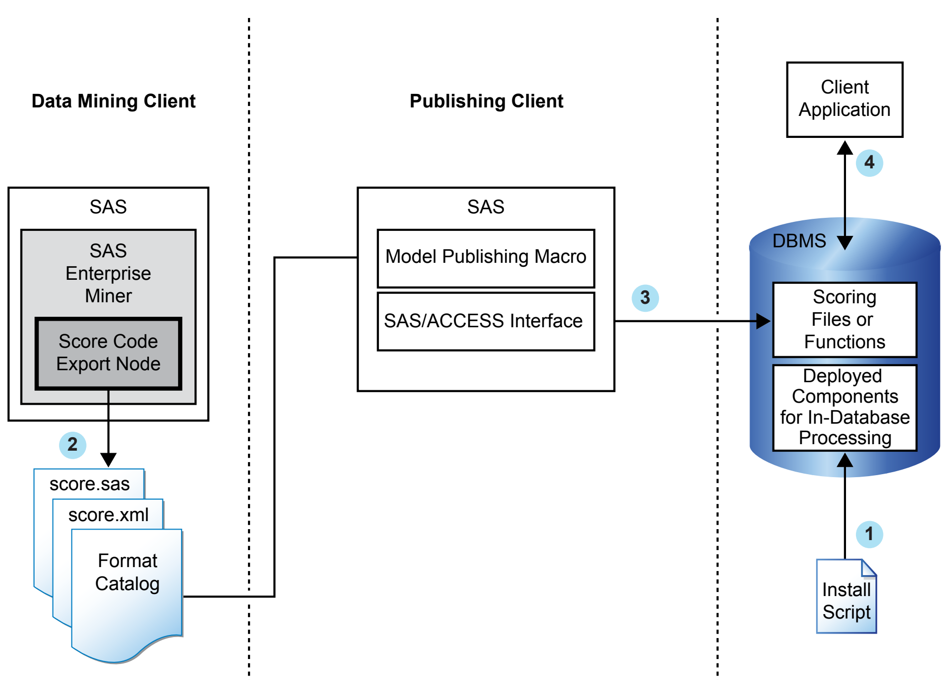
You can also use the SAS Scoring Accelerator and SAS Model Manager to import SAS/STAT linear models and SAS High-Performance Analytics models from a SAS package file (.SPK). Models that have a DATA step score code type can be scored, published, and included in performance monitoring. For more information, see the *SAS Model Manager: User's Guide*.

How It Works for SAS/ACCESS Databases

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 data source. Inside the data source, one or more scoring files or functions are created and registered for use in SQL queries.

The following figure illustrates this process.

Figure 2.1 *Process Flow Diagram*



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

The components that are deployed are different for each data source. 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. Use the Score Code Export node to export files that are used to create the scoring files or functions to a score output directory.

For more information, see Chapter 3, “Exporting the Scoring Model Files from SAS Enterprise Miner,” on page 23.

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

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

4 After the scoring files or functions are created, you can run your scoring model.

For more information, see the topic on running the scoring model in the Scoring Accelerator chapter for your data source.

SAS Scoring Accelerator for SPD Server

Overview of the SAS Scoring Accelerator for SPD Server

The SAS Scoring Accelerator for SPD Server embeds the robustness of SAS Enterprise Miner scoring models directly in the highly scalable SPD Server. By using the SAS In-Database technology and the SAS Scoring Accelerator, the scoring process is done inside the SPD Server.

The SAS Scoring Accelerator for SPD Server takes the models that are developed by SAS Enterprise Miner and creates SPD server tables.

The SAS Scoring Accelerator consists of three components:

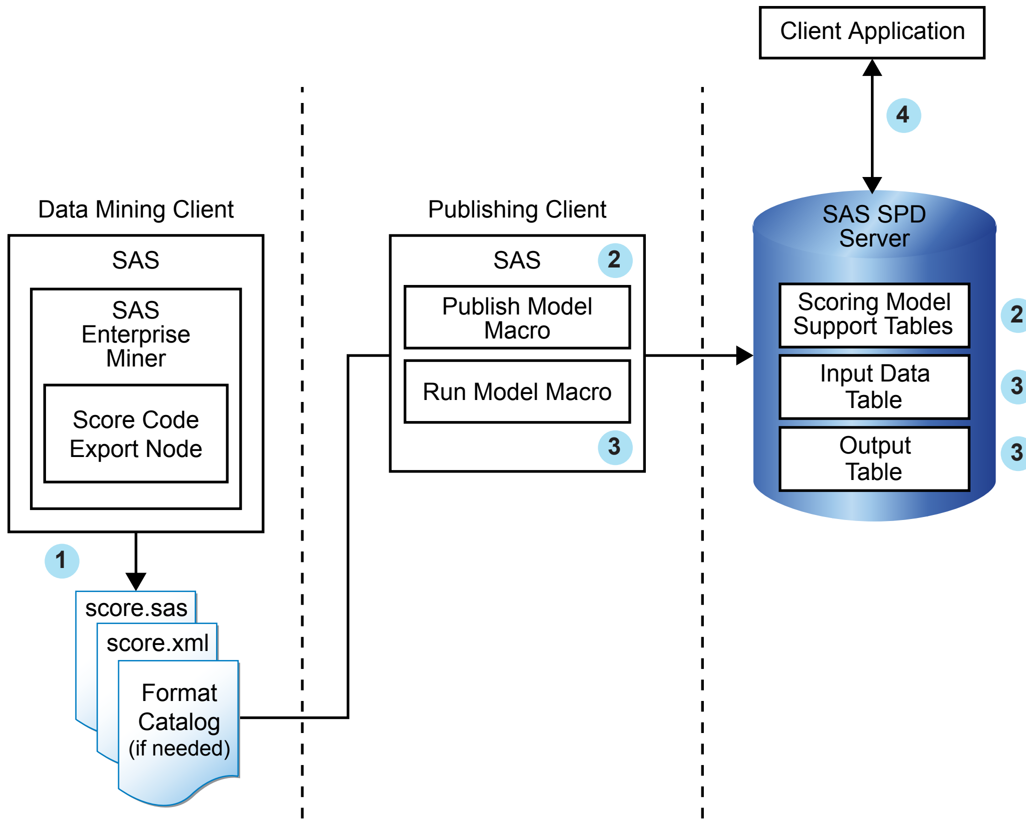
- the Score Code Export node in SAS Enterprise Miner. This extension exports the model scoring logic (including metadata about the required input and output variables) from SAS Enterprise Miner.
- a scoring publishing macro. This macro translates the scoring model into two or three SPD server tables that are needed to run the scoring model.
- a run model macro. This macro takes the SPD Server tables that are produced by the scoring publishing macro and an input data table and runs the scoring model. This macro produces an SPD Server table that contains the output from the scoring model.

How It Works for SAS SPD Server

Using SAS Enterprise Miner, you can generate SAS DATA step code. The SAS Scoring Accelerator for SPD Server takes the scoring model code, the associated property file that contains model inputs and outputs, and, if needed, a catalog of user-defined formats. The SAS Scoring Accelerator for SPD Server deploys (or publishes) them to the SPD Server. Inside the SPD Server, SPD Server tables are created and used to run the scoring model. An output table that contains the results is created.

Note: Only the installation of Base SAS and SAS Scalable Performance Data Server is required for the in-database processing.

The following figure illustrates this process.



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

For more information, see Chapter 3, "Exporting the Scoring Model Files from SAS Enterprise Miner," on page 23.

2 Start SAS and run the publish model macro. This macro creates the SPD Server tables that are needed to run the scoring model.

For more information, see "Running Scoring Models in SPD Server" on page 139 and "%INDSP_PUBLISH_MODEL Macro Syntax" on page 142.

3 Run the run model macro. This macro runs the scoring model and creates an output table.

For more information, see "Running Scoring Models in SPD Server" on page 139 and "%INDSP_PUBLISH_MODEL Macro Syntax" on page 142.

4 Use PROC SQL to query the output table data.

For more information, see "Scoring Output" on page 145.

Scoring with User-Defined Functions and the SAS Embedded Process

There are two methods by which format publishing models are processed inside the data source:

- user-defined functions

  Scoring models are converted by the publishing macros into scoring functions that are similar to any user-defined functions in the data source.

  In-database processing of scoring models by means of user-defined functions is supported by DB2 under UNIX, Greenplum, Netezza, and Teradata.

- SAS Embedded Process

  The SAS Embedded Process is a SAS server process that is installed and runs inside the data source to read and write data from the data source. The advantage of using the SAS Embedded Process is that a single function or a stored procedure is used instead of multiple, user-defined functions.

  The SAS Embedded Process is supported for Aster, DB2, Greenplum, Hadoop, Netezza, Oracle, SAP HANA, and Teradata scoring models.

  The SAS Embedded Process is one of the deployed components for in-database processing. For more information, see the *SAS In-Database Products: Administrator's Guide*.

Considerations When Creating or Modifying DATA Step Score Code

How the SAS Scoring Accelerator Processes the DATA Step Score Code

The score.sas file is DATA step score code and is used as input by the SAS Scoring Accelerator. You can generate DATA step score code by using SAS Enterprise Miner, SAS Model Manager, SAS/STAT, and other SAS software.

Some SAS language elements and syntax are not supported when you create or modify your score code. Only the SAS language elements and syntax that are required to run critical data transformations and model scoring functions are available. If you use a statement or function that is not supported, an error occurs and your model is not published to the data source.

For more information, see “Supported Language Elements and Syntax” on page 17.

Supported Language Elements and Syntax

The following items describe the syntax that is supported:

- all forms of DATA step array brackets (‘[‘. ‘{‘, ‘(‘).
- declaration statements using LENGTH.

This means declarations including lists of variables with the standard DATA step numeric or character length specifications (for example, 8 and $32).

- attribute statements, although these have no semantic meaning.

  Only the syntax is supported. That is, you do not get a syntax error for them.

- array statements and array initializers.

  This includes both temporary and variable arrays. For variable arrays, DATA step aliasing variables are available. However, these variable arrays are implemented literally as array references instead of as variable aliases as in the DATA step.

- standard control structures, including IF, THEN, ELSE, DO, WHILE, UNTIL, SELECT/WHEN/OTHERWISE, CONTINUE, LEAVE, RETURN, LINK, and GOTO.

  The standard **`do i = 1 to n`** syntax is supported. WHILE and UNTIL are also supported. However, features such as list syntax, **`do i= 1,3,5`**, are not supported.

- STOP and RUN are syntactically supported but have no semantic meaning.
- assignment statements using arrays and scalars.
- SUBSTR references, including left-hand-side 'pseudo' substring.
- basic syntax for the PUT statement using lists of variables.

  Line and column controls are not supported.

- DROP, FORMAT, and LABEL statements.

  FORMAT and LABEL syntax is supported, but the format or label information is not used.

  DROP is supported both syntactically and semantically. Dropped variables are made into local variables and are not included in any output table. Basic lists of variables are supported for DROP. Some syntax is supported for variable enumeration such as **`drop A1-A3`**. Colon syntax (**`A:`**) is not supported.

- variable array syntax, variable dash lists, and OF lists (for example, **`a1-a10`** and **`sum(of a[*])`**).
- all DATA Step expressions.
- constant lists (as used in IN clauses and array initializations).

  This includes standard lists such as (1,2,3,4) and those including iterators such as (4 * 99). Array initializations are translated into DS2 array assignment statements.

- some hash object syntax.

  This includes the basic declaration constructor and the DEFINEKEY, DEFINEDATA, DEFINEDONE, ADD, REPLACE, FIND, and CLEAR methods.

- Format justifiers ('L', 'C', 'R') and some PUT modifiers ('?') are syntactically supported.
- If you use the SAS Embedded Process to run your scoring model, you can use any function that is supported by the DS2 language. For more information, see “DS2 Functions” in *SAS DS2 Language Reference*.
- If you use scoring functions to run your scoring model, only the following functions are supported:

  ABS
  ARCOS

ARSIN
ATAN
ATAN2
CEIL
COS
COSH
DMNORM
DMRAN
DMINIT
EXP
FLOOR
INDEX
INT
LEFT
LENGTH
LOG
LOG10
LOWCASE
MAX
MIN
MISSING
MOD
N
NMAX
SIN
SINH
SQRT
STRIP
SUBSTR
SUM
TAN
TANH
TRIM
UPCASE

Note: The KLEFT, KTRIM, KLENGTH, KLOWCASE, KUPCASE, and KINDEX functions are syntactically supported by mapping each to its corresponding standard function.

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(%))
¬	%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

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).

- an XML file that contains the scoring variables and other properties 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 25. For more information about using SAS Enterprise Miner, see the SAS Enterprise Miner online 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.

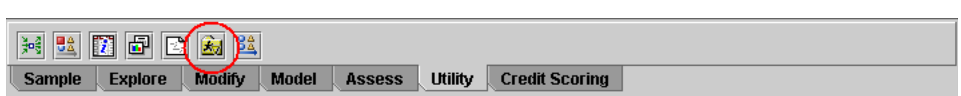
The Score Code Export node exports score code created by SAS Enterprise Miner into a format that can be used by the SAS Scoring Accelerator. The exported files are stored in a directory, not the SAS Metadata Server.

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

Using the Score Code Export Node in a Process Flow Diagram

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

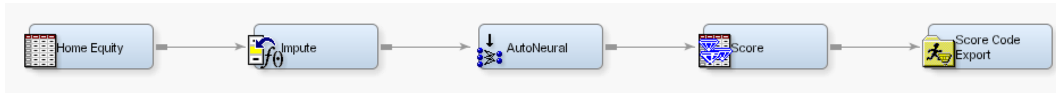
Figure 3.1 The Diagram Toolbar with the SAS Score Code Export Node Icon Highlighted



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 process flow diagram and transfers it to the Score Code Export node. 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. To see the SAS Enterprise Miner nodes that produce score code, see the *SAS Enterprise Miner Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS*.

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 each of the following properties is set to the default value of Yes:
   - **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 the *SAS Enterprise Miner Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS*.

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 appears and indicates 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 25.

Output Created by the Score Code Export Node

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 27. *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

Each node in SAS Enterprise Miner creates different types of score code.

These types can include the following:

SAS DATA Step
SAS Program
PMML
C
Java
DBMS

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.

Note: There is limited support for user-written code in the Variable Clustering and Rules Builder nodes. User-written code could produce errors or unexpected results.

For information about the Enterprise Miner nodes and the type of score code that each node produces, see the *SAS Enterprise Miner Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS*.

Chapter 4
SAS Scoring Accelerator for Aster

Overview of Running Scoring Models in Aster

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

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

The SAS Embedded Process is a SAS server process that runs inside Aster to read and write data. The model publishing macro creates scoring files that are then used in a stored procedure to run the scoring model.

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 a table. For more information, see "Scoring Files Table" on page 37.

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 37.

The SAS Scoring Accelerator for Aster requires a specific version of the Aster client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

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. Populate the directory with the score.sas file, the score.xml file, and, if needed, the format catalog.
3. Start SAS and submit one of the following commands in the Program Editor or Enhanced Editor:

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

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

   For more information, see the "INDCONN Macro Variable" on page 30.
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 32.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Aster. 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 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 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 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 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 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 4.6. For Aster 4.5, the scoring model XML files are published to the PUBLIC schema.

Requirement: Any schema that is used must be in the search path.

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 19

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 Scoring Files" on page 35.
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 27 and "Aster Scoring Files" on page 35.

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. For Aster 4.5, this file is stored in the NC_INSTALLED_FILES table under the PUBLIC schema. For Aster 4.6, this file is stored in the NC_USER_INSTALLED_FILES table under the schema that you specified in the INDCONN macro variable.

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 1660.
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. For Aster 4.5, this file is stored in the NC_INSTALLED_FILES table under the PUBLIC schema. For Aster 4.6, this file is stored in the NC_USER_INSTALLED_FILES table under the schema that you specified in the INDCONN macro variable.

DATABASE=*database-name*
specifies the name of an Aster 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 30.
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 either the NC_INSTALLED_FILES table (Aster 4.5) or the NC_USER_INSTALLED_FILES table (Aster 4.6) in the database.

Default CREATE

Tip If the scoring files have been previously defined and you specify ACTION=CREATE, you receive warning messages from Aster. 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 Scoring Files” on page 35.

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

See “Special Characters in Directory Names” on page 19

Model Publishing Macro Example

```
%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 285.

- sasscore_score_io.xml. See "Example of an Input and Output Variables Scoring File" on page 305.
- sasscore_score_ufmt.xml. See "Example of a User-Defined Formats Scoring File" on page 312.

After the scoring files are installed, they can be invoked in Aster using the SAS_SCORE() function. For more information, see "SAS_SCORE() Function" on page 37.

Aster Permissions

For Aster 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 4.6, the following permissions are needed for the schema by the person who runs the scoring publishing macros.

- USAGE permission
- INSTALL FILE permission
- CREATE permission

These permissions are needed because all functions and files can be published to a specific schema. 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 "Aster Permissions" in Chapter 2 of *SAS In-Database Products: Administrator's Guide*.

Scoring Files and Functions inside the Aster Database

Aster 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.

For Aster 4.5, these files are stored in the NC_INSTALLED_FILES table under the PUBLIC schema. For Aster 4.6, these files are stored in the NC_USER_INSTALLED_FILES table under the schema that you specified in the INDCONN macro variable. See Appendix 1, "Scoring File Examples," on page 285 for an example of each of these files.

Note: When you publish a model using Aster 4.5, you are likely to receive warnings about multiple lengths and unbalanced quotation marks. This warning does not keep the model from being published successfully. The error occurs because the .ds2 scoring file is inserted into an Aster system table as a long quoted string.

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

- Log on to the database using the Aster 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 filename 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 filename, fileowner, uploadtime
         from nc_user_installed_files where
            name like 'sasscore_reg%');
   disconnect from aster;
quit;
```

- 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_MODEL macro.

  The SampleSQL.txt file contains basic SQL code that can be used to run your score code inside Aster. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

  For example, the SampleSQL.txt file refers to an ID column in **score_outtab** that is populated with a unique integer from 1 to *n*. *n* is the number of rows in the table.

  Note: 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 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 that is created when the %INDAC_PUBLISH_MODEL macro was run. 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. The SAS_SCORE() function is deployed and stored in the PUBLIC schema during the installation and configuration of the in-database deployment for Aster.

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

Scoring Files Table

The NC_INSTALLED_FILES table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *model-name.model-table-name*.

Column Name	Description	Specifications
ModelName	contains the name of the model	VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC
ModelDS2	contains the sasscore_*modelname*.ds2 file	BLOB(209708800)
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file	BLOB(209708800)
ModelOwner	contains the name of the user who published the model	VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC
ModelUpdated	contains the date and time that the model was published	TIMESTAMP(6)

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 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('*schema-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.

Default	your default schema. To determine your default schema name, use the show **search_path command** from the Aster Client Tool (ACT).
Restriction	This argument is valid only for Aster 4.6. For Aster 4.5, the scoring model files are published to the PUBLIC schema.
Requirement	Any schema that is used must be in the search path.

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

Overview of Running Scoring Models in DB2

There are two ways to run scoring models in DB2.

- You can create scoring functions for each EM_ output variable. The model publishing macro creates the scoring functions that are published as DB2 user-defined functions. These functions can then be used in any SQL query. For more information, see "Using Scoring Functions to Run Scoring Models" on page 40.
- You can use the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs inside DB2 to read and write data. The model publishing macro creates scoring files. These scoring files are then used by a DB2 built-in function to run the scoring model. For more information, see "Using the SAS Embedded Process to Run Scoring Models" on page 45.

The SAS Scoring Accelerator for DB2 requires a certain version of the DB2 client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Using Scoring Functions to Run Scoring Models

How to Run a Scoring Model Using Scoring Functions

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 27.

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 *Configuring SSH Client Software in UNIX and Windows Environments for Use with the SFTP Access Method in SAS 9.2, SAS 9.3, and SAS 9.4* located at http://support.sas.com/techsup/technote/ts800.pdf.

To run the scoring model using scoring functions, follow these steps.

1. Run the %INDB2_PUBLISH_MODEL macro.

   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 re-created. 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, one of the following actions must occur:

- The database must be stopped and restarted to clean up the cache.
- 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.

2. Use the scoring functions in any SQL query.

   For more information, see "Using Scoring Functions to Run a Scoring Model" on page 44.

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 52.

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 27.

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 %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.

Viewing 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 "Using Scoring Functions to Run a Scoring Model" on page 44.

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 that 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;
```

- 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 SQL code that can be used to run your score code inside DB2. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

  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*. *n* is the number of rows in the table.

  Note: 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 that is created when the %INDB2_PUBLISH_MODEL macro was run. A message that indicates whether a scoring function is successfully or not successfully executed is printed to the SAS log.

Using Scoring Functions to Run a Scoring Model

The scoring functions are available to use in any SQL expression in the same way that DB2 built-in functions are used.

The following example code creates the scoring functions.

```
%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;
```

Using the SAS Embedded Process to Run Scoring Models

How to Run a Scoring Model with the SAS Embedded Process

The integration of the SAS Embedded Process and DB2 allows scoring code to run directly using the SAS Embedded Process on DB2.

Note: The SAS Embedded Process might require a later release of DB2 than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.

To run the scoring model using the SAS Embedded Process, follow these steps.

1. Create a table to hold the scoring files.

   The %INDB2_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

   For more information, see "Creating a Model Table" on page 46.

2. Run the %INDB2_PUBLISH_MODEL to create the scoring files.

   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:

   - translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process.
   - 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 DB2 to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDB2_CREATE_MODELTABLE macro.

   For more information, see "Running the %INDB2_PUBLISH_MODEL Macro" on page 50 and "DB2 Scoring Files" on page 49.

3. Use the ANALYZE_TABLE function in the FROM clause in any SQL expression to run the scoring model.

   For more information, see "ANALYZE_TABLE Function" on page 48.

Creating a Model Table

Overview

When using the SAS Embedded Process to publish a scoring model in DB2, you must create a table to hold the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files. You must run the %INDB2_CREATE_MODELTABLE macro to create the table before you run the %INDB2_PUBLISH_MODEL macro.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *schema-name.model-table-name*.

Column Name	Description	Specification
ModelName	contains the name of the model	VARCHAR(128) NOT NULL PRIMARY KEY
ModelDS2	contains the sasscore_*modelname*.ds2 file	BLOB(4M) NOT NULL
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file	BLOB(4M)
ModelMetadata	Reserved by SAS for future use	BLOB(4M)
ModelUUID*	contains the UUID of the source model	VARCHAR (36)
Notes*	contains additional information that describes the source model	VARCHAR (512)

* This column is for use by SAS Model Manager. If you have a model table that was created prior to SAS 9.4 and you want this column in your model table, you must run the %INDB2_CREATE_MODELTABLE macro to re-create your model table.

%INDB2_CREATE_MODELTABLE Run Process

To run the %INDB2_CREATE_MODELTABLE macro, complete the following steps:

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 50.

2. Run the %INDB2_CREATE_MODELTABLE macro.

   For more information, see "%INDB2_CREATE_MODELTABLE Macro Syntax" on page 47.

%INDB2_CREATE_MODELTABLE Macro Syntax

%INDB2_CREATE_MODELTABLE

(TS_PRIMARYPAR=*tablespace-name*
<, DATABASE=*database-name*>
<, MODELTABLE=*model-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

TS_PRIMARYPAR=*tablespace-name*
specifies the name of the tablespace that resides in the primary partition.

Tip: You can get the name of the tablespace from your database administrator.

DATABASE=*database-name*
specifies the name of a DB2 database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default: The database specified in the INDCONN macro variable or your current database

MODELTABLE=*model-table-name*
specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default	sas_model_table
Requirements	The maximum table name length is 128 characters and it must be a valid DB2 table name.
	The table name that you specify for this macro must be the same table name that is used in the %INDB2_PUBLISH_MODEL macro.
See	"%INDB2_PUBLISH_MODEL Macro Syntax" on page 52

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

CREATE
creates a new table.

Tip: If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

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

Tip: If you specify ACTION = REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP
causes all models in this table to be dropped.

Default: CREATE

ANALYZE_TABLE Function

Overview of the ANALYZE_TABLE Function

The ANALYZE_TABLE function is the interface for running the scoring model inside DB2 with the SAS Embedded Process. The ANALYZE_TABLE function uses the information that is stored in the model table. The ANALYZE_TABLE function is a built-in DB2 function.

Using the ANALYZE_TABLE Function

You can use the ANALYZE_TABLE function using explicit pass-through and PROC SQL or you can use other DB2 query tools such as the Command Line Processor. Use the ANALYZE_TABLE function in the FROM clause in any SQL expression to run the scoring model.

TIP 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 %INDB2_PUBLISH_MODEL macro. The SampleSQL.txt file contains basic SQL code that can be used to run your score code inside DB2. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

Note: Before using the ANALYZE_TABLE function with the SAS Embedded Process, you must create the model table with the %INDB2_CREATE_MODELTABLE macro. Then, you must publish the files to the model table with the %INDB2_PUBLISH_MODEL macro. For more information, see “Creating a Model Table” on page 46 and “Running the %INDB2_PUBLISH_MODEL Macro” on page 50.

Here is an example using PROC SQL.

```
proc sql;
 connect to db2 (user=userid password=xxxx database=mydatabase);
 create table work.sas_score_out1 as select * from connection to db2
    (WITH T1 as (SELECT * from SCORE_INPUT_TABLE where X1 < 1.0)
      SELECT * from T1 ANALYZE_TABLE
         (IMPLEMENTATION 'PROVIDER=SAS;
            ROUTINE_SOURCE_TABLE=myschema.SAS_PUBLISH_MODEL;
            ROUTINE_SOURCE_NAME="Intr_Tree";') );
 disconnect from db2;
quit;
```

ANALYZE_TABLE Function Syntax

The syntax of the ANALYZE_TABLE function is as follows:

FROM *input-table-name* **ANALYZE_TABLE** (IMPLEMENTATION 'PROVDER=SAS’;
ROUTINE_SOURCE_TABLE=*schema.model-table-name*;
ROUTINE_SOURCE_NAME="*model-name*"; ')

Arguments

input-table-name
specifies the input table that is used by the ANALYZE_TABLE function.

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

model-table-name

specifies the name of the model table where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files were published with the %INDB2_CREATE_MODELTABLE macro.

Requirement The table name that you specify for this function must be the same table name that is used in the %INDB2_CREATE_MODELTABLE macro. For more information, see "%INDB2_CREATE_MODELTABLE Macro Syntax" on page 47.

model-name

specifies the name of the model.

DB2 Scoring Files

When using the SAS Embedded Process, the %INDB2_PUBLISH_MODEL macro produces two scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the ANALYZE_TABLE function.
- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published. This file is used by the ANALYZE_TABLE function.

These files are published to the model table that you specify in the %INDB2_PUBLISH_MODEL macro. See Appendix 1, "Scoring File Examples," on page 285 for an example of each of these files.

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

Although you cannot view the scoring files directly, there are two ways to see the models whose files are created:

- Run this query from the DB2 command line processor:

```
db2> connect to databasename user userid using password
db2> select modelname from sasmodeltablename
```

- Run a PROC SQL query from SAS.

```
proc sql;
 connect to db2 (user=userid password=xxxx database=mydatabase);
  select * from connection to db2
 (select modelname from sas_model_table);
 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*.

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 and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 50.
4. If you use the SAS Embedded Process, run the %INDB2_CREATE_MODELTABLE macro.

   For more information, see "Creating a Model Table" on page 46.
5. Run the %INDB2_PUBLISH_MODEL macro.

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

   For more information, see "%INDB2_PUBLISH_MODEL Macro Syntax" on page 52.

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:

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

Arguments

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`.

SERVER=*server*

specifies the DB2 server name or the IP address of the server host.

Restriction: This argument is required when using function-based scoring. It is not used if you use the SAS Embedded Process.

Requirement: The name must be consistent with how the host name was cached when SFTP 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 SFTP 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 SFTP command, see “DB2 Installation and Configuration Steps” in the *SAS In-Database Products: Administrator's Guide*.

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.

Restriction: This argument is not used if you use the SAS Embedded Process.

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). In order for the SFTP process to be successful, Pageant must be running on Windows and the SSH agent must be running on UNIX.

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*
<, MECHANISM=STATIC | EP>
<, MODELTABLE=*model-table-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 19

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. If you use the SAS Embedded Process, the model name is the primary index field in the model table.

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 41.
Requirement	If you use scoring functions, the model name must be a valid SAS name that is 10 characters or fewer. If you use the SAS Embedded Process, the model name can be up to 128 characters. 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 27 and "Scoring Function Names" on page 41.

MECHANISM=STATIC | EP

specifies whether scoring functions or scoring files are created. MECHANISM= can have one of the following values:

STATIC

specifies that scoring functions are created.

These scoring functions are used in an SQL query to run the scoring model.

See "Using Scoring Functions to Run Scoring Models" on page 40

EP

specifies that scoring files are created.

These scoring files are used by the SAS Embedded Process to run the scoring model. A single entry in the model table is inserted for each new model. The entry contains both the score.sas and score.xml in separate columns. The scoring process includes reading these entries from the table and transferring them to each instance of the SAS Embedded Process for execution.

Requirement	If you specify MECHANISM=EP, you must also specify the MODELTABLE= argument.
Note	The SAS Embedded Process might require a later release of DB2 than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.
See	"Using the SAS Embedded Process to Run Scoring Models" on page 45

Default STATIC

MODELTABLE=*model-table-name*

specifies the name of the model table where the scoring files are published.

Default	sas_model_table
Restriction	This argument is valid only when using the SAS Embedded Process.
Requirement	The name of the model table must be the same as the name specified in the %INDB2_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in "%INDB2_CREATE_MODELTABLE Macro Syntax" on page 47.

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.
	If you use scoring functions to run scoring models, the maximum number of output variables is 128. If you use the SAS Embedded Process, the maximum depends on the page size of the database table space. For a 4K page size database, the limit is 500. If you have it configured for any of the larger page sizes (8K, 16K, 32K), then the limit is 1012.
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 or the scoring files are published.

Requirements	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`**.
	If you use the SAS Embedded Process, the name of the database must be the same as the database specified in the %INDB2_CREATE_MODELTABLE macro. For more information, see the DATABASE argument in “%INDB2_CREATE_MODELTABLE Macro Syntax” on page 47.
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 50.

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 new functions or files.

REPLACE

overwrites the current functions or files, if functions or files by the same name are already registered.

DROP

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

Default	CREATE
Tip	If the function or file has been previously defined and you specify ACTION=CREATE, you receive warning messages from DB2. If the function or file 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
Restriction	This argument is valid only when using the scoring functions. It has no effect if you specify MECHANISM=EP.
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 56

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
Restriction	This argument is valid only when using the scoring functions. It has no effect if you specify MECHANISM=EP.
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 that 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 plus 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
Restriction	This argument is valid only when using the scoring functions. It has no effect if you specify MECHANISM=EP.
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 that 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 plus 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 41.

Tip	This argument is useful when testing your scoring models.
See	"Special Characters in Directory Names" on page 19

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.

The SAS Embedded Process always operates in its own process that is equivalent to fenced mode functions. An optimized data transport mechanism allows the SAS Embedded Process to provide fenced mode protection with speed that is as good as or better than unfenced functions.

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.

DB2 Permissions

Scoring Function 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 more information about specific permissions, see “DB2 Permissions” in Chapter 3 of *SAS In-Database Products: Administrator's Guide*.

SAS Embedded Process Permissions

You must have CREATE TABLE user permission to create a model table when using the SAS Embedded Process.

For more information about specific permissions, see “DB2 Permissions” in Chapter 3 of *SAS In-Database Products: Administrator's Guide*.

Chapter 6

SAS Scoring Accelerator for Greenplum

Overview of Running Scoring Models in Greenplum

There are two ways to run scoring models in Greenplum.

- You can create scoring functions for each EM_ output variable. The model publishing macro creates the scoring functions that are published as Greenplum user-defined functions. These functions can then be used in any SQL query. For more information, see "Using Scoring Functions to Run Scoring Models" on page 60.
- You can use the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs inside Greenplum to read and write data. The model publishing macro creates scoring files. These scoring files are then used by a Greenplum built-in function to run the scoring model. For more information, see "Using the SAS Embedded Process to Run Scoring Models" on page 65.

The SAS Scoring Accelerator for Greenplum requires a certain version of the Greenplum client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Using Scoring Functions to Run Scoring Models

How to Run a Scoring Model Using Scoring Functions

The %INDGP_PUBLISH_MODEL macro creates the files that are needed to build the scoring functions. The macro then 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 27.

To run the scoring model using scoring functions, follow these steps.

1. Run the %INDGP_PUBLISH_MODEL macro. 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.
   - processes the format catalog, if a format catalog is available, 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 in 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. These shared object files 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.

2. Use the scoring functions in any SQL query.

   For more information, see "Using Scoring Functions to Run a Scoring Model" on page 64.

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 73.

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 27.

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 %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.

Viewing 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 "Using Scoring Functions to Run a Scoring Model" on page 64.

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 greenplm (<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);

  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;
  ```

- 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 SQL code that can be used to run your score code inside Greenplum. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

  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*. *n* is the number of rows in the table.

Note: 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 that is created when the %INDGP_PUBLISH_MODEL macro was run. A message that indicates whether a scoring function is successfully or not successfully executed is printed to the SAS log.

Using Scoring Functions to Run a Scoring Model

The scoring functions are available to use in any SQL expression in the same way that Greenplum built-in functions are used.

The following example code creates the scoring functions.

```
%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;
```

Using the SAS Embedded Process to Run Scoring Models

How to Run a Scoring Model with the SAS Embedded Process

The integration of the SAS Embedded Process and Greenplum allows scoring code to run directly using the SAS Embedded Process on Greenplum.

Note: The SAS Embedded Process might require a later release of Greenplum than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.

To run the scoring model using the SAS Embedded Process, follow these steps.

1. Create a table to hold the scoring files.

   The %INDGP_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

   For more information, see “Creating a Model Table” on page 66.

2. Run the %INDGP_PUBLISH_MODEL to create the scoring files.

   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:

- translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process.
- 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 Greenplum to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDGP_CREATE_MODELTABLE macro.

For more information, see "Running the %INDGP_PUBLISH_MODEL Macro" on page 71 and "Greenplum Scoring Files" on page 70.

3. Use the SAS_EP function in the FROM clause in any SQL expression to run the scoring model.

For more information, see "SAS_EP Function" on page 68.

Creating a Model Table

Overview

When using the SAS Embedded Process to publish a scoring model in Greenplum, you must create a table to hold the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files. You must run the %INDGP_CREATE_MODELTABLE macro to create the table before you run the %INDGP_PUBLISH_MODEL macro.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *schema-name.model-table-name*.

Column Name	Description	Specification
ModelName	contains the name of the model	VARCHAR(128) NOT NULL PRIMARY KEY
ModelDS2	contains the sasscore_*modelname*.ds2 file	BYTEA NOT NULL
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file	BYTEA
ModelMetadata	Reserved by SAS for future use	BYTEA
ModelUUID*	contains the UUID of the source model	VARCHAR (36)
Notes*	contains additional information that describes the source model	VARCHAR (512)

* This column is for use by SAS Model Manager. If you have a model table that was created prior to SAS 9.4 and you want this column, you must run the %INDGP_CREATE_MODELTABLE macro to re-create your model table.

%INDGP_CREATE_MODELTABLE Run Process

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

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 71.

2. Run the %INDGP_CREATE_MODELTABLE macro.

   For more information, see "%INDGP_CREATE_MODELTABLE Macro Syntax" on page 67.

%INDGP_CREATE_MODELTABLE Macro Syntax

%INDGP_CREATE_MODELTABLE

<DATABASE=*database-name*>
<, MODELTABLE=*model-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

DATABASE=*database-name*
specifies the name of a Greenplum database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default The database specified in the INDCONN macro variable or your current database

MODELTABLE=*model-table-name*
specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default	sas_model_table
Requirements	The maximum table name length is 63 characters and it must be a valid Greenplum table name.
	The table name that you specify for this macro must be the same table name that is used in the %INDGP_PUBLISH_MODEL macro.
See	"%INDGP_PUBLISH_MODEL Macro Syntax" on page 73

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

CREATE
creates a new table.

Tip If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

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

Tip If you specify ACTION = REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP
causes all models in this table to be dropped.

Default CREATE

SAS_EP Function

Overview of the SAS_EP Function

The SAS_EP function is the interface for running the scoring model inside Greenplum with the SAS Embedded Process. The SAS_EP function uses the information that is stored in the model table. The SAS_EP function is a built-in Greenplum function.

Using the SAS_EP Function

You can use the SAS_EP function using explicit pass-through and PROC SQL or you can use other Greenplum query tools such as psql. Use the SAS_EP function in the FROM clause in any SQL expression to run the scoring model.

TIP 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 %INDGP_PUBLISH_MODEL macro. The SampleSQL.txt file contains basic SQL code that can be used to run your score code inside Greenplum. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

Note: Before using the SAS_EP function with the SAS Embedded Process, you must create the model table with the %INDGP_CREATE_MODELTABLE macro. Then, you must publish the files to the model table with the %INDGP_PUBLISH_MODEL macro. For more information, see "Creating a Model Table" on page 66 and "Running the %INDGP_PUBLISH_MODEL Macro" on page 71.

Here is an example using PROC SQL.

```
%let indconn = user=user1 password=open1 dsn=dsn6 schema=GPschema;

%indgp_publish_model
  (dir= C:\models,
   modelname= almush02,
   action=create,
   mechanism=ep,
   outdir=C:\test
   );

proc sql noerrorstop;
connect to greenplm (user=user1 password=open1 dsn=dsn6 schema=GPschema);
create table test.dbscore as select * from connection to greenplm
   (select id,
    "EM_CLASSIFICATION" ,
    "EM_EVENTPROBABILITY" ,
     "EM_PROBABILITY" from public.SAS_EP(TABLE(select id
     ,"capcolor"
     ,"capsurf"
     ,"odor"
     ,"ringnumb"
     ,"sporepc"
```

```
     ,"stalkcbr"
     ,"stalksbr"
    from model.almush02),
      'select modelds2, modelformats from model.sas_model_table
         where upper(modelname)=''ALMUSH02''
      ');
    ) ;
quit;
```

SAS_EP Function Syntax

The basic syntax of the SAS_EP table function is as follows:

<*SAS_EP-schema*.>**SAS_EP**(TABLE (SELECT
* | *column* <, … *column-n*>
, FROM <*input-table-schema*.>*input-table-name*
<SCATTER BY *column*<, … *column-n*> | SCATTER RANDOMLY>
<ORDER BY *column* <, … *column-n*>>),
'SELECT MODELDS2<, MODELFORMATS> FROM <*schema*.>*model-table-name*
WHERE MODELNAME =' '*model-name*' '
');

Arguments

SAS_EP-schema
specifies the name of the schema where the SAS_EP function was created.

Note The SAS_EP function is created in the database by the %INDGP_PUBLISH_COMPILEUDF_EP macro. For more information, see *SAS In-Database Products: Administrator's Guide*.

column
specifies the name of the column or columns that are read from the input table and passed to the SAS_EP function.

input-table-schema
specifies the name of the schema where the input table exists.

input-table-name
specifies the input table that is used by the SAS_EP function.

SCATTER BY *column*<, …*column-n*> | SCATTER RANDOMLY
specifies how the input data is distributed.

ORDER BY *column*<, …*column-n*>
specifies how the input data is sorted within its distribution.

model-table-name
specifies the name of the model table where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files were published with the %INDGP_CREATE_MODELTABLE macro.

Requirement The table name that you specify for this function must be the same table name that is used in the %INDGP_CREATE_MODELTABLE macro. For more information, see "%INDGP_CREATE_MODELTABLE Macro Syntax" on page 67.

model-name
specifies the name of the model.

Greenplum Scoring Files

When using the SAS Embedded Process, the %INDGP_PUBLISH_MODEL macro produces two scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the SAS_EP function.
- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published. This file is used by the SAS_EP function.

These files are published to the model table that you specify in the %INDGP_PUBLISH_MODEL macro. See Appendix 1, “Scoring File Examples,” on page 285 for an example of each of these files.

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

Although you cannot view the scoring files directly, there are two ways to see the models whose files are created:

- Run this query from psql:

  ```
  select modelname from <schema.>sas-model-table;
  ```

- Run a PROC SQL query from SAS.

  ```
  proc sql;
   connect to greenplm (user=userid password=xxxx dsn=mydsn schema=myschema);
    select * from connection to greenplm
   (select modelname from schema.sas_model_table);
   disconnect from greenplm;
  quit;
  ```

Starting and Stopping the SAS Embedded Process

The SAS Embedded Process starts when a query is submitted using the SAS_EP function. It continues to run until it is manually stopped or the database is shut down.

Manually starting and stopping the SAS Embedded Process has implications for all scoring model publishers, requires superuser permissions, and must be run from the Greenplum master node. It should not be done without consulting your database administrator. For more information, see “Controlling the SAS Embedded Process” in Chapter 4 of *SAS In-Database Products: Administrator's Guide*.

SAS Embedded Process Troubleshooting Tips

If you have problems running scoring models with the SAS Embedded Process, these are the most likely areas where a problem could occur:

- Greenplum Partner Connector (GPPC) version 1.2 must be installed. You can verify that GPPC is installed by running this command.

  ```
  ls $GPHOME/lib/*gppc*
  ```

- When you use the SAS_EP function in an SQL query, the schema name is either SASLIB or a schema that was specified when the SAS_EP function was registered. SASLIB is the default schema name for the INDCONN macro variable when the %INDGP_PUBLISH_COMPILEUDF_EP macro is run to create the SAS_EP function. For more information, see “Running the %INDGP_PUBLISH_COMPILEUDF_EP Macro” in Chapter 4 of *SAS In-Database Products: Administrator's Guide*.

- When you refer to the model table in an SQL query, the schema name is either the user ID or a schema that was specified when the model table was created. The user ID is the default schema name for the INDCONN macro variable when the %INDGP_PUBLISH_MODELTABLE macro is run to create the model table. For more information, see “Creating a Model Table” on page 66.
- When you use the SAS_EP function, you must specify the schema where the SAS_EP function was registered.
- $GPHOME can be referenced by a symbolic link or the explicit path. When you update the Greenplum version, it is safer to always use the explicit path. Here is an example.

```
/usr/local/greenplum-db -> /usr/local/greenplum-db-4.2.2.0
```

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. Populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS and submit one of the following commands in the Program Editor or Enhanced Editor:

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

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

   For more information, see the “INDCONN Macro Variable” on page 71.

4. Run the %INDGP_PUBLISH_MODEL macro. For more information, see “%INDGP_PUBLISH_MODEL Macro Syntax” on page 73.

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

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Greenplum. You must specify user, password, either the DSN name or the server and database name. The schema name is optional. 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*<'>
<SCHEMA=<'>*schemaname*<'>> <PORT=<'>*port-number*<'>>

USER=<'>*username*<'> PASSWORD=<'>*password*<'> SERVER=<'>*servername*<'>
DATABASE=<'>*databasename*<'> <SCHEMA=<'>*schemaname*<'>>
<PORT=<'>*port-number*<'>>

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 the user name 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 he password 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 the DSN 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 the server name 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 the database name 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.

PORT=<'>*port-number*<'>
specifies the psql port number.

Default 5432

Requirement The server-side installer uses psql, and psql default port is 5432. If you want to use another port, you must have the UNIX or database administrator change the psql port.

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*
<, MECHANISM=STATIC | EP>
<, MODELTABLE=*model-table-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 19

MODELNAME=*name*

specifies the name that is prepended to each output function to ensure that each scoring function name is unique in 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 61.

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 27 and "Scoring Function Names" on page 61.

MECHANISM=STATIC | EP

specifies whether scoring functions or scoring files are created. MECHANISM= can have one of the following values:

STATIC

specifies that scoring functions are created.

These scoring functions are used in an SQL query to run the scoring model.

See "Using Scoring Functions to Run Scoring Models" on page 60

EP

specifies that scoring files are created.

These scoring files are used by the SAS Embedded Process to run the scoring model. A single entry in the model table is inserted for each new model. The entry contains both the score.sas and score.xml files in separate columns. The scoring process includes reading these entries from the table and transferring them to each instance of the SAS Embedded Process for execution.

Requirement	If you specify MECHANISM=EP, you must also specify the MODELTABLE= argument.
Note	The SAS Embedded Process might require a later release of Greenplum than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.
See	"Using the SAS Embedded Process to Run Scoring Models" on page 65

Default STATIC

MODELTABLE=*model-table-name*

specifies the name of the model table where the scoring files are published.

Default	sas_model_table
Restriction	This argument is valid only when using the SAS Embedded Process.
Requirement	The name of the model table must be the same as the name specified in the %INDGP_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in "%INDGP_CREATE_MODELTABLE Macro Syntax" on page 67.

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.
	If you use scoring functions to run scoring models, the maximum number of output variables is 128. If you use the SAS Embedded Process, the maximum is 1660.
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 71.

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 61.

Tip	This argument is useful when testing your scoring models.
See	"Special Characters in Directory Names" on page 19

Greenplum Permissions

Scoring Function 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.

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

For more information about specific permissions, see "Greenplum Permissions" in Chapter 4 of *SAS In-Database Products: Administrator's Guide*.

SAS Embedded Process Permissions

In addition to Greenplum superuser permissions, you must have CREATE TABLE permission to create a model table when using the SAS Embedded Process.

For more information about specific permissions, see "Greenplum Permissions" in Chapter 4 of *SAS In-Database Products: Administrator's Guide*.

Chapter 7

SAS Scoring Accelerator for Hadoop

Overview of Running Scoring Models in Hadoop

The integration of the SAS Embedded Process and Hadoop allows scoring code to be run directly on Hadoop using the SAS Embedded Process.

The SAS Embedded Process is a SAS server process that runs inside Hadoop to read and write data. A model publishing macro creates scoring files and stores them in a Hadoop Distributed File System (HDFS) directory. These scoring files are then used by a Hadoop MapReduce function to run the scoring model.

The SAS Scoring Accelerator for Hadoop requires a specific version of Hadoop. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Running Scoring Models in Hadoop

To run a scoring model in Hadoop, follow these steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Start SAS.
3. [Optional] Create a metadata file for the input data file.

The metadata file has the extension .sashdmd and must be stored in the HDFS. Use PROC HDMD to generate the metadata file.

Note: You do not have to create a metadata file for the input data file if the data file is created with a Hadoop LIBNAME statement that contains the HDFS_DATADIR= and HDFS_METADIR options. In this instance, metadata files are automatically generated.

Note: SAS/ACCESS requires Hadoop data to be in Hadoop standard UTF-8 format. If you are using DBCS encoding, you must extract the value of the character length in the engine-generated SASHDMD metadata file and multiply it by the number of bytes of a single character in order to create the correct byte length for the record.

For more information, see “Creating a Metadata File for the Input Data File” on page 88 and PROC HDMD in *SAS/ACCESS for Relational Databases: Reference*.

4. Connect to the HDFS using this command.

```
%let indconn=hdfs_server=myhdfsserver hdfs_port=8020 user=myuserid;
```

For more information, see “INDCONN Macro Variable” on page 79.

5. Run the %INDHD_PUBLISH_MODEL macro.

The %INDHD_PUBLISH_MODEL macro uses some of the files that the SAS Enterprise Miner Score Code Export node creates:

- the scoring model program (score.sas file)
- the properties file (score.xml file)
- a format catalog (if the training data includes SAS user-defined formats)

The %INDHD_PUBLISH_MODEL macro translates the score.sas file into a DS2 program and, if needed, generates an XML file for the user-defined formats. Then all model files (the SAS program, the DS2 program, the score.xml file, and the XML file for user-defined formats) are copied to the HDFS.

For more information, see “%INDHD_PUBLISH_MODEL Syntax” on page 81.

6. Connect to the MapReduce JobTracker using this command.

```
%let indconn=hdfs_server=myhdfsserver hdfs_port=hdfsport
   mapred_server=mapred-server-name mapred_port=mapred-port-number;
```

For more information, see “INDCONN Macro Variable” on page 79.

7. Run the %INDHD_RUN_MODEL macro.

The %INDHD_PUBLISH_MODEL macro publishes the model to Hadoop, making the model available to run against data that is stored in the HDFS.

The %INDHD_RUN_MODEL macro starts a MapReduce job that uses the files generated by the %INDHD_PUBLISH_MODEL to execute the DS2 program. The MapReduce job stores the DS2 program output in the HDFS location that is specified by either the OUTPUTDATADIR= argument or by the <outputDir> element in the HDMD file.

For more information, see “%INDHD_RUN_MODEL Syntax” on page 83.

8. Submit an SQL query against the output file.

For more information, see “Scoring Output” on page 90.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to the Hadoop HDFS and MapReduce JobTracker. You must assign the INDCONN macro variable before you run the %INDHD_PUBLISH_MODEL and the %INDHD_RUN_MODEL macros.

Note: The INDCONN macro variable can be set once and used for both macros.

- When you assign the INDCONN macro variable before you run the %INDHD_PUBLISH_MODEL macro, specify the server and port in order to access the machine on which you have installed the Hadoop HDFS. The user ID and password are optional. Here is the syntax.

  HDFS_SERVER=*hdfs-server-name* HDFS_PORT=*hdfs-port-number* <USER=*user*> <PASSWORD=*password*><HADOOP_CFG=*configuration-file*>

- When you assign the INDCONN macro variable before you run the %INDHD_RUN_MODEL macro, specify either the Hadoop configuration file or the HDFS and MapReduce server and the HDFS and MapReduce port in order to access the machine on which you have installed the MapReduce JobTracker. The user ID and password are optional. The INDCONN macro variable for the %INDHD_RUN_MODEL macro has one of these formats:

  HDFS_SERVER=*hdfs-server-name* HDFS_PORT=*hdfs-port-number* MAPRED_SERVER=*mapred-server-name* MAPRED_PORT=*mapred-port-number* <USER=*user* > <PASSWORD=*password*> <HADOOP_CFG=*configuration-file*>

  HADOOP_CFG=*configuration-file* <USER=*user* > <PASSWORD=*password*>

Arguments

HDFS_SERVER=*hdfs-server-name*
specifies the HDFS server name or the IP address of the server host.

Restriction	This argument is for use with the Cloudera 4.x and Hortonworks 1.3.x distributions. If you have Cloudera 5.x or Hortonworks 2.x, you must use the HADOOP_CFG argument.
Interaction	It is recommended that you use the HADOOP_CFG argument instead of the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments. The configuration file that you specify with the HADOOP_CFG argument overrides what you specify in the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments.
See	"HADOOP_CFG=*configuration-file*" on page 80

HDFS_PORT=*hdfs-port-number*
specifies the HDFS name node process port number.

Restriction	This argument is for use with the Cloudera 4.x and Hortonworks 1.3.x distributions. If you have Cloudera 5.x or Hortonworks 2.x, you must use the HADOOP_CFG argument.
Interaction	It is recommended that you use the HADOOP_CFG argument instead of the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and

MAPRED_PORT arguments. The configuration file that you specify with the HADOOP_CFG argument overrides what you specify in the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments.

See "HADOOP_CFG=*configuration-file*" on page 80

MAPRED_SERVER=*mapred-server-name*
specifies the MapReduce JobTracker server name or the IP address of the server.

Restriction This argument is for use with the Cloudera 4.x and Hortonworks 1.3.x distributions. If you have Cloudera 5.x or Hortonworks 2.x, you must use the HADOOP_CFG argument.

Interaction It is recommended that you use the HADOOP_CFG argument instead of the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments. The configuration file that you specify with the HADOOP_CFG argument overrides what you specify in the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments.

Note This argument is required only when you run the %INDHD_RUN_MODEL macro.

See "HADOOP_CFG=*configuration-file*" on page 80

MAPRED_PORT=*mapred-port-number*
specifies the MapReduce JobTracker port number.

Restriction This argument is for use with the Cloudera 4.x and Hortonworks 1.3.x distributions. If you have Cloudera 5.x or Hortonworks 2.x, you must use the HADOOP_CFG argument.

Interaction It is recommended that you use the HADOOP_CFG argument instead of the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments. The configuration file that you specify with the HADOOP_CFG argument overrides what you specify in the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments.

Note This argument is required only when you run the %INDHD_RUN_MODEL macro.

See "HADOOP_CFG=*configuration-file*" on page 80

USER=*username*
specifies the Hadoop user name (also called the user ID) that is used to connect to the HDFS.

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

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

HADOOP_CFG=*configuration-file*
specifies the location of the Hadoop configuration file that is used with the %INDHD_PUBLISH_MODEL and the %INDHD_RUN_MODEL macros.

Restriction	If you have Cloudera 5.x or Hortonworks 2.x, you must use the HADOOP_CFG argument. You cannot use the HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments with Cloudera 5.x or Hortonworks 2.x.
Interaction	The configuration file that you specify with the HADOOP_CFG= argument overrides what you specify in the INDCONN HDFS_SERVER, HDFS_PORT, MAPRED_SERVER, and MAPRED_PORT arguments.
Note	This argument is required only when you run the %INDHD_RUN_MODEL macro on a YARN/MapReduce 2 Hadoop cluster.
See	"HDFS_SERVER=*hdfs-server-name*" on page 79
	"HDFS_PORT=*hdfs-port-number*" on page 79
	"MAPRED_SERVER=*mapred-server-name*" on page 80
	"MAPRED_PORT=*mapred-port-number*" on page 80

%INDHD_PUBLISH_MODEL Syntax

%INDHD_PUBLISH_MODEL
(DIR=*input-directory-path*
, MODELNAME=*name*
, MODELDIR=*hdfs-directory-path*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, FMTCAT=*format-catalog-filename* | *libref.format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
<, TRACE=YES | NO>
);

Arguments

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

This is the directory that the SAS Enterprise Miner Score Code Export node creates.

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 19

MODELNAME=*name*
specifies the model name. This name is used to create the HDFS directory, in the directory path specified by the MODELDIR option. The model files (the SAS

program, the DS2 program, the score.xml file, and the XML file for user-defined formats) are placed in the HDFS directory.

Requirement The model name must be a valid SAS name. There is no limit on the number of characters in the model name. For more information about valid SAS names, see the topic on rules for words and names in *SAS Language Reference: Concepts*.

MODELDIR=*hdfs-directory-path*
specifies the base HDFS path where the scoring model directory is located.

Restriction You must use a fully qualified pathname.

See "Special Characters in Directory Names" on page 19

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.

Requirement The scoring model program file must be located in the DIR directory.

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

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

Requirement The properties XML file must be located in the DIR directory.

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.

FMTCAT=*format-catalog-filename* | *libref.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*.

Note The format catalog is stored locally and is copied to the HDFS to the same directory where the metadata file is stored.

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

CREATE
creates a new set of model files.

Tip If the model files have been previously defined and you specify ACTION=CREATE, an error occurs.

REPLACE
overwrites the current set of model files.

DROP
deletes the current set of model files.

Default CREATE

Note If the current files do not exist and you specify REPLACE or DROP, an error occurs.

TRACE=YES | NO
specifies whether debug messages are displayed.

Default NO

%INDHD_RUN_MODEL Syntax

%INDHD_RUN_MODEL
(INMETANAME=*input-filename*.SASHDMD
, SCOREPGM=*model_score_program_ds2_file*
<, OUTDATADIR=*hdfs-directory-path*>
<, OUTMETADIR=*hdfs-directory-path*>
<, INFILETYPE=*type*>
<, INPUTFILE=*input-file-name*>
<, OUTFILEDELMITER=*file-delimiter*>
<, OUTTEXTQUALIFIER=*text-qualifier*>
<, OUTFILETYPE=*output-file-type*>
<, OUTRECORDFORMAT=*output-record-format*>
<, FORMATFILE=*user-defined-format-filename*>
<, FORCEOVERWRITE=TRUE | FALSE>
<, KEEP=*variable-keep-list*>
<, KEEPFILENAME=*keep-list-configuration-filename*>
<, TRACE=YES | NO>
);

Arguments

INMETANAME=*input-filename*.SASHDMD
specifies the HDFS full path of the input metadata file (.sashdmd file).

Requirement	The metadata file must already exist or must be generated with PROC HDMD before running the %INDHD_RUN_MODEL macro. You do not have to create a metadata file for the input data file if the data file is created with a Hadoop LIBNAME statement that contains the HDFS_DATADIR= and HDFS_METADIR options. In this instance, metadata files are automatically generated.
Interaction	This file is read by the MapReduce job.
See	"Creating a Metadata File for the Input Data File" on page 88
	PROC HDMD in *SAS/ACCESS for Relational Databases: Reference*

SCOREPGM=*model_score_program_ds2_file*
specifies the name of the scoring model program file that is executed by the SAS Embedded Process.

OUTDATADIR=*hdfs-directory-path*
specifies the name of the HDFS directory where the MapReduce job stores the output files.

Interaction	The *hdfs-directory-path* overrides what is specified in the <outputDir> element in the input metadata file (.sashdmd file).

OUTMETADIR=*hdfs-directory-path*
specifies the name of the HDFS directory where the MapReduce job stores the output file metadata.

Interaction	The *hdfs-directory-path* overrides what is specified in the <metaDir> element in the input file metadata (.sashdmd file).

INFILETYPE=*type*
specifies the type of input file. *type* can be one of the following:

DELIMITED
specifies a delimited file.

Note	This type maps to the `com.sas.access.hadoop.ep.delimited.DelimitedInputFormat` input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

CUSTOM
specifies a custom file.

Note	This type maps to the `com.sas.access.hadoop.ep.custom.CustomFileInputFormat` input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

CUSTOM_SEQUENCE
specifies a custom sequence file.

Note This type maps to the **com.sas.access.hadoop.ep.custom.CustomSequenceFileInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

SEQUENCE
specifies a sequence file.

Note This type maps to the **com.sas.access.hadoop.ep.sequence.EpSequenceFileInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

BINARY
specifies a fixed record length file.

Alias FIXED

Note This type maps to the**com.sas.access.hadoop.ep.binar.FixedRecLenBinaryInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

XML
specifies an XML file.

Note This type maps to the **com.sas.access.hadoop.ep.xml.XmlInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

JSON
specifies a JSON file.

Note This type maps to the **com.sas.access.hadoop.ep.json.JsonInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

SPD
specifies an SPD file type.

Note This type maps to the **com.sas.hadoop.ep.spd.EPSPDInputFormat** input format in the <epInputFormat> element in the input file metadata (.sashdmd file).

Interaction The *type* overrides what is specified in the <epInputFormat> element in the input file metadata (.sashdmd file).

Note If this option is specified, the %INDHD_RUN_MODEL macro automatically matches the type with the correct input format Java class for the SAS Embedded Process. See each *type* for the mapping that is performed.

INPUTFILE=*input-filename*
specifies an HDFS fully qualified input filename. This file is read by the MapReduce job.

Interaction	The *input-filename* overrides what is specified in the <inputDir> element in the input file metadata (.sashdmd file).

OUTFILEDELIMTER=*file-delimiter*
specifies the delimiter for variables (fields) in the output file. Here is how you can specify the delimiter.

- ','
- '\t'
- ^A
- ^Z
- '09'x
- 32

Default	^A
Range	You can specify only a single character between the Unicode range of U+0001 to U+007F.
Restriction	The value of this option cannot be the same character as for OUTTEXTQUALIFIER and cannot be a newline ('0a'x).
Requirement	This option is valid only for DELIMITED. Other formats do not use it.
Note	Valid values are 0–127, a comma (",”), or "\t".

OUTTEXTQUALIFIER=*text-qualifier*
specifies the text qualifier to be used in the output data file.

Default	none
Range	You can specify only a single character between the Unicode range of U+0001 to U+007F.
Restriction	The value of this option cannot be the same character as for OUTFILEDELIMTER and cannot be a newline ('0a'x).
Requirement	This option is valid only for DELIMITED. Other formats do not use it.

OUTFILETYPE=*output-file-type*
specifies the output file type. *output-file-type* can be one of the following values:

DELIMITED
specifies a delimited file.

BINARY
specifies a fixed record length file.

Alias	FIXED

SPD
specifies an SPD file.

Default — If the input file type is fixed, the output file type is fixed. Otherwise, it is delimited.

OUTRECORDFORMAT=*output-record-format*
specifies the output record format. *output-record-format* can be one of the following values:

DELIMITED
specifies a delimited format.

FIXED
specifies a fixed record length format.

Default — DELIMITED

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

Interaction — This name is the same one that you specified in the %INDHD_PUBLISH_MODEL macro's FMTCAT argument.

See — "FMTCAT=*format-catalog-filename* | *libref.format-catalog-filename*" on page 82

FORCEOVERWRITE=TRUE | FALSE
specifies whether the output directory is deleted before the MapReduce job is executed.

Default — FALSE

KEEP=*variable-keep-list*
specifies a list of variables that the SAS score program retains.

Restriction — KEEP and KEEPFILENAME are mutually exclusive.

Requirement — The list of variables must be separated by spaces and should not be enclosed by single or double quotation marks.

KEEPFILENAME=*keep-list-configuration-filename*
specifies the name of an XML configuration file that contains the list of variables that are passed to the SAS score program.

The keep list configuration file should have the following format:

```
<configuration>
   <property>
      <name>sas.ep.ds2.keep.list</name>
      <value>var1 var2 var3 var4... varn</value>
   </property>
</configuration>
```

Restriction — KEEP and KEEPFILENAME are mutually exclusive.

Requirement — You must specify the full path.

TRACE=YES | NO
specifies whether debug messages are displayed.

Default NO

Creating a Metadata File for the Input Data File

Before running the %INDHD_RUN_MODEL macro, the metadata file for the input data file must be present in an HDFS location.

There are three ways that the metadata file can be present depending on where the file exists.

- The file is not in Hive.

  No metadata exists. You must use PROC HDMD to create the metadata file. Here is an example:

```
/****************************************************************
* Assign a libname to the Hadoop Engine that specifies the
* locations where data, metadata,and temporary data will be stored.
*****************************************************************/
libname hdlib hadoop
   server=hdoop
   user=hadoop_user1
   HDFS_METADIR="/metadata"
   HDFS_TEMPDIR="/tmp";

/****************************************************************
* Create a metadata file for input file defined under data_file.
* The metadata file name is defined in the NAME= option and is
* stored under the HDFS folder defined in HDFS_METADIR.
****************************************************************/
proc hdmd
   name=hdlib.pilotmd
   format=delimited
   sep=','
   data_file='pilot.dat';

      column EmployeeID char(6);
      column FirstName  char(13);
      column LastName   char(15);
      column JobCode    char(7);
      column Salary     char(6);
      column Category   char(3);
run;
```

- The file is in a Hive library.

  Metadata is associated with the file in Hive, but the metadata is not in HDMD format. You must generate an HDMD file for it. Here is an example:

```
/*********************************************************
* Assigns a libname to the Hadooop Engine that specifies the locations
* where data, metadata and temporary data will be stored.
*******************************************************************/
libname gridlib hadoop server="cdh123"
   user="hadoop"
```

```
   HDFS_TEMPDIR="/data/temp"
   HDFS_DATADIR="/data/dlm/data"
   HDFS_METADIR="/data/dlm/meta"
   DBCREATE_TABLE_EXTERNAL=NO;

/*********************************************************
* Assigns a libname to the Hadooop Engine that specifies
* that the data and metdata will be in Hive
*********************************************************/
libname hive hadoop server="cdh123"
   user=hadoop
   database=hpsumm
   subprotocol=hive2;

/*************************************************************
* Creates an HDMD file from Hive table 'stthive' and stores
* it under the directory specified in HDFS_METADIR option of
* the 'gridlib' libname.
*************************************************************/
proc hdmd
   from=hive.stthive
   name=gridlib.sttout;
run;
```

- The file is created with the ACCESS Hadoop engine.

  When a file is created with a Hadoop LIBNAME statement that contains the HDFS_DATADIR= and HDFS_METADIR options, the HDMD file is automatically generated. Here is an example:

```
/*****************************************************
* Assigns a libname to the Hadooop Engine that specifies
* the locations where data, metadata and temporary data
* will be stored.
*********************************************************/
libname gridlib hadoop server="cdh123"
   user="hadoop"

   HDFS_TEMPDIR="/data/temp"
   HDFS_DATADIR="/data/dlm/data"
   HDFS_METADIR="/data/dlm/meta"
   DBCREATE_TABLE_EXTERNAL=NO;

/*****************************************
* Assigns a libname to a local SAS file
*****************************************/
libname mydata "C:/tmp/myfiles"

/**************************************************************
* Creates a Hadoop file from mydata.intrid along with its HDMD
* file and stores under what was specified on HDFS_DATADIR of
* 'gridlib'.
**************************************************************/
proc sql;
create table hdlib.flights98
        as select * from sasflt.flt98;
quit;
```

The metadata file has the extension .sashdmd.

For more information, see PROC HDMD in *SAS/ACCESS for Relational Databases: Reference*.

Scoring Output

Scoring Output File

When you run the %INDHD_RUN_MODEL macro, a delimited or fixed output file is created.

In addition, a metadata file is created for the output file.

You can specify which columns are written to the output file by using the KEEP or KEEPFILENAME argument of the %INDHD_RUN_MODEL macro. For more information about the KEEP and KEEPFILENAME arguments, see "%INDHD_RUN_MODEL Syntax" on page 83.

Querying and Viewing the Scoring Output File

Note: The %INDHD_RUN_MODEL macro does not generate a SampleSQL.txt file.

To view the output data in a SAS session, you can use PROC PRINT as long as you have a LIBNAME statement to access the Hadoop output file. Here is an example that prints the first ten rows of the output table.

```
/* Hadoop configuration file */

%let INDCONN=%str(HDFS_SERVER=hd.mycompany.com
    HDFS_PORT=8120
    MAPRED_SERVER=hd.mycompany.com
    MAPRED_PORT=8021
    USER=myuserid
    PASSWORD=mypwd);

/* libname pointing to Hadoop */
libname gridlib hadoop user=myuserid
   pw=mypwd
   server="hd.mycompany.com"
   HDFS_TEMPDIR="/user/hdmd/temp"
   HDFS_DATADIR="/user/hdmd/data"
   HDFS_METADIR="/user/hdmd/meta";

/* Delete HDMD file */
proc delete data=gridlib.peopleseq; run;

/* Create HDMD file */
proc hdmd NAME=GRIDLIB.PEOPLESEQ
   FILE_FORMAT=DELIMITED
   SEP=tab
   FILE_TYPE=custom_sequence
   INPUT_CLASS='com.sas.hadoop.ep.inputformat.sequence.
```

```
      PeopleCustomSequenceInputFormat'
   DATA_FILE='people.seq';

   COLUMN name   varchar(20);
   COLUMN sex    varchar(1);
   COLUMN age    int;
   column height double;
   column weight double;
run;

/*===================================================================
/* Start MR Job using the run model for Hadoop macro
*==================================================================*/
%indhd_run_model(infiletype=custom_sequence
    , inmetaname=/user/hdmd/meta/peopleseq.sashdmd
    , outdatadir=/user/hdmd/output/peopletxt.out
    , outmetadir=/user/hdmd/meta/peopletxt.sashdmd
    , scorepgm=/user/hdmd/ds2/inout.ds2
    , forceoverwrite=true
    , trace=no);

/* Print output file */
proc print data=gridlib.peopletxt(obs=10); run;
```

The columns in the output file are available to use in any SQL query expression.

```
select * from gridlib.peopletxt;

select em_classification from gridlib.peopletxt;
```

Hadoop Permissions

You must have permissions for the domains that you specify in the INDCONN macro variable when you execute the publish and run macros.

You also need Write permission when writing files to the MODELDIR directory in the %INDSP_RUN_MODEL macro. Without these permissions, the publishing of the scoring model fails.

To obtain these permissions, contact your Hadoop administrator.

Chapter 8

SAS Scoring Accelerator for Netezza

Overview of Running Scoring Models in Netezza

There are two ways to run scoring models in Netezza.

- You can create scoring functions for each EM_ output variable. The model publishing macro creates the scoring functions that are published as Netezza user-defined functions. These functions can then be used in any SQL query. For more information, see “Using Scoring Functions to Run Scoring Models” on page 94.
- Starting with the SAS 9.4 release, you can also use the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs inside Netezza to read and write data. The model publishing macro creates scoring files that are then used in a stored procedure to run the scoring model. For more information, see “Using the SAS Embedded Process to Run Scoring Models” on page 98.

The SAS Scoring Accelerator for Netezza requires a certain version of the Netezza client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Using Scoring Functions to Run Scoring Models

How to Run a Scoring Model Using Scoring Functions

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 27.

To run the scoring model using scoring functions, follow these steps:

1. Run the %INDNZ_PUBLISH_MODEL macro.

   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 and produces a set of .c, .cpp, and .h files. These .c, .cpp, 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.
   - processes the format catalog and creates an .h file with C structures if a format catalog is available. This file is 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.

   For more information, see "Running the %INDNZ_PUBLISH_MODEL Macro" on page 104. For more information about the scoring functions, see "Scoring Function Names" on page 94 and "Viewing the Scoring Functions" on page 95.

2. Use the scoring functions in any SQL query.

   For more information, see "Using Scoring Functions to Run Scoring Models" on page 94.

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 104.

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 27.

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: The 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.

Viewing the Scoring Functions

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 look at the SAS log that is created when the %INDNZ_PUBLISH_MODEL macro was run. A message is printed to the SAS log that indicates whether a scoring function is successfully or not successfully created or replaced.

- 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 SQL code that can be used to run your score code inside Netezza. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

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*. *n* is the number of rows in the table.

Note: 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 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 ;
```

Using Scoring Functions to Run a Scoring Model

The scoring functions are available to use in any SQL expression in the same way that Netezza built-in functions are used.

After the scoring functions are created, 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. The SampleSQL.txt file shown in "Viewing the Scoring Functions" on page 95 was modified to create the SELECT statement in this example.

```
select id, allmush1_em_classification
(
"BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC")
  as "EM_CLASSIFICATION",
 from allmush1_intab ;
```

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

Using the SAS Embedded Process to Run Scoring Models

How to Run a Scoring Model with the SAS Embedded Process

The integration of the SAS Embedded Process and Netezza allows scoring code to run directly using the SAS Embedded Process on Netezza.

Note: The SAS Embedded Process might require a later release of Netezza than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.

To run the scoring model using the SAS Embedded Process, follow these steps.

1. Create a table to hold the scoring files.

   The %INDNZ_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

   For more information, see "Creating a Model Table" on page 98.

2. Run the %INDNZ_PUBLISH_MODEL to create the scoring files.

   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:

   - translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process
   - 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 Netezza to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDNZ_CREATE_MODELTABLE macro.

   For more information, see "Running the %INDNZ_PUBLISH_MODEL Macro" on page 104 and "Netezza Scoring Files" on page 103.

3. Execute the SAS_EP stored procedure to run the scoring model.

   For more information, see "SAS_EP Stored Procedure" on page 100.

Creating a Model Table

Overview

When using the SAS Embedded Process to publish a scoring model in Netezza, you must create a table to hold the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files. You must run the

%INDNZ_CREATE_MODELTABLE macro to create the table before you run the %INDNZ_PUBLISH_MODEL macro.

You have to create the table only one time to hold a model's scoring files.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *model-name.model-table-name*.

Column Name	Description	Specification
ModelName	contains the name of the model.	NVARCHAR(128) NOT NULL
ModelSequence	contains the sequence number for the block of ModelDS2 data. The ModelDS2 files is divided into blocks with each block in one row for one model.	INTEGER NOT NULL
ModelDS2	contains the sasscore_*modelname*.ds2 file.	NVARCHAR (8000)
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file.	NVARCHAR (8000)
ModelUUID*	contains the UUID of the source model.	VARCHAR (36)
Notes*	contains additional information that describes the source model.	VARCHAR (512)

* This column is for use by SAS Model Manager. If you have a model table that was created prior to SAS 9.4 and you want this column in your model table, you must run the %INDNZ_CREATE_MODELTABLEmacro to re-create your model table.

%INDNZ_CREATE_MODELTABLE Run Process

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

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 105.

2. Run the %INDNZ_CREATE_MODELTABLE macro.

   For more information, see "%INDNZ_CREATE_MODELTABLE Macro Syntax" on page 100.

%INDNZ_CREATE_MODELTABLE Macro Syntax

%INDNZ_CREATE_MODELTABLE

(<DATABASE=*database-name*>
<, MODELTABLE=*model-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

DATABASE=*database-name*
specifies the name of a Netezza database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default	The database specified in the INDCONN macro variable or your current database.

MODELTABLE=*model-table-name*
specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default	sas_model_table
Requirement	The maximum table name length is 128 characters, and it must be a valid Netezza table name.
Interaction	The table name that you specify for this macro must be the same table name that is used in the %INDNZ_PUBLISH_MODEL macro.
See	"%INDNZ_PUBLISH_MODEL Macro Syntax" on page 106

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

CREATE
creates a new table.

Tip	If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

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

Tip	If you specify ACTION = REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP
causes all models in this table to be dropped.

Default	CREATE

SAS_EP Stored Procedure

Overview of the SAS_EP Stored Procedure

The SAS_EP stored procedure is the interface for running the scoring model inside Netezza with the SAS Embedded Process. The SAS_EP stored procedure uses the files that are stored in the model table. The stored procedure parameters enable you to control the name and location of the output table, how much data is returned, and how it is returned.

The SAS_EP stored procedure is installed in the NZRC directory. To run the stored procedure, you must have the following permissions:

- User name must be created with the IBM Netezza Analytics utility
- Database must be created with the IBM Netezza Analytics utility

For more information, see “Netezza Permissions” on page 111.

Running the SAS_EP Stored Procedure

You can run the SAS_EP stored procedure using explicit pass-through and PROC SQL or you can use NZSQL.

Note: Before running the SAS_EP stored procedure, you must create the model table with the %INDNZ_CREATE_MODELTABLE macro and then you must publish the files to the model table with the %INDNZ_PUBLISH_MODEL macro.

Here is an example using PROC SQL.

```
proc sql;
   connect to netezza (user=userid password=xxxx server=myserver);
   execute
      (CALL NZRC..SAS_EP
         ('
           model_name=intr_reg,
           input_table=intr_reg_20m,
           input_columns= id count dif_srvr flag hot sam_srat service srv_cnt,
           output_table=intr_reg_out,
           journal_table=intr_reg_out_jnl,
           ds2_keep=id EM_CLASSIFICATION EM_EVENTPROBABILITY EM_PROBABILITY
          ');
      ) by netezza;
   disconnect from netezza;
quit;
```

For more information about the stored procedure parameters, see “SAS_EP Stored Procedure Syntax” on page 101.

SAS_EP Stored Procedure Syntax

NZRC..SAS_EP
('
MODEL_NAME=*model-name*,
INPUT_TABLE=<<">*input-database*<">..><">*input-table*<">,
OUTPUT_TABLE=<< ">*output-database*<">..><">*output-table*<">,
<MODEL_TABLE=<<">*model-table-database*<">..><">*model-table*<">,>
<INPUT_COLUMNS=<">*column*<"><… <">*column*<">>,>
<INPUT_WHERE=*where-clause*,>
<OUTPUT_DISTRIBUTION=<">*column*<"><… <">*column*<">>,>
<OUTPUT_TEMPORARY= YES | NO,>
<JOURNAL_TABLE=<<">*journal-database*<">.. ><">*journal-table*<">>
<PARTITION_BY=<">*column*<"><… <">*column*<">>>
<ORDER_BY=<">*column*<"><, … <">*column*<">>>
');

Parameters

MODEL_NAME=*model-name*
specifies the name of the model.

Requirement	There must be at least one row in the model table with this key in the ModelName column and with a non-null value in the ModelDS2 column.
Tip	The model name is case sensitive.

INPUT_TABLE=<<"><*input-database*<">..>><">*input-table*<">
specifies the name of the scoring model input table.

Requirement	If the input table name is case sensitive, you must use double quotation marks (for example, `"myinDB..myinTABLE"`).

OUTPUT_TABLE=<<">*output-database*<">.. ><">*output-table*<">
specifies the name of the scoring model output table.

Requirement	If the output table name is case sensitive, you must use double quotation marks (for example, `"myoutDB..myoutTABLE"`).

MODEL_TABLE=<<">*model-table-database*<">.. ><">*model-table*<">
specifies the name of the model table where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files were published with the %INDTD_CREATE_MODELTABLE macro.

Default	sas_model_table
Restriction	This argument is available only when using the SAS Embedded Process.
Requirements	If the model table name is case sensitive, you must use double quotation marks(for example, `"mymtDB..mymtTABLE"`).
	The name of the model table must be the same as the name specified in the %INDNZ_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in “%INDNZ_CREATE_MODELTABLE Macro Syntax” on page 100.

INPUT_COLUMNS=<">*column*<"><…<">*column*<">>
specifies one or more columns from the input table that are passed to the SAS Embedded Process.

Default	All columns
Requirement	If the column name is case sensitive, you must use double quotation marks (for example, `"Profit" "Sales" "Margin"`).

INPUT_WHERE=*where-clause*
specifies a valid WHERE clause that selects the rows from the input table.

OUTPUT_DISTRIBUTION=<">*column*<"><, …<">*column*<">>
specifies one or more columns that are the distribution key for the output table.

Default	Current database distribution
Requirement	If the column name is case sensitive, you must use double quotation marks (for example, `"Profit", "Sales", "Margin"`).

OUTPUT_TEMPORARY= YES | NO

specifies whether the output table is temporary.

Default	NO

JOURNAL_TABLE=<<">*journal-database*<">..><">*journal-table*<">

specifies the name of a table that the SAS_EP stored procedure creates. This table holds any journal messages and notes from the SAS journal facility that are produced when executing the store procedure.

Requirement	If the journal table name is case sensitive, you must use double quotation marks (for example, **`"myjnlDB..myjnlTABLE"`**).
Tip	If JOURNAL_TABLE is not specified, a journal is not created. If JOURNAL_TABLE is specified and a journal table exists, the journal is appended.

PARTITION_BY=<">*column*<"><, …<">*column*<">>

specifies one or more columns by which to partition the input.

Default	No partitioning occurs
Requirement	If the column name is case sensitive, you must use double quotation marks(for example, **`"Profit", "Sales", "Margin"`**).

ORDER_BY=<">*column*<"><, …<">*column*<">>

specifies one or more columns by which to order the input.

Default	No reordering occurs
Requirement	If the column name is case sensitive, you must use double quotation marks (for example, **`"Profit", "Sales", "Margin"`**).

Tips for Using the SAS_EP Stored Procedure

- No specific parameter order is required.
- Table names must be enclosed in double quotation marks if they are case sensitive. Otherwise, double quotation marks are optional.
- Tables can be qualified with a database name. If a table name is not qualified with a database name, the table name is resolved based on the default database for your session.
- All parameters are passed as strings to the SAS_EP stored procedure, so they must be enclosed in single quotation marks. To pass a single quotation mark as part of the SQL within a parameter, use two adjacent single quotation marks as shown in the following example:

```
'INPUT_WHERE=where name like ''%Jones%'''
```

Netezza Scoring Files

When using the SAS Embedded Process, the %INDNZ_PUBLISH_MODEL macro produces two scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the SAS_EP stored procedure.

- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published. This file is used by the SAS_EP stored procedure.

These files are published to the model table that you specify in the %INDNZ_PUBLISH_MODEL macro. See Appendix 1, “Scoring File Examples,” on page 285 for an example of each of these files.

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

Although you cannot view the scoring files directly, there are two ways to see the models whose files 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 table where the scoring files were published is **modtable** and the model name is **super**.

```
nzsql database username password

 select modelname, modelsequence from modtable where
   modelname like '%super%'
```

- 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 files is **super**.

```
proc sql noerrorstop;
  connect to netezza (user=username password=xxxx server=myserver);

select * from connection to netezza
      (select modelname, modelsequence
         from sasmodeltablename
            where modelname like '%super%');
disconnect 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*.

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. Populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the “INDCONN Macro Variable” on page 105.

4. If you use the SAS Embedded Process, run the %INDNZ_CREATE_MODELTABLE macro.

   For more information, see "Creating a Model Table" on page 98.

5. Run the %INDNZ_PUBLISH_MODEL macro.

   For more information, see "%INDNZ_PUBLISH_MODEL Macro Syntax" on page 106.

   Messages are written to the SAS log that indicate whether the scoring functions or files were successfully created.

Note: The %INDNZ_PUBLISH_JAZLIB macro and the %INDNZ_PUBLISH_COMPILEUDF macro (if needed) must be run before you can publish your scoring models using scoring functions. 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 the *SAS In-Database Products: Administrator's Guide*.

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 or the %INDNZ_CREATE_MODELTABLE macros are 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 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.

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

Requirement You must specify the DATABASE= argument if you use the SAS Embedded Process.

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 that others cannot access the user credentials.

%INDNZ_PUBLISH_MODEL Macro Syntax

%INDNZ_PUBLISH_MODEL
(DIR=*input-directory-path*, MODELNAME=*name*
<, MECHANISM=STATIC | EP>
<, MODELTABLE=*model-table-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 19

MODELNAME=*name*
specifies the name that is prepended to each output function to ensure that each scoring function or filename is unique on the Netezza database. If you are using the SAS Embedded Process, the model name is part of the scoring filenames.

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 94.
Requirement	If you use scoring functions, the model name must be a valid SAS name that is ten characters or fewer. If you use the SAS Embedded Process, the model name cannot exceed 128 characters. 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 27 and "Scoring Function Names" on page 94.

MECHANISM=STATIC | EP

specifies whether scoring functions or scoring files are created. MECHANISM= can have one of the following values:

STATIC

specifies that scoring functions are created.

These scoring functions are used in an SQL query to run the scoring model.

See	"Using Scoring Functions to Run Scoring Models" on page 94

EP

specifies that scoring files are created.

These scoring files are used by the SAS Embedded Process to run the scoring model. A single entry in the model table is inserted for each new model. The entry contains both the score.sas and score.xml in separate columns. The scoring process includes reading these files from the table and transferring them to each instance of the SAS Embedded Process for execution.

Requirement	If you specify MECHANISM=EP, you must also specify the MODELTABLE= argument.
Note	The SAS Embedded Process might require a later release of Netezza than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.
See	"Using the SAS Embedded Process to Run Scoring Models" on page 98

Default	STATIC

MODELTABLE=*model-table-name*

specifies the name of the model table where the scoring files are published.

Default	sas_model_table
Restriction	This argument is available only when using the SAS Embedded Process.
Requirement	The name of the model table must be the same as the name specified in the %INDNZ_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in "%INDNZ_CREATE_MODELTABLE Macro Syntax" on page 100.

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.
	If you use scoring functions to run scoring models, the maximum number of output variables is 128. If you use the SAS Embedded Process, the maximum is 1600. However, Netezza also has a maximum row size of 64K. If you have very large character columns, you might exceed the row limit before you exceed the maximum number of variables.
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 or the scoring files are published.

Requirement	You must specify the DATABASE= argument if you use the SAS Embedded Process.
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 104.
Tip	You can publish the scoring functions and formats or the scoring files 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
Restriction	This argument is ignored when MECHANISM=EP.
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
Restriction	This argument is ignored when MECHANISM=EP.

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 new functions or files.

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

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

Default	CREATE
Tip	If the function or file was published previously and you specify ACTION=CREATE, you receive warning messages that the function or file already exists and you are prompted to use REPLACE. If you specify ACTION=DROP and the function or file 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
Restrictions	This argument is ignored when MECHANISM=EP.
	The MODE= argument is supported for Netezza 6.0. The argument is ignored for previous versions of Netezza.
Tip	There are limited resources available in Netezza when you run in fenced mode. For example, there is a limit to the number of columns available.
See	"Modes of Operation" on page 110

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

Default	UPPERCASE
Restriction	This argument is ignored when MECHANISM=EP.
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 option 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	"Viewing the Scoring Functions" on page 95 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 "Viewing the Scoring Functions" on page 95.

Tip	This argument is useful when testing your scoring models.
See	"Special Characters in Directory Names" on page 19

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. 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 ignored when MECHANISM=EP.

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

Netezza Permissions

If you are using scoring functions to run your scoring model, 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.

If you are using the SAS Embedded Process to run your scoring model, your user ID and the database must be created with the IBM Netezza Analytics utility.

To obtain these permissions, contact your database administrator.

For more information about specific permissions, see “Netezza Permissions” in Chapter 6 of *SAS In-Database Products: Administrator's Guide*.

Chapter 9

SAS Scoring Accelerator for Oracle

Overview of Running Scoring Models

The integration of the SAS Embedded Process and Oracle allows scoring code to be run directly using the SAS Embedded Process on Oracle.

The SAS Embedded Process is a SAS server process that runs inside Oracle to read and write data. A model publishing macro creates scoring files and stores them in an Oracle table. These scoring files are then used by an Oracle function to run the scoring model.

The SAS Scoring Accelerator for Oracle requires a certain version of the Oracle client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Oracle Permissions

For Oracle, the following permissions are needed by the person who runs the scoring publishing macros.

- The person who runs the %INDOR_CREATE_MODELTABLE needs CREATE permission to create the model table.
- The person who runs the %INDOR_PUBLISH_MODEL macro needs INSERT permission to load data into the model table.

Without these permissions, the %INDOR_CREATE_MODELTABLE macro and the %INDOR_PUBLISH_MODEL macro fails. To obtain these permissions, contact your database administrator.

For more information about specific permissions, see “Oracle Permissions” in Chapter 7 of *SAS In-Database Products: Administrator's Guide*.

How to Run a Scoring Model in Oracle

To run a scoring model using the SAS Embedded Process, follow these steps.

1. Create a scoring model using SAS Enterprise Miner.
2. Start SAS and create a table to hold the scoring files.

   The %INDOR_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

   For more information, see “Creating a Model Table” on page 115.
3. Run the %INDOR_PUBLISH_MODEL macro to create the scoring files.

   The %INDOR_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 %INDOR_PUBLISH_MODEL macro performs the following tasks:

   - translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process.
   - 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 Oracle to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDOR_CREATE_MODELTABLE macro.

   For more information, see “Running the %INDOR_PUBLISH_MODEL Macro” on page 117 and “Oracle Scoring Files” on page 120.
4. Use the SASEPFUNC table function in the FROM clause in any SQL expression to run the scoring model.

   For more information, see “SASEPFUNC Table Function” on page 121.

Creating a Model Table

Overview

When publishing a model in Oracle on which the SAS Embedded Process is deployed, you must create a table to hold the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files. You must run the %INDOR_CREATE_MODELTABLE macro to create the table before you run the %INDOR_PUBLISH_MODEL macro.

You have to create the table only one time to hold a model's scoring files.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *schema-name.model-table-name*.

Column Name	Description	Specification
ModelName	contains the name of the model	VARCHAR(128)
ModelDS2	contains the sasscore_*modelname*.ds2 file	BLOB not null
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file	BLOB
ModelMetadata	reserved by SAS for future use	BLOB
ModelUUID*	contains the UUID of the source model	VARCHAR (36)
Notes*	contains additional information that describes the source model	VARCHAR (512)

* This column is for use by SAS Model Manager. If you have a model table that was created prior to SAS 9.4 and you want this column in your model table, you must run the %INDOR_CREATE_MODELTABLE macro to re-create your model table.

%INDOR_CREATE_MODELTABLE Run Process

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

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

   ```
   %let indconn = user=myuserid password=xxxx path=ortest;
   ```

   For more information, see the "INDCONN Macro Variable" on page 117.

2. Run the %INDOR_CREATE_MODELTABLE macro.

   For more information, see "%INDOR_CREATE_MODELTABLE Macro Syntax" on page 116.

%INDOR_CREATE_MODELTABLE Macro Syntax

%INDOR_CREATE_MODELTABLE
(<DATABASE=*database-name*>
<, MODELTABLE=*model-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

DATABASE=*database-name*
specifies the name of an Oracle database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default	The database specified in the INDCONN macro variable

MODELTABLE=*model-table-name*
specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default	sas_model_table
Requirement	The maximum table name length is 30 characters, and it must be a valid Oracle table name.
Interaction	The table name that you specify for this macro must be the same table name that is used in the %INDOR_PUBLISH_MODEL macro.
See	“%INDOR_PUBLISH_MODEL Macro Syntax” on page 118

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

CREATE
creates a new table.

Tip	If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

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

Tip	If you specify ACTION = REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP
causes all models in this table to be dropped.

Default	CREATE

Running the %INDOR_PUBLISH_MODEL Macro

%INDOR_PUBLISH_MODEL Run Process

To run the %INDOR_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.

   This directory will contain the score.sas file, the score.xml file, and, if needed, the format catalog.
3. Start SAS and submit this command in the Program Editor or Enhanced Editor:

   ```
   %let indconn = user=myuserid password=XXXX path=ortest;
   ```

   For more information, see the "INDCONN Macro Variable" on page 117.
4. Run the %INDOR_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 "%INDOR_PUBLISH_MODEL Macro Syntax" on page 118.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Oracle. You must specify user, password, and database information to access the machine on which Oracle is installed. You must assign the INDCONN macro variable before the %INDOR_PUBLISH_MODEL macro is invoked.

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

USER=*user* PASSWORD=*password* PATH=*path*

Arguments

USER=*user*
specifies the Oracle 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 Oracle user ID.

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

PATH=*path*
specifies the Oracle driver, node, and 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 %INDOR_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 that others cannot access the user credentials.

%INDOR_PUBLISH_MODEL Macro Syntax

%INDOR_PUBLISH_MODEL
(DIR=*input-directory-path*, MODELNAME=*name*
<, MODELTABLE=*model-table-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.

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 19

MODELNAME=*name*
specifies the name for the published model.

Restriction	The model filename cannot exceed 128 characters. For more information, see "Oracle Scoring Files" on page 120.

MODELTABLE=*model-table-name*
specifies the name of the model table where the scoring files are published.

Default	sas_model_table
Requirement	The name of the model table must be the same as the name specified in the %INDOR_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in "%INDOR_CREATE_MODELTABLE Macro Syntax" on page 116.

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 1000.
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 an Oracle database to which the scoring files are published.

Default	The database specified in the INDCONN macro variable
Requirement	The name of the database must be the same as the database specified in the %INDOR_CREATE_MODELTABLE macro. For more information, see the DATABASE argument in "%INDOR_CREATE_MODELTABLE Macro Syntax" on page 116.
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 "%INDOR_PUBLISH_MODEL Run Process" on page 117.

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 new files.

REPLACE
overwrites the current files, if files with the same name are already registered.

DROP
causes all files for this model to be dropped from the Oracle database.

Default CREATE

Tip If the model has been previously published and you specify ACTION=CREATE, you receive warning messages from Oracle. If the model has been previously published 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 "Oracle Scoring Files" on page 120.

Tip This argument is useful when testing your scoring models.

See "Special Characters in Directory Names" on page 19

Oracle Scoring Files

When using the SAS Embedded Process, the %INDOR_PUBLISH_MODEL macro produces two scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the SASEPFUNC table function.
- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published. This file is used by the SASEPFUNC table function.

These files are published to the model table that you specify in the %INDOR_PUBLISH_MODEL macro. See Appendix 1, "Scoring File Examples," on page 285 for an example of each of these files.

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

Although you cannot view the scoring files directly, there are two ways to see the models whose files are published:

- Log on to the database using SQLPlus and submit an SQL statement. The following example assumes that the model table where the scoring files were published is **register** and the model name is **reg1**.

```
sqlplus userid/pwd@address
select modelname, modelDS2 from sas_model_table
   where modelname like '%reg1%';
```

The model name and the model .ds2 filename are listed.

- 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 files is **reg**.

```
proc sql noerrorstop;
  connect to oracle (user=username password=xxxx path=mypath);

select * from connection to oracle
      (select modelname, modelDS2
         from sasmodeltablename
            where modelname like '%reg%');
disconnect from oracle;
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*.

SASEPFUNC Table Function

Overview of the SASEPFUNC Table Function

The SASEPFUNC table function is the interface for running the scoring model inside Oracle with the SAS Embedded Process. The SASEPFUNC table function performs the scoring based on the parameters that are passed to it. It uses the .ds2 and .ufmt XML files that are stored in the model table.

This function is created by the SASADMIN user when the in-database deployment package is installed and configured. For more information, see the *SAS In-Database Products: Administrator's Guide*.

Using the SASEPFUNC Table Function

You can use the SASEPFUNC table function using explicit pass-through and PROC SQL or you can use other Oracle query tools such as SQLPlus. Use the SASEPFUNC function in the FROM clause in any SQL expression to run the scoring model.

TIP Look at the SampleSQL.txt file is produced when the %INDOR_PUBLISH_MODEL macro is successfully run. This file can be found in the output directory (OUTDIR argument) that you specify in the %INDOR_PUBLISH_MODEL macro. The SampleSQL.txt file contains basic SQL code that can be used to run your score code inside Oracle. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

TIP The SampleSQL.txt file refers to an ID column in the example table that is populated with a unique integer from 1 to *n*. *n* is 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: Before using the SASEPFUNC table function with the SAS Embedded Process, you must create the model table with the %INDOR_CREATE_MODELTABLE macro. Then, you must publish the files to the model table with the %INDOR_PUBLISH_MODEL macro. For more information, see “Creating a Model

Table” on page 115 and “Running the %INDOR_PUBLISH_MODEL Macro” on page 117.

Here is an example using PROC SQL.

```
proc sql;
connect to oracle (user=userid password=xxxx path=mydatabase);
create table work.sas_score_out1 as select * from connection to oracle
   (SELECT * from table(SASEPFUNC(CURSOR(select * from intrtree),
      'myschema.SAS_PUBLISH_MODEL', 'INTRREG1', 'null',
      'select * from intrtree'));
disconnect from oracle;quit;
```

SASEPFUNC Table Function Syntax

The syntax of the SASEPFUNC table function is as follows.

FROM TABLE (**SASEPFUNC**(
CURSOR (SELECT /* + PARALLEL(*table-alias*, *dop*) */
* FROM *input-table table-alias*),
'*schema-name.model-table-name*',
'*model-name*', '*null*',
'SELECT * FROM '*input-table*'))

Arguments

CURSOR(SELECT /*PARALLEL (*table-alias*, *dop*) */ * FROM *input-table table-alias*)
specifies the SELECT statement to read from the input table.

Tip: You can specify a hint for the degree of parallelism (*dop*) value that is used for reading the input table. For more information, see “Run-Time Guidance for the Oracle Degree of Parallelism (DOP) Setting” on page 123.

table-alias
specifies an alias for the input table name.

Requirement: The table alias must be the same in the parallel hint and the FROM clause.

input-table
specifies the name of the input table that is used by the SASEPFUNC table function.

schema-name.model-table-name
specifies the fully qualified model table name.

Requirement: The table name that you specify for this function must be the same table name that is used in the %INDOR_CREATE_MODELTABLE macro. For more information, see “Creating a Model Table” on page 115.

model-name
specifies the model name.

Requirement: The model name must be the same name that is used in the %INDOR_PUBLISH_MODEL macro. For more information, see “%INDOR_PUBLISH_MODEL Macro Syntax” on page 118.

null
specifies a placeholder value at this time

Tip You can specify either 'null' or an empty string ' '.

SELECT * FROM *input-table*
specifies a simple SELECT statement for the input table.

Requirement The input table name must be the same in the first SELECT statement.

Run-Time Guidance for the Oracle Degree of Parallelism (DOP) Setting

The performance of the Scoring Accelerator for Oracle can be affected by altering the Degree of Parallelism (DOP) setting. The DOP setting defines how many parallel processes work on a single statement. In a Real Application Clusters (RAC) environment, the parallel processes are distributed among the available database instances when the chosen DOP exceeds the expected capabilities of a single node. In environments with mixed workloads and multi-concurrency, you should rely on the parallelism provided by the Oracle database. However, you might want to consider adjusting the DOP setting to achieve maximum throughput for dedicated Scoring Accelerator operations.

Because Oracle and SAS use separate threads during execution, improvements throughput diminish for DOP values that are greater than half the total number of cores available. For example, if you have 128 cores total available for all instances, a DOP greater than 64 is not likely to yield improved performance. Performance is not improved because both the Oracle and SAS processes tend to be CPU intensive. Setting the DOP up to this maximum level assumes that the system is solely dedicated to running the SAS Scoring Accelerator. For a mixed load system, a lower DOP value might be more appropriate.

In RAC environments, Oracle allocates parallel execution servers based on an internal load-balancing algorithm. This allocation ensures approximately average loads across all nodes that are accessible for a given parallel operation. Because the load of the SAS processes is not compensated for in Oracles internal algorithms, it can be beneficial in some environments to change Oracles default behavior. There are two ways of doing so:

- Disable Oracles internal load balancing: this can be accomplished by setting the internal parameter _parallel_load_balancing to FALSE (the default value of this parameter is TRUE). Oracle then does a plain round-robin allocation of processes across all available nodes. This parameter can be changed on a system and session level.
- Adjust the number of parallel execution servers per load balance unit. The load balance unit is chosen internally by Oracle to ensure a maximum co-location of parallel execution servers on a single node. This unit is dependent on the number of available CPUs in a system. You can decrease the unit by setting the internal Oracle parameter _parallel_load_bal_unit. The default value of this parameter is 0, meaning the system internally calculates this value. Similar to DOP, setting the _parallel_load_bal_unit parameter beyond half the core total per instance is not likely to be beneficial.

CAUTION:
Influencing Oracles internal load balancing for parallel execution does not harm a system in any way. However, influencing (changing) Oracles internal algorithms

even in a single session has an impact on the overall system through the different allocation of parallel processing resources across the cluster. You have to test your adjustments for possible run time and performance impacts of the overall system.

Chapter 10

SAS Scoring Accelerator for SAP HANA

Overview of Running Scoring Models in SAP HANA

The integration of the SAS Embedded Process and SAP HANA allows scoring code to be run directly in SAP HANA using the SAS Embedded Process.

The SAS Embedded Process is a SAS server process that runs inside SAP HANA to read and write data. A run model macro creates AFL functions that are used to run the scoring model.

The SAS Scoring Accelerator for SAP HANA requires a certain version of the SAP HANA client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

How to Run a Scoring Model in SAP HANA

To run a scoring model using the SAS Embedded Process, follow these steps.

1. Create a scoring model using SAS Enterprise Miner.
2. Start SAS.
3. Create a table to hold the scoring files.

   The %INDHN_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

   For more information, see "Creating the Model Table" on page 128.
4. Run the %INDHN_PUBLISH_MODEL macro to create the scoring files.

   The %INDHN_PUBLISH_MODEL macro performs the following tasks:

   - translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process.
   - 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 SAP HANA to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDHN_CREATE_MODELTABLE macro.

   For more information, see "Running the %INDHN_PUBLISH_MODEL Macro" on page 131.
5. Run the %INDHN_RUN_MODEL macro.

   The %INDHN_RUN_MODEL macro runs the scoring model. The macro uses the files that were created by the %INDHN_PUBLISH_MODEL macro as well as the table that contains the input data.

   For more information, see "Running the %INDHN_RUN_MODEL Macro" on page 134.
6. Submit an SQL query against the output table.

   For more information, see "Scoring Output" on page 137.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to SAP HANA. You must specify user ID and password to access the machine on which you have installed the SAP HANA system. You must assign the INDCONN macro variable before the %INDHN_CREATE_MODELTABLE, the%INDHN_PUBLISH_MODEL, and the %INDHN_PUBLISH_MODEL macros are invoked.

Note: If you do not specify the connection information in the INDCONN macro variable, use the %INDHN_CREATE_MODELTABLE,

%INDHN_PUBLISH_MODEL, or %INDHN_RUN_MODEL macro DATABASE= argument to specify the DSN.

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

USER=*user* PASSWORD=*password*
<SERVER=*server*>
<PORT=*port-number* | INSTANCE=*instance-number* >
<SCHEMA=*schema-name*>
<PRESERVE_TAB_NAMES=YES | NO>
<PRESERVE_COL_NAMES=YES | NO>

Arguments

USER=*user*
specifies the SAP HANA 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 SAP HANA user ID.

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

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

Interaction You can use the DATABASE= argument in the %INDHN_CREATE_MODELTABLE, %INDHN_PUBLISH_MODEL, or %INDHN_RUN_MODEL macro instead of specifying the SERVER= argument.

PORT=*port-number*
specifies the port number.

Interactions Specify either the PORT= argument or the INSTANCE= argument.

You can use the DATABASE= argument in the %INDHN_CREATE_MODELTABLE, %INDHN_PUBLISH_MODEL, or %INDHN_RUN_MODEL macro instead of specifying the PORT= argument.

INSTANCE=*instance-number*
specifies the instance number.

Interactions Specify either the PORT= argument or the INSTANCE= argument.

You can use the DATABASE= argument in the %INDHN_CREATE_MODELTABLE, %INDHN_PUBLISH_MODEL, or %INDHN_RUN_MODEL macro instead of specifying the INSTANCE= argument.

SCHEMA=*schema-name*
specifies the SAP HANA schema that contains the tables and views that you want to access.

Default If you do not specify a value for the SCHEMA argument, the default schema is used.

PRESERVE_TAB_NAMES=YES | NO

preserves spaces, special characters, and case sensitivity in SAP HANA table names.

YES

specifies that table names are read from and passed to the SAP HANA with special characters, and the exact, case-sensitive spelling of the name is preserved.

NO

specifies that when you create SAP HANA tables or refer to an existing table, the table names are derived from SAS member names by using SAS member name normalization. However, SAP HANA applies its own normalization rules to the SAS member names. Therefore, the table names are created or referenced in the database following the SAP HANA normalization rules.

Default NO

PRESERVE_COL_NAMES=YES | NO

preserves spaces, special characters, and case sensitivity in SAP HANA column names when you create SAP HANA tables.

YES

specifies that column names that are used in table creation are passed to the DBMS with special characters and the exact, case-sensitive spelling of the name is preserved.

NO

specifies that column names that are used to create SAP HANA tables are derived from SAS variable names (VALIDVARNAME= system option) by using the SAS variable name normalization rules. However, SAP HANA applies its own normalization rules to the SAS variable names when creating the SAP HANA column names.

Default NO

TIP The INDCONN macro variable is not passed as an argument to the %INDHN_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 that others cannot access the user credentials.

Creating the Model Table

Overview

When publishing a model in SAP HANA that has deployed the SAS Embedded Process, you must create a table to hold the sasscore_modelname.ds2 and sasscore_modelname_ufmt.xml scoring files. You must run the %INDHN_CREATE_MODELTABLE macro to create the table before you run the %INDHN_PUBLISH_MODEL macro.

You have to create the table only one time to hold a model's scoring files.

The %INDHN_CREATE_MODELTABLE macro uses the SAP HANA table type **`"SASLINK":"sas.ep::TT_SAS_MODEL_TABLE"`**. This table type is created in the SAP HANA system when the SAS Embedded Process is installed.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *schema-name.model-table-name*.

Column Name	Description	Specification
MODELNAME	Contains the name of the model.	NVARCHAR(128) NOT NULL
MODELDS2	Contains the sasscore_*modelname*.ds2 file.	NCLOB NOT NULL
MODELFORMATS	Contains the sasscore_*modelname*_ufmt.xml file.	NCLOB
MODELMETADATA	Reserved by SAS for future use.	NCLOB
MODELUUID*	Contains the UUID of the source model.	NVARCHAR (36)
NOTES*	Contains additional information that describes the source model.	NVARCHAR (512)

* This column is for use by SAS Model Manager.

%INDHN_CREATE_MODELTABLE Run Process

To run the %INDHN_CREATE_MODELTABLE macro, follow these steps:

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor.

   ```
   %let indconn = %str(user='youruserid' password='yourpwd'
       schema=yourschema);
   ```

   The arguments supplied to the INDCONN macro variable can vary. For more information, see the “INDCONN Macro Variable” on page 126.

2. Run the %INDHN_CREATE_MODELTABLE macro.

   For more information, see “%INDHN_CREATE_MODELTABLE Macro Syntax” on page 129.

%INDHN_CREATE_MODELTABLE Macro Syntax

%INDHN_CREATE_MODELTABLE

(<DATABASE=*database-name*>
<, MODELTABLE=<<">*model-schema*<".>><">*model-table-name*<">>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

DATABASE=*database-name*
specifies the name of an SAP HANA database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default The database specified in the INDCONN macro variable or your current database.

Interaction The database that is specified by the DATABASE argument takes precedence over the server, port, or instance value that you specify in the INDCONN macro variable. For more information, see “INDCONN Macro Variable” on page 126.

MODELTABLE=<<">*model-schema*<">.><">*model-table-name*<">
specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default SAS_MODEL_TABLE

Requirements The maximum table name length is 30 characters, and it must be a valid SAP HANA table name.

If the table name is case sensitive, you must enclose the name in double quotation marks and set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

Interactions If the name of the model table is not fully qualified in the macro call, the table is created in the schema specified in either the INDCONN macro variable or the user’s default schema.

The table name that you specify for this macro must be the same table name that is used in the %INDHN_PUBLISH_MODEL and the %INDHN_RUN_MODEL macros.

See “%INDHN_PUBLISH_MODEL Macro Syntax” on page 131

“INDCONN Macro Variable” on page 126

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

CREATE
creates a new table.

Tip If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

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

Tip If you specify ACTION=REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP
causes all models in this table to be dropped.

Default CREATE

Running the %INDHN_PUBLISH_MODEL Macro

%INDHN_PUBLISH_MODEL Macro Run Process

To run the %INDHN_PUBLISH_MODEL macro, follow these 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. Populate the directory with the score.sas file, the score.xml file, and (if needed) the format catalog.
3. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

   ```
   %let indconn = %str(user='youruserid' password='yourpwd'
       schema=yourschema);
   ```

   The arguments supplied to the INDCONN macro variable can vary. For more information, see the “INDCONN Macro Variable” on page 126.
4. Run the %INDHN_PUBLISH_MODEL macro.

   For more information, see “%INDHN_PUBLISH_MODEL Macro Syntax” on page 131.

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

%INDHN_PUBLISH_MODEL Macro Syntax

%INDHN_PUBLISH_MODEL
(DIR=*input-directory-path*, MODELNAME=*name*
<, MODELTABLE=<<">*model-schema*<".>><">*model-table-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 directory 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 19

MODELNAME=*name*
specifies the name for the published model.

Restriction	The model name cannot exceed 128 characters.

MODELTABLE=<<">*model-schema*<">.><">*model-table-name*<">
specifies the name of the model table where the scoring files are published.

Default	SAS_MODEL_TABLE
Requirements	The name of the model table must be the same as the name specified in the %INDHN_CREATE_MODELTABLE macro. For more information, see the MODELTABLE argument in “%INDHN_CREATE_MODELTABLE Macro Syntax” on page 129.
	The maximum table name length is 30 characters, and it must be a valid SAP HANA table name.
	If the model table name is case sensitive, you must enclose the name in double quotation marks and set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.
See	“INDCONN Macro Variable” on page 126

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 1000.

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 the ODBC data source for the SAP HANA system to which the scoring files are published.

Default	The database specified in the INDCONN macro variable
Requirement	The name of the database must be the same as the database specified in the %INDHN_CREATE_MODELTABLE macro. For more information, see the DATABASE argument in "%INDHN_CREATE_MODELTABLE Macro Syntax" on page 129.
Interaction	The database that is specified by the DATABASE argument takes precedence over the server, port, or instance value that you specify in the INDCONN macro variable. For more information, see "INDCONN Macro Variable" on page 126.

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 new files.

REPLACE
overwrites the current files, if files with the same name are already registered.

DROP
causes all files for this model to be dropped from the SAP HANA database.

Default	CREATE
Tip	If the model has been previously published and you specify ACTION=CREATE, you receive warning messages from SAP HANA. If the model has been previously published 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.

Tip This argument is useful when testing your scoring models.

See "Special Characters in Directory Names" on page 19

Running the %INDHN_RUN_MODEL Macro

%INDHN_RUN_MODEL Macro Run Process

To run the %INDHN_RUN_MODEL macro, follow these steps:

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

```
%let indconn = %str(user='youruserid' password='yourpwd'
    schema=yourschema);
```

   The arguments supplied to the INDCONN macro variable can vary. For more information, see the "INDCONN Macro Variable" on page 126.

2. Run the %INDHN_RUN_MODEL macro.

   For more information, see "%INDHN_RUN_MODEL Macro Syntax" on page 134.

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

%INDHN_RUN_MODEL Macro Syntax

%INDHN_RUN_MODEL
(DATABASE=*database-name*
, MODELNAME=*name*
, INPUTTABLE=<<">*input-schema*<".>><">*input-table-name*<">
, OUTPUTTABLE=<<">*output-schema*<".>><">*output-table-name*<">
<, MODELTABLE=<<">*model-schema*<".>><">*model-table-name*<">>
<, KEEP=*column-keep-list*>
<, TRACE=*trace-level*>
<, FORCEOVERWRITE=TRUE | FALSE>
<, SASTRACETABLE=*sas-trace-table-name*>
<, EPTRACETABLE=<<">*schema*<".>><">*ep-trace-table-name*<">>
<, NUMTHREADS=*number-of-threads*>
<, NUMDATAPARTITIONS=*number-of-partitions*>
<, DBMAXTEXT=*string-length*>
);

Arguments

DATABASE=*database-name*
specifies the name of the ODBC data source for the SAP HANA system to which the output table is published.

Default The database specified in the INDCONN macro variable

Requirement The name of the database must be the same as the database specified in the %INDHN_CREATE_MODELTABLE macro. For more information, see the DATABASE argument in “%INDHN_CREATE_MODELTABLE Macro Syntax” on page 129.

Interaction The database that is specified by the DATABASE argument takes precedence over the server, port, or instance value that you specify in the INDCONN macro variable. For more information, see “INDCONN Macro Variable” on page 126.

MODELNAME=*name*
specifies the name for the published model.

Restriction The model name cannot exceed 128 characters.

INPUTTABLE=<<">*input-schema*<">.><">*input-table-name*<">
specifies the name of the scoring model input table.

Requirement If the input table name is case sensitive, you must set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

See “INDCONN Macro Variable” on page 126

OUTPUTTABLE=<<">*output-schema*<">.><">*output-table-name*<">
specifies the name of the scoring model output table.

Requirement If the output table name is case sensitive, you must set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

MODELTABLE=<<">*model-schema*<">.><">*model-table-name*<">
specifies the name of the model table where the scoring files are published with the %INDHN_CREATE_MODELTABLE and %INDHN_PUBLISH_MODEL macros.

Default SAS_MODEL_TABLE

Requirements The name of the model table must be the same as the name specified in the %INDHN_CREATE_MODELTABLE and %INDHN_PUBLISH_MODEL macros. For more information, see the MODELTABLE argument in “%INDHN_CREATE_MODELTABLE Macro Syntax” on page 129 and “%INDHN_RUN_MODEL Macro Syntax” on page 134.

The maximum table name length is 30 characters, and it must be a valid SAP HANA table name.

If the model table name is case sensitive, you must set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

See “INDCONN Macro Variable” on page 126

FORCEOVERWRITE=TRUE | FALSE
specifies whether to delete the output table before running the scoring model.

Default FALSE

KEEP=*variable-keep-list*
specifies the column or columns to keep in the output table.

Requirement The list of variables must be separated by spaces and should not be enclosed by single or double quotation marks.

TRACE= *trace-level*
specifies whether debug messages are displayed.

Default 0

Interaction Tracing for the stored procedure is on if TRACE= is greater than zero.

SASTRACETABLE=*sas-trace-data-set-name*
specifies the name of a SAS data set for tracing messages generated by the call to a stored procedure.

Default WORK.SASEP_LOG

Interaction Tracing for the stored procedure is on if TRACE= is greater than zero.

See “TRACE= *trace-level*” on page 136

EPTRACETABLE=<<">*schema*<">.><">*ep-trace-table-name*<">
specifies the name of the SAP HANA table for logging.

Requirement If the trace table name is case sensitive, you must set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

Interaction Tracing for the stored procedure is on if TRACE= is greater than zero.

Note If *ep-trace-table-name* starts with a number sign (#), a local temporary table is used. The table is deleted at the end of the session. If *ep-trace-table-name* does not start with a number sign (#), a permanent table is created in SAP HANA.

See “TRACE= *trace-level*” on page 136

“INDCONN Macro Variable” on page 126

NUMTHREADS=*number-of-threads*
specifies the number of DS2 threads used to run the scoring model.

Default 1

See For more information about DS2 threads, see *SAS DS2 Language Reference*.

NUMDATAPARTITIONS=*number-of-data-partitions*
specifies the number of data partitions used for DS2 processing.

Default 1

Tip The default value of 1 should be used for in-database scoring. You can change the value of NUMDATAPARTITIONS if you are using High-Performance Analytics.

DBMAXTEXT=*string-length*
specifies the default maximum string length for character input columns.

Default 1024 characters

Note The length of the string is also reflected in the output table.

Scoring Output

Scoring Output Table

When you run the %INDHN_RUN_MODEL macro, an output file is created in the database with the name that you specified in the macro's OUTPUTTABLE= argument.

In addition to the input table columns, the %INDHN_RUN_MODEL macro generates a column for each Enterprise Miner output variable in the form of **`EM_outputvarname`**.

You can specify which columns are written to the output table by using the KEEP= argument of the %INDHN_RUN_MODEL macro.

For more information, see "%INDHN_RUN_MODEL Macro Syntax" on page 134.

Querying and Viewing the Scoring Output Table

Note: The %INDHN_RUN_MODEL macro does not generate a SampleSQL.txt file.

The columns in the output table are available to use in any SQL query expression. In these examples, the output table name specified in the %INDDN_RUN_MODEL macro OUTTABLE= argument is **`out_mymodel`**.

```
select * from out_mymodel;
select em_classification from out_mymodel;
```

The output table is written to an SAP HANA table. To use the output data in a SAS session, use a LIBNAME statement to access the table.

To view the output data in a SAS session, you can use PROC PRINT if you have a LIBNAME statement to access the SAP HANA output table.

SAP HANA Permissions

For SAP HANA, the following permissions are needed by the person who runs the scoring publishing macros:

- EXECUTE on SYSTEM.afl_wrapper_generator
- EXECUTE on SYSTEM.afl_wrapper_eraser
- AFL__SYS_AFL_SASLINK_AREA_EXECUTE

- EXECUTE, SELECT, INSERT, UPDATE, and DELETE on the schema that is used for scoring

In addition, the role of **sas.ep::User** and **AFL_SYS_AFL_SASLINK_AREA_EXECUTE** must be assigned to any user who wants to perform in-database processing.

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

For more information about specific permissions, see “SAP HANA Permissions” in Chapter 8 of *SAS In-Database Products: Administrator's Guide*.

Chapter 11

SAS Scoring Accelerator for SPD Server

Overview of Running Scoring Models in SPD Server

The SAS Scoring Accelerator for SPD Server embeds the robustness of SAS Enterprise Miner scoring models directly in the highly scalable SPD Server. The SAS Scoring Accelerator for SPD Server takes the models that are developed by SAS Enterprise Miner and outputs the data to an SPD server table.

The SAS Scoring Accelerator for SPD Server requires SAS Scalable Performance Data Server version 5.1 and SAS 9.4. Installation of an in-database deployment package is not required.

Running Scoring Models in SPD Server

To run the scoring model in SPD Server, follow these steps:

1. Create a scoring model using SAS Enterprise Miner.
2. Start SAS.
3. Submit this command in the Program Editor or Enhanced Editor to provide SPD Server connection information to the %INDSP_PUBLISH_MODEL and the %INDSP_PUBLISH_MODELmacros.

```
%let indconn = domain=mydomain server=myserver port=myport
   user=myuserid <password=XXXX>;
```

For more information, see “INDCONN Macro Variable” on page 141.

4. Run the %INDSP_PUBLISH_MODEL macro.

   The %INDSP_PUBLISH_MODEL macro publishes the model to the SPD Server. This model is made available to run using data stored in the SPD Server.

   The %INDSP_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.

   After the model is published, two SPD Server tables are created in the INDCONN domain: *modelname*_ARGS and *modelname*_PKG. If you have user-defined formats in the model, an additional table, *modelname*_FMT, is created.

   For more information, see “%INDSP_PUBLISH_MODEL Macro Syntax” on page 142.

5. Submit these commands in the Program Editor or Enhanced Editor to provide SPD Server connection information to the %INDSP_PUBLISH_MODEL and the %INDSP_PUBLISH_MODEL macros.

   ```
   %let indconn = domain=mydomain server=myserver port=myport
      user=myuserid <password=XXXX>;

   %let inddata domain=mydomain server=myserver port=myport
      user=myuserid <password=XXXX>;
   ```

   The INDDATA macro variable provides the SPD Server connection information for the input and output tables.

   Note: You do not have to resubmit **%let indconn=** if you already submitted the command in your SAS session to publish the model.

   For more information, see “INDCONN Macro Variable” on page 141 and “INDDATA Macro Variable” on page 141.

6. Run the %INDSP_RUN_MODEL macro.

   The %INDSP_RUN_MODEL macro runs the scoring model. The %INDSP_RUN_MODEL macro uses the tables that were created by the %INDSP_PUBLISH_MODEL macro as well as table that contains the input data.

   The %INDSP_RUN_MODEL macro creates an SPD Server output table by default. The output table name is *modelname*_OUT. When the model is run, another SPD Server table, *modelname*_THR, is created in the INDCONN domain.

   Note: The %INDSP_PUBLISH_MODEL and the %INDSP_RUN_MODEL must be run from sessions that use the same encoding or locale.

   For more information, see “%INDSP_RUN_MODEL Macro Syntax” on page 144.

7. Submit an SQL query against the output table.

   For more information, see “Scoring Output” on page 145.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to the SPD Server. You must specify domain, server, port, and user. The password is optional. You must assign the INDCONN macro variable before the %INDSP_PUBLISH_MODEL or %INDSP_RUN_MODEL macro is invoked.

Note: The connection information provided by the INDCONN macro variable must be the same for both the %INDSP_PUBLISH_MODEL macro and the %INDSP_RUN_MODEL macro.

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

DOMAIN=*domain-name* SERVER=*server* PORT=*port-number* USER=*user* <PASSWORD=*password*>

Arguments

DOMAIN=*domain-name*
specifies the domain name.

SERVER=*server*
specifies the server name.

PORT=*port-number*
specifies the port number.

USER=*username*
specifies the user name (also called the user ID) that is used to connect to the SPD Server.

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

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

INDDATA Macro Variable

The INDDATA macro variable is used to provide SPD Server connection credentials to the input table. You must specify domain, server, port, and user. The password is optional. You must assign the INDDATA macro variable before the %INDSP_RUN_MODEL macro is invoked.

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

DOMAIN=*domain-name* SERVER=*server* PORT=*port-number* USER=*user* <PASSWORD=*password*>

Arguments

DOMAIN=*domain-name*
specifies the domain name.

SERVER=*server*
specifies the server name.

PORT=*port-number*
specifies the port number.

USER=*username*
specifies the user name (also called the user ID) that is used to connect to the SPD Server.

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

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

%INDSP_PUBLISH_MODEL Macro Syntax

Note: The %INDSP_PUBLISH_MODEL and the %INDSP_RUN_MODEL must be run from sessions that use the same encoding or locale.

%INDSP_PUBLISH_MODEL
(MODELNAME=*name*, INPUTDIR=*input-directory-path*
<, DATASTEP=*score-program-filename*>
<, XML=*xml-filename*>
<, FMTCAT=*format-catalog-filename* | *libref.format-catalog-filename*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

MODELNAME=*name*
specifies the name that is prepended to the SPD Server tables that are created after the %INDSP_PUBLISH_MODEL macro runs.

Requirement	The model name cannot exceed ten characters and 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*.
Tip	After the model is published, two SPD Server tables are created in the INDCONN domain: *modelname*_ARGS and *modelname*_PKG. If you have user-defined formats in the model, an additional table, *modelname*_FMT, is created.

INPUTDIR=*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.
See	“Special Characters in Directory Names” on page 19

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.
Requirement	The scoring model program file must be located in the INPUTDIR directory.
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
Restriction	Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.
Requirement	The properties XML file must be located in the INPUTDIR directory.
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.

FMTCAT=*format-catalog-filename* | *libref.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	FORMATS
Restriction	Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.
Requirement	If the catalog is specified as a one-level name, it is expected to be found in the same location as the scoring model program (score.sas) and the scoring model XML file (score.xml). If the catalog is specified as a two-level name (*libref.format-catalog-name*), the catalog is expected to be found using the two-level name.
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 model.

Note	If the model by the same name exists, an error occurs.

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

Note	If the current model does not exist, an error occurs.

DROP
causes the model to be deleted.

Notes	If you specify ACTION=DROP, the *modelname*_ARGS, *modelname*_PKG, *modelname*_THR, and if created, *modelname*_FMT tables are also deleted.
	If the model does not exist, an error occurs.

Default	CREATE

%INDSP_RUN_MODEL Macro Syntax

Note: The %INDSP_PUBLISH_MODEL and the %INDSP_RUN_MODEL must be run from sessions that use the same encoding or locale.

%INDSP_RUN_MODEL
(MODELNAME=*name*, INDATA=*input-table-name*
<, OUTDATA=*output-table-name*>
<, KEEP=*variable-keep-list*>
<, OUTDIR=*sample-sql-directory*>
<, THREADS=*number-of-threads*>
);

Arguments

MODELNAME=*name*
specifies the name of the model.

Requirement	The model name cannot exceed ten characters and 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	*modelname* is also used to create the output table name (*modelname*_OUT) if a name is not specified in the OUTDATA= argument.
Tip	When the model is published, the *modelname*_THR SPD Server table is created in the INDCONN domain.

INDATA=*input-table-name*
specifies the name of the SPD Server table to be used as input to the scoring model.

Restriction The table must exist in the INDDATA domain.

OUTDATA=*output-table-name*
specifies the name of the SPD Server table created as output from the scoring model.

Default *modelname*_OUT

Note The table is created in the INDDATA domain.

KEEP=*variable-keep-list*
specifies a list of variables to keep in the output table.

Requirement The variables in the list must be separated by spaces.

Example `keep=id em_eventprobability em_classification`

OUTDIR=*sample-sql-directory*
specifies a directory that contains the sample SQL code (SampleSQL.txt).

Tip This argument is useful when testing your scoring models. If you do not specify this directory, the sample SQL code is not available after the model runs.

See For more information about the SampleSQL.txt file, see "Querying Scoring Output Tables" on page 146.

"Special Characters in Directory Names" on page 19

THREADS=*number-of-threads*
specifies the number of threads to use when the model is run.

Default 1

Scoring Output

Scoring Output Table

When you run the %INDSP_RUN_MODEL macro, an SPD Server table is created in the domain that was specified in the INDDATA macro variable. The default table name is *modelname*_OUT. However, you can specify a different table name in the OUTDATA argument of the %INDSP_RUN_MODEL macro.

In addition to the input table columns, the %INDSP_RUN_MODEL macro generates a column for each Enterprise Miner output variable in the form of **`EM_`*`outputvarname`***.

You can specify which columns are written to the scoring output table by using the KEEP argument of the %INDSP_RUN_MODEL macro. For more information about the MODELNAME and KEEP arguments, see "%INDSP_RUN_MODEL Macro Syntax" on page 144.

Querying Scoring Output Tables

The columns in the output table are available to use in any SQL query expression.

```
select * from mymodel_out;

select em_classification from mymodel_out;
```

The output table is written to an SPD Server table. To use the output data in a SAS session, use a LIBNAME statement to access the SPD Server table.

In addition, a SampleSQL.txt file is produced when the %INDSP_RUN_MODEL macro runs. This file can be found in the output directory (OUTDIR argument) that you specify in the macro.

The SampleSQL.txt file contains SPD Server SQL and DS2 code that can be used to query the model output table. The SampleSQL.txt file code is specific to the scoring model that was run against the specified input table.

To run the code in the SampleSQL.txt file, you need only to include this file within a PROC SQL statement.

```
proc sql;
%inc 'SampleSQL.txt';
quit;
```

The following example assumes the model name that you used is **almush01**. The output table is **almush01_out**. The CONNECT TO and EXECUTE... BY statements are SPD Server SQL statements. The code within the EXECUTE block is DS2 code that executes in SPD Server through SQL.

```
/* Model almush01 run 14JUN13 7:41:42 PM */
/* Input is table almush01; Output is table almush01_out. */
/* Running model with 1 thread. */
connect to sasspds(dbq="model" host="myhost" serv="1234" user="anonymous");
execute(reset dialect=ds2) by sasspds;
execute(
ds2_options sas tkgmac scond=none;
thread almush01_thr / overwrite=yes;
dcl package modSPDS.almush01_pkg scorepkg();
dcl char( 32) "EM_CLASSIFICATION";
dcl double "EM_EVENTPROBABILITY";
dcl double "EM_PROBABILITY";
dcl double "G_CAPCOLOR";
dcl double "G_GILLCOLO";
dcl double "G_HABITAT";
dcl double "G_ODOR";
dcl double "G_POPULAT";
dcl double "G_RINGTYPE";
dcl double "G_SPOREPC";
dcl double "G_STALKCBR";
dcl double "G_STALKROO";
dcl double "G_STALKSAR";
dcl double "G_VEILCOLO";
dcl double "H11";
dcl double "H12";
dcl double "H13";
dcl char( 1) "I_TARGET";
```

```
dcl double "P_TARGETE";
dcl double "P_TARGETP";
dcl double "S_G_CAPCOLOR";
dcl double "S_G_GILLCOLO";
dcl double "S_G_HABITAT";
dcl double "S_G_ODOR";
dcl double "S_G_POPULAT";
dcl double "S_G_RINGTYPE";
dcl double "S_G_SPOREPC";
dcl double "S_G_STALKCBR";
dcl double "S_G_STALKROO";
dcl double "S_G_STALKSAR";
dcl double "S_G_VEILCOLO";
dcl char( 1) "U_TARGET";
dcl char( 4) "_WARN_";
method run();
set dataSPDS.almush01;
scorepkg.score("BRUISES",
"CAPCOLOR",
"GILLCOLO",
"GILLSIZE",
"HABITAT",
"ODOR",
"POPULAT",
"RINGNUMB",
"RINGTYPE",
"SPOREPC",
"STALKCBR",
"STALKROO",
"STALKSAR",
"STALKSHA",
"VEILCOLO",
"EM_CLASSIFICATION",
"EM_EVENTPROBABILITY",
"EM_PROBABILITY",
"G_CAPCOLOR",
"G_GILLCOLO",
"G_HABITAT",
"G_ODOR",
"G_POPULAT",
"G_RINGTYPE",
"G_SPOREPC",
"G_STALKCBR",
"G_STALKROO",
"G_STALKSAR",
"G_VEILCOLO",
"H11",
"H12",
"H13",
"I_TARGET",
"P_TARGETE",
"P_TARGETP",
"S_G_CAPCOLOR",
"S_G_GILLCOLO",
"S_G_HABITAT",
"S_G_ODOR",
```

```
"S_G_POPULAT",
"S_G_RINGTYPE",
"S_G_SPOREPC",
"S_G_STALKCBR",
"S_G_STALKROO",
"S_G_STALKSAR",
"S_G_VEILCOLO",
"U_TARGET",
"_WARN_");
output;
end;
endthread;
data dataSPDS.almush01_out (overwrite=yes);
keep ID EM_CLASSIFICATION EM_EVENTPROBABILITY EM_PROBABILITY;
dcl thread almush01_thr st;
method run();
set from st threads=1;
output;
end;
enddata;
) by sasspds;
```

For more information about SPD Server SQL language, see the *SAS Scalable Performance Data Server: User's Guide*. For more information about the DS2 language, see the *SAS DS2 Language Reference*.

SPD Server Permissions

You must have permissions for the domains that you specify in the INDCONN and INDDATA macro variables when you execute the publish and run macros.

You also need regular Read, Write, and Alter permissions when writing files to the OUTDIR directory in the %INDSP_RUN_MODEL macro.

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

Chapter 12

SAS Scoring Accelerator for Teradata

Overview of Running Scoring Models in Teradata

There are two ways to run scoring models in Teradata.

- You can create scoring functions for each EM_ output variable. The model publishing macro creates the scoring functions that are published as Teradata user-defined functions. These functions can then be used in any SQL query. For more information, see "Using Scoring Functions to Run Scoring Models" on page 150.
- You can use the SAS Embedded Process. The SAS Embedded Process is a SAS server process that runs inside the Teradata Enterprise Data Warehouse (EDW) to read and write data. The model publishing macro creates scoring files that are then used in a stored procedure to run the scoring model. For more information, see "Using the SAS Embedded Process to Run Scoring Models" on page 153.

The SAS Scoring Accelerator for Teradata requires a certain version of the Teradata client and server environment. For more information, see the SAS Foundation system requirements documentation for your operating environment.

Using Scoring Functions to Run Scoring Models

How to Run a Scoring Model Using Scoring Functions

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 27.

To run the scoring model using scoring functions, follow these steps.

1. Run the %INDTD_PUBLISH_MODEL macro.

   The %INDTD_PUBLISH_MODEL macro uses some 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 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.
   - processes the format catalog and creates an .h file with C structures if a format catalog is available. This file is also necessary to build the scoring functions.
   - produces a script of the Teradata commands that are used to register the scoring functions on the Teradata EDW.
   - uses SAS/ACCESS Interface to Teradata to run the script and publish the scoring model files to the Teradata EDW.

   For more information, see "Running the %INDTD_PUBLISH_MODEL Macro" on page 164.

   For more information about the scoring functions that are created, see "Scoring Function Names" on page 150 and "Viewing the Scoring Functions" on page 151.

2. Use the scoring functions in any SQL query.

   For more information, see "Using Scoring Functions to Run a Scoring Model" on page 153.

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 164.

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 27.

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 %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.

Viewing the Scoring Functions

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 from teradata;
  quit;
  ```

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

- 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 SQL code that can be used to run your score code inside Teradata. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

  For example, this SampleSQL.txt file refers to an ID column in **`allmush1_intab`** that is populated with a unique integer from 1 to *n*. *n* is 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 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 ;
```

Using Scoring Functions to Run a Scoring Model

The scoring functions are available to use in any SQL expression in the same way that Teradata built-in functions are used.

After the scoring functions are created, 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. The SampleSQL.txt file shown in "Viewing the Scoring Functions" on page 151 was modified to create the SELECT statement in this example.

```
select id, allmush1_em_classification
(
"BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC")
  as "EM_CLASSIFICATION",
 from allmush1_intab ;
```

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

Using the SAS Embedded Process to Run Scoring Models

How to Run a Scoring Model with the SAS Embedded Process

The integration of the SAS Embedded Process and Teradata allows scoring code to run directly using the SAS Embedded Process on Teradata.

Note: The SAS Embedded Process might require a later release of Teradata than function-based scoring does. For more information, see the SAS Foundation system requirements documentation for your operating environment.

To run the scoring model using the SAS Embedded Process, follow these steps.

1. Create a table to hold the scoring files.

The %INDTD_CREATE_MODELTABLE macro creates a table that holds the scoring files for the model that is being published.

For more information, see "Creating a Model Table" on page 154.

2. Run the %INDTD_PUBLISH_MODEL to create the scoring files.

   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 performs the following tasks:

   - translates the scoring model into the sasscore_*modelname*.ds2 file that is used to run scoring inside the SAS Embedded Process.
   - 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 Teradata to insert the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files into the model table that was created using the %INDTD_CREATE_MODELTABLE macro.

   For more information, see "Running the %INDTD_PUBLISH_MODEL Macro" on page 164 and "Teradata Scoring Files" on page 162.

3. Execute the SAS_SCORE_EP stored procedure to run the scoring model.

   For more information, see "SAS_SCORE_EP Stored Procedure" on page 156.

Creating a Model Table

Overview

When using the SAS Embedded Process to publish a scoring model in Teradata, you must create a table to hold the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files. You must run the %INDTD_CREATE_MODELTABLE macro to create the table before you run the %INDTD_PUBLISH_MODEL macro.

You have to create the table only one time to hold a model's scoring files.

The model table contains the following columns. The ModelName column is the table key. The table is referenced by the two-level name *model-name.model-table-name*.

Column Name	Description	Specification
ModelName	contains the name of the model	VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC
ModelDS2	contains the sasscore_*modelname*.ds2 file	BLOB(209708800)
ModelFormats	contains the sasscore_*modelname*_ufmt.xml file	BLOB(209708800)

Column Name	Description	Specification
ModelMetadata	reserved by SAS for future use	BLOB(4M)
ModelOwner**	contains the name of the user who published the model	VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC
ModelUpdated**	contains the date and time that the model was published	TIMESTAMP(6)
ModelUUID*	contains the UUID of the source model	VARCHAR (36)
Notes*	contains additional information that describes the source model	VARCHAR (512)

* This column is for use by SAS Model Manager. If you have a model table that was created prior to SAS 9.4 and you want this column in your model table, you must run the %INDTD_CREATE_MODELTABLE macro to re-create your model table.

** This column exists in model tables that were run in SAS 9.3 and earlier releases. These columns are compatible with SAS 9.4, but it is not created if you run the %INDTD_CREATE_MODELTABLE macro in SAS 9.4. The ModelUUID and Notes columns are created instead.

%INDTD_CREATE_MODELTABLE Run Process

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

1. Start SAS and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the “INDCONN Macro Variable” on page 164.

2. Run the %INDTD_CREATE_MODELTABLE macro.

   For more information, see “%INDTD_CREATE_MODELTABLE Macro Syntax” on page 155.

%INDTD_CREATE_MODELTABLE Macro Syntax

%INDTD_CREATE_MODELTABLE
(<DATABASE=*database-name*>
<, MODELTABLE=*model-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
);

Arguments

DATABASE=*database-name*
specifies the name of a Teradata database where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files are held.

Default	The database specified in the INDCONN macro variable or your current database
Requirement	The maximum database name length is 30 characters, and it must be a valid Teradata database name.

MODELTABLE=*model-table-name*

specifies the name of the table that holds the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files.

Default	sas_model_table
Requirement	The maximum table name length is 30 characters, and it must be a valid Teradata table name.
Interaction	The table name that you specify for this macro must be the same table name that is used in the %INDTD_PUBLISH_MODEL macro.
See	"%INDTD_PUBLISH_MODEL Macro Syntax" on page 165

ACTION = CREATE | REPLACE | DROP

specifies one of the following actions that the macro performs:

CREATE

creates a new table.

Tip: If the table has been previously defined and you specify ACTION=CREATE, an error is issued.

REPLACE

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

Tip: If you specify ACTION = REPLACE, and the current table contains sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml files, the files are deleted and an empty table is re-created.

DROP

causes all models in this table to be dropped.

Default: CREATE

SAS_SCORE_EP Stored Procedure

Overview of the SAS_SCORE_EP Stored Procedure

The SAS_SCORE_EP stored procedure is the interface for running the scoring model inside Teradata with the SAS Embedded Process. The SAS_SCORE_EP stored procedure uses the files that are stored in the model table. The stored procedure parameters enable you to control the name and location of the output table, how much data is returned, and how it is returned.

The SAS_SCORE_EP stored procedure is installed in the SAS_SYSFNLIB database. To run the stored procedure, you must have the following permissions:

- EXECUTE PROCEDURE permission on the SAS_SYSFNLIB database
- EXECUTE FUNCTION permission on the SAS_SYSFNLIB database
- EXECUTE FUNCTION ON SYSLIB.MonitorVirtualConfig permission on the SYSLIB.MonitorVirtualConfig function

For more information, see "Teradata Permissions" on page 169.

Running the SAS_SCORE_EP Stored Procedure

You can run the SAS_SCORE_EP stored procedure using explicit pass-through and PROC SQL or you can use other Teradata query tools.

TIP 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 %INDTD_PUBLISH_MODEL macro. The SampleSQL.txt file contains basic SQL code that can be used to run your score code inside Teradata. Please note that you must modify the sample code before using it. Otherwise, the sample code returns an error.

Note: Before running the SAS_SCORE_EP stored procedure, you must create the model table with the %INDTD_CREATE_MODELTABLE macro. Then, you must publish the files to the model table with the %INDTD_PUBLISH_MODEL macro.

Here is an example using PROC SQL.

```
proc sql;
   connect to teradata (user=userid password=xxxx server=myserver mode=Teradata);
   execute
      (CALL SAS_SYSFNLIB.SAS_SCORE_EP
         (
            'MODELTABLE="grotto"."sas_publish_model"',
            'MODELNAME=Intr_Tree',
            'INQUERY=SELECT * from "grotto"."score_input_table" WHERE x1 < 1.0',
            'OUTTABLE="grotto"."sas_score_out1"',
            'OUTKEY=id',
            'OPTIONS='  /* can be blank or NULL if no options are needed */
         )
      ) by teradata;
   disconnect from teradata;
quit;
```

Note: You must specify MODE=TERADATA in your connection string.

For more information about the stored procedure parameters, see “SAS_SCORE_EP Stored Procedure Syntax” on page 157.

SAS_SCORE_EP Stored Procedure Syntax

SAS_SYSFNLIB.SAS_SCORE_EP

('MODELTABLE="*database*"."*model-table-name*" ',
'MODELNAME=*model-name*',
'INQUERY=SELECT …'
'OUTTABLE=< "*output-database-name*". >"*output-table-name*" ',
'OUTKEY=*column*<…, *column*,> | NO PRIMARY INDEX,
NULL | 'OPTIONS=' | 'OPTIONS=*option*; <…; *option*;>'
);

Parameters

database

specifies the name of the database that contains the model table.

Default	Database in the current session
Requirements	The database must be the same as the one specified in the %INDTD_PUBLISH_MODEL macro’s DATABASE argument.

The maximum database name length is 30 characters, and it must be a valid Teradata database name.

model-table-name
specifies the name of the model table where the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files were published with the %INDTD_CREATE_MODELTABLE macro.

model-name
specifies the name of the model.

SELECT …
specifies a SELECT statement that defines the inputs to the SAS Embedded Process.

Range	The INQUERY= parameter string can be up to 30,000 characters long.
Restrictions	The maximum number of characters in the query is 30,000.
	The maximum number of input and output columns is 1024.
Requirements	If the query is greater than 1,000 characters, the INQUERY= parameter must be the first parameter listed in the stored procedure.
	The SELECT statement must be syntactically correct SQL.
Interaction	A query can reference tables or views, except if you specify the DIRECT option. If you specify the DIRECT option, the query can reference only tables.
Tips	If you want to query all data from the input data without any filtering or subsetting (**'INQUERY=SELECT * FROM***table*'), you can use the table name in the INQUERY argument. However, you must also add DIRECT=YES to the OPTIONS argument. Here is an example

```
...
'inquery="myDatabase"."myTableName"',
'options=direct=yes',
...
```

To pass single quotation marks around character literals, use two adjacent single quotation marks. This is an example.

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%'''
```

< "*output-database-name*". >"*output-table-name*"
specifies the location of the scoring model output table.

output-database-name
specifies the database where the table is created or currently exists.

Default	Current database
Requirement	The maximum database name length is 30 characters, and it must be a valid Teradata database name.

output-table-name
specifies the name of the table to be created or the table that currently exists.

Requirement	The output table can already exist. If the output table already exists, scored rows are inserted into the table along with any existing rows.

	If the output table already exists, the output columns must match the existing table's columns for the insert to succeed.
Interaction	The output table can be a temporary table by adding VOLATILE=YES in the OPTIONS parameter. The temporary table can be used only for the duration of the SQL session where it is created.

column

specifies the column or columns that are used as the primary index for the output table.

Requirements	The column must exist in the output table.
	If there are multiple primary index columns, the column names must be separated by commas.
Tip	Specifying the same primary index for both the input and output tables enables Teradata to avoid redistribution of data across its AMPs.

NO PRIMARY INDEX

specifies that there is no primary index for the output table and that output rows are placed on the same Teradata Access Module Processor (AMP) that ran the scoring code for the corresponding input row.

NULL | 'OPTIONS='

specifies that no options are used.

Tip You can use either 'OPTIONS=' or NULL to indicate that no options are used.

Example These two code lines are identical.

```
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
null);
```

```
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
'options=');
```

option

specifies additional options for the stored procedure. *option* can be any of the following values:

CONTRACT=YES | NO

specifies whether to write the output metadata to the table specified in the OUTTABLE= parameter.

Default	NO
Interaction	If you specify CONTRACT=YES, the OUTKEY= parameter is ignored.
Tip	The output is written to the table in the form of one row per output value with the sequence, name, data type, and length for each output.

DIRECT=YES | NO

specifies whether direct retrieve optimization is enabled.

Default	NO
Interaction	This option affects the stored procedure SQL generation.
Tip	The direct retrieve optimization improves performance in the case where the input to the SAS Embedded Process is a table and the input query is **`SELECT * FROM table`**. When DIRECT=YES, the INQUERY= parameter is only the table name. No SELECT statement is needed.

DS2_KEEP=*column-name*<*…column-name*>

specifies the column or columns that are passed to the SAS_SCORE_EP procedure and are applied as a dynamic KEEP= statement in the sasscore_*modelname*.ds2 file.

Requirement	If more than one column is specified, column names must be separated with spaces.
Interaction	Specify CONTRACT=YES to preview the available output columns without executing the model.

ENCODING= LATIN | UNICODE

specifies the character data encoding for the column data. This is for internationalization purposes.

Default	LATIN
See	*SAS National Language Support (NLS): Reference Guide*

EPTRACE=YES

specifies that journal messages are written to the journal table.

HASHBY=*column-name*<*…, column-name*>

specifies one or more columns to use for the HASH BY clause.

Requirement	If more than one column is specified, column names must be separated with commas.
Interaction	This option affects the stored procedure SQL generation.
Note	Data is redistributed by hash code to the TERADATA AMPs based on this column or columns although there is no implied ordering to the groups.

JOURNALTABLE=*journal-table-name*

specifies the name of a table that the stored procedure creates. This table holds any journal messages and notes from the SAS journal facility that are produced when executing the store procedure.

Requirement	The name must follow Teradata naming conventions for table names.
Note	Use a SELECT statement to retrieve the journal messages from the table after the stored procedure call is complete.

LOCALE=*sas-locale*

specifies set of attributes in the SAS session that reflect the language, local conventions, and culture for a geographical region.

Requirement *sas-locale* must be one of the five-character POSIX values, for example fr_FR.

See *SAS National Language Support (NLS): Reference Guide*

ORDERBY=*column-name*<…, *column-name*>
specifies one or more columns to use for the LOCAL ORDER BY (BY groups) clause.

Requirement If more than one column is specified, column names must be separated with commas.

Interaction This option affects the stored procedure SQL generation.

SELECT_LIST=*column-name*<…, *column-name*>
specifies the column or columns that are used in the SQL that is generated by the SAS_SCORE_EP stored procedure.

Default * (asterisk) which indicates all columns

Requirement If more than one column is specified, column names must be separated with commas.

SQLTRACE=*table-name*
specifies the name of a table to hold the generated SQL code.

Tip This table is useful for stored procedure debugging or to reference later if you want to customize the SQL code that is used to call the SAS Embedded Process.

UNIQUE=YES | NO
specifies whether the primary index of the output table is unique.

Default NO

VOLATILE=YES | NO
specifies whether the output table is created as a temporary table.

Default NO

Interaction This option affects the stored procedure SQL generation.

Range The OPTIONS= parameter string can be from 0–20,000 characters long.

Requirements Each option must end with a semicolon, including the last option in the list.

If the OPTIONS= parameter string is greater than 1,000 characters, the OPTIONS= parameter must be the last one.

Note *option* can be blank or NULL if no options are needed.

Tip Options that are not recognized as directives to the stored procedure are passed to the SAS Embedded Process as Query Band name-value pairs. If the SAS Embedded Process does not recognize them, they are ignored. Up to ten user-defined Query Band name-value pairs can be specified in addition to the options listed here that are Query Band name-value pairs. The maximum length of the query band is 2048 characters. User-defined Query Band information is logged in Teradata Database Query Log (DBQL) that makes it useful for workload analysis and reporting.

Tips for Using the SAS_SCORE_EP Stored Procedure

- The SAS Embedded Process for Teradata supports only ISO-8859-1 (Latin-1) encoding for table metadata. Examples of table metadata include table and column names.
- No specific parameter order is required. However, the INQUERY parameter must be the first parameter if its string is greater than 1,000 characters. Similarly, if the OPTIONS parameter string is greater than 1,000 characters, it must be the last parameter.
- Database object names (for example tables and columns) must be enclosed in double quotation marks if they are Teradata reserved words. Otherwise, quotation marks are optional.
- Tables can be qualified with a database name. If a table name is not qualified with a database name, the table name is resolved based on the default database for your session.
- All parameters are passed as strings to the SAS_SCORE_EP stored procedure, so they must be enclosed in single quotation marks. To pass a single quotation mark as part of the SQL within a parameter, use two adjacent single quotation marks as shown in the following example:

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%''',
```

Teradata Scoring Files

When using the SAS Embedded Process, the %INDTD_PUBLISH_MODEL macro produces two scoring files for each model:

- sasscore_*modelname*.ds2. This file contains code that is executed by the SAS_SCORE_EP stored procedure.
- sasscore_*modelname*_ufmt.xml. This file contains user-defined formats for the scoring model that is being published. This file is used by the SAS_SCORE_EP stored procedure.

These files are published to the model table that you specify in the %INDTD_PUBLISH_MODEL macro. See Appendix 1, "Scoring File Examples," on page 285 for an example of each of these files.

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

Although you cannot view the scoring files directly, there are two ways to see the models whose files are created:

- Log on to the database using BTEQ and submit an SQL statement. The following example assumes that the model table where the scoring files were published is **register** and the model name is **reg1**.

```
bteq .logon myserver/myuserid,mypassword
select modelname, modelowner, modelupdated from register
   where modelname like '%reg1%';
```

The model name, user ID, and date and time that the model files were published are listed.

- 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 files is **reg**.

```
proc sql noerrorstop;
  connect to teradata (user=username password=xxxx server=myserver);

select * from connection to teradata
      (select modelname,modelowner,modelupdated
         from sasmodeltablename
            where modelname like '%reg%');
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*.

Controlling the SAS Embedded Process

The SAS Embedded Process starts when a query is submitted. It continues to run until it is manually stopped or the database is shut down.

You can check the status of the SAS Embedded Process or disable it so that no new queries can be started. Use the following commands to perform those actions.

Action Performed	Command
Provides the status of the SAS Embedded Process.	CALL DBCEXTENSION.SERVERCONTROL ('status', :A); * CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'status', :A); ** CALL SQLJ.SERVERCONTROL ('SAS', 'status', :A); ***
Stops new queries from being started. Queries that are currently running continue to run until they are complete.	CALL DBCEXTENSION.SERVERCONTROL ('disable', :A); * CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'disable', :A); ** CALL SQLJ.SERVERCONTROL ('SAS', 'disable', :A); ***
Enables new queries to start running.	CALL DBCEXTENSION.SERVERCONTROL ('enable', :A);* CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'enable', :A);** CALL SQLJ.SERVERCONTROL ('SAS', 'enable', :A); ***

* For Teradata 13.10 and 14.00 only. Note that the Cmd parameter (for example, 'status', must be lowercase.

** For Teradata 14.10 only. Note that the Languagename parameter, 'SAS', is required and must be uppercase. The Cmd parameter (for example, 'status'), must be lowercase.

*** For Teradata 15 only. Note that the Languagename parameter, 'SAS', is required and must be uppercase. The Cmd parameter (for example, 'status'), must be lowercase.

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. 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 and submit the following command in the Program Editor or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 164.
5. If you use the SAS Embedded Process, run the %INDTD_CREATE_MODELTABLE macro.

   For more information, see "Creating a Model Table" on page 154.
6. Run the %INDTD_PUBLISH_MODEL macro.

   Messages are written to the SAS log that indicate whether the scoring functions or files were successfully created.

   For more information, see "%INDTD_PUBLISH_MODEL Macro Syntax" on page 165.

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 or the %INDTD_CREATE_MODELTABLE macros are 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.

Default	Your current database
Requirement	You must specify the DATABASE= argument if you use the SAS Embedded Process.

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*
<, MECHANISM=STATIC | EP>
<, MODELTABLE=*model-table-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 19

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. If you use the SAS Embedded Process, the model name is part of the scoring filenames.

Restriction	The scoring function name is a combination of the model name and the output variable name. The scoring function name cannot exceed

30 characters. For more information, see “Scoring Function Names” on page 150.

Requirement	If you use scoring functions, the model name must be a valid SAS name that is ten characters or fewer. If you use the SAS Embedded Process, the model name can be up to 128 characters. 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 27 and “Scoring Function Names” on page 150.

MECHANISM=STATIC | EP

specifies whether scoring functions or scoring files are created. MECHANISM= can have one of the following values:

STATIC

specifies that scoring functions are created.

These scoring functions are used in an SQL query to run the scoring model.

See “Using Scoring Functions to Run Scoring Models” on page 150

EP

specifies that scoring files are created.

These scoring files are used by the SAS Embedded Process to run the scoring model. A single entry in the model table is inserted for each new model. The entry contains both the score.sas and score.xml in separate columns. The scoring process includes reading these entries from the table and transferring them to each instance of the SAS Embedded Process for execution.

Requirement	If you specify MECHANISM=EP, you must also specify the MODELTABLE= argument.
Note	The SAS Embedded Process might require a later release of Teradata than function-based scoring. For more information, see the SAS Foundation system requirements documentation for your operating environment.
See	“Using the SAS Embedded Process to Run Scoring Models” on page 153

Default STATIC

MODELTABLE=*model-table-name*

specifies the name of the model table where the scoring files are published.

Default	sas_model_table
Restriction	This argument is available only when using the SAS Embedded Process.
Requirement	The name of the model table must be the same as the name specified in the %INDTD_CREATE_MODELTABLE macro. For more

information, see the MODELTABLE argument in “%INDTD_CREATE_MODELTABLE Macro Syntax” on page 155.

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.
	If you use scoring functions to run scoring models, the maximum number of output variables is 128. If you use the SAS Embedded Process and Teradata version 13.1 or 14.0, the maximum is 1024. If you use the SAS Embedded Process and Teradata version 14.10, the maximum is 2048.
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 or the scoring files are published.

Default	The database specified in the INDCONN macro variable or your current database
Requirements	If you use the SAS Embedded Process, the name of the database must be the same as the database specified in the %INDTD_CREATE_MODELTABLE macro. For more information, see the DATABASE argument in “%INDTD_CREATE_MODELTABLE Macro Syntax” on page 155.
	The maximum database name length is 30 characters, and it must be a valid Teradata database name.
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 164.

Tip	You can publish the scoring functions and formats or 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 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 new functions or files.

REPLACE
overwrites the current functions or files, if functions or files with the same name are already registered.

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

Default	CREATE
Tip	If the function or file has been previously defined and you specify ACTION=CREATE, you receive warning messages from Teradata. If the function or file 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
Restriction	This argument is valid only when . It has no effect if you specify MECHANISM=EP.
Tip	After a function is validated in PROTECTED mode, it can be republished in UNPROTECTED mode. This can result in a significant performance gain.

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

Produced files 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 150.

Tip This argument is useful when testing your scoring models.

See "Special Characters in Directory Names" on page 19

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. The SAS Embedded Process always operates in its own process, which is equivalent to fenced mode functions. An optimized data transport mechanism allows the SAS Embedded Process to provide fenced mode protection with speed that is as good as or better than unfenced functions.

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.

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.

If you use scoring functions to run your scoring model, you must have the following permissions on the database where the functions are published:

CREATE FUNCTION

DROP FUNCTION

EXECUTE FUNCTION

ALTER FUNCTION

If you use the SAS Embedded Process to run your scoring model, you must have these permissions:

SELECT, CREATE TABLE, INSERT ON *database* TO *userid*

EXECUTE FUNCTION ON SAS_SYSFNLIB

EXECUTE FUNCTION ON SYSLIB.MonitorVirtualConfig

EXECUTE PROCEDURE ON SAS_SYSFNLIB

The SAS_SCORE_EP procedure runs with access rights of the calling user.

To obtain database permissions, contact your database administrator.

For more information about specific permissions, see "Teradata Permissions" in Chapter 10 of *SAS In-Database Products: Administrator's Guide*.

Chapter 13
SAS Scoring Accelerator and SAS Model Manager

Using the SAS Scoring Accelerator with SAS Model Manager

You can use SAS Scoring Accelerator in conjunction with SAS Model Manager to manage and deploy scoring models in DB2, Greenplum, Hadoop, Oracle, Netezza, SAP HANA, and Teradata.

SAS Model Manager enables you to publish to a configured database the project champion model and challenger models that are associated with the DATA Step score code type. SAS Model Manager uses the SAS Scoring Accelerator and SAS/ACCESS interface to the database to publish models to the database. The Scoring Accelerator takes the models from SAS Model Manager and translates them into scoring files or functions that can be deployed inside the database. After the scoring functions are published using the SAS/ACCESS interface to the database, the functions extend the database’s SQL language and can be used in SQL statements such as other database functions. After the scoring files are published, they are used by the SAS Embedded Process to run the scoring model.

You can also use SAS Model Manager to import SAS/STAT linear models and SAS High-Performance Analytics models from a SAS package file (.SPK), and import PMML models from a PMML model file. Models that have a DATA step score code type can be scored, published, and included in performance monitoring.

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

Part 3

SAS In-Database Code Accelerator

Chapter 14

Using the SAS In-Database Code Accelerator

Overview of the SAS In-Database Code Accelerator

The SAS In-Database Code Accelerator enables you to publish a DS2 thread program to the database and execute that thread program in parallel inside the database. Examples of thread programs include large transpositions, computationally complex programs, scoring models, and BY-group processing.

The SAS In-Database Code Accelerator for Hadoop and the SAS In-Database Code Accelerator for Teradata also enable you to publish and execute the DS2 data program inside the database.

To use the SAS In-Database Code Accelerator, the following requirements must be met. Otherwise, the thread program will be run in multiple threads on the client machine.

- The following products must be licensed at your site:
  - Base SAS
  - SAS In-Database Code Accelerator
  - SAS/ACCESS Interface to your database (Greenplum, Hadoop, or Teradata)
- The SAS Embedded Process must be installed and configured on your database.

  For information about installing and configuring the SAS Embedded Process, see *SAS In-Database Products: Administrator's Guide*.
- Your DS2 code includes a thread program and a data program.
- The table used as input to the thread program must reside in the database.
- Either the PROC DS2 DS2ACCEL option must be set to YES or the DS2ACCEL system option must be set to ANY.

With in-database processing, data is distributed on different data partitions. Each DS2 thread that is running inside the database has access to its own data partition. When doing BY-group processing, each DS2 thread with a BY statement can group and order only the rows in the same data partition. The data partition might have only part of the entire group of data. You would need to do a final aggregation in the main data program. However, if you use the PROC DS2 statement's BYPARTITION=YES option, the entire group of data resides on the same data partition. For more information, see "BY-Group Processing When Running Thread Programs inside the Database" on page 181.

SAS In-Database Code Accelerator for Greenplum

When you use the SAS In-Database Code Accelerator for Greenplum, the thread program and its associated files (format files, packages, and so on) are published to the database. The thread program is executed inside the database, and its result is brought to the data program running on client machine for final processing or aggregation if needed.

SAS In-Database Code Accelerator for Hadoop

Overview

When you use the SAS In-Database Code Accelerator for Hadoop, the data and thread programs run in the MapReduce framework, either MapReduce 1 or YARN/MapReduce 2.

Note: The SAS In-Database Code Accelerator for Hadoop supports only Cloudera 5.2 and Hortonworks 2.1 or later.

Supported File Types

The SAS In-Database Code Accelerator for Hadoop supports these file types:

- Hive: Avro*
- Hive: delimited

- Hive: ORC*
- Hive: Parquet*
- Hive: RCFile*
- Hive: sequence
- HDMD: binary
- HDMD: delimited
- HDMD: sequence
- HDMD: XML

*In the February 2015 release, the SAS In-Database Code Accelerator for Hadoop supports these file types.

TIP Partitioned Avro or Parquet data is not supported as input to the SAS In-Database Code Accelerator for Hadoop.

TIP The availability of these file types depends on the version of Hive that you use.

The following file types are not supported:

- SPD file format
- SASHDAT

Automatic File Compression with SAS Hadoop

By default, the SAS In-Database Code Accelerator for Hadoop automatically compresses certain output files.

The following default file compressions apply to Hive files unless the user has explicitly configured another compression algorithm:

- Delimited files are not automatically compressed.
- ORC files compress themselves using ZLIB.
- Avro, Parquet, and Sequence files are automatically compressed using Snappy.

HDMD files are never automatically compressed.

Using HCatalog within the SAS Environment

HCatalog is a table management layer that presents a relational view of data in the HDFS to applications within the Hadoop ecosystem. With HCatalog, data structures that are registered in the Hive metastore, including SAS data, can be accessed through standard MapReduce code and Pig. HCatalog is part of Apache Hive.

In the February 2015 release, the SAS In-Database Code Accelerator for Hadoop uses HCatalog to process complex, non-delimited files.

Table 14.1 *Summary of HCatalog File I/O*

File Type	Input	Output
delimited	HDFS direct-read HCat if partitioned, skewed, or escaped	HDFS direct-read HCat if partitioned, skewed, or escaped
RCFile	HCat	HCat
ORC	HCat	HCat
Parquet	HCat*	CREATE TABLE AS SELECT**
sequence	HDMD HCat if partitioned or skewed	HCat
Avro	HCat*	CREATE TABLE AS SELECT***

* Partitioned input is not supported.

** Unable to write output directly to Parquet files due to these issues: https://issues.apache.org/jira/browse/HIVE-8838.

*** Unable to write output directly to Avro files due to these issues: https://issues.apache.org/jira/browse/HIVE-8687.

Consider these requirements when using HCatalog:

- Data that you want to access with HCatalog must first be registered in the Hive metastore.
- The recommended Hive version for the SAS In-Database Code Accelerator for Hadoop is 0.13.0.
- Avro is not a native file type. Additional JAR files are required and must be defined in the SAS_HADOOP_JAR_PATH environment variable.
- Support for HCatalog varies by vendor. For more information, see the documentation for your Hadoop vendor.

Additional Prerequisites When Accessing Files That Are Processed Using HCatalog

If you plan to access complex, non-delimited file types such as Avro or Parquet, there are additional prerequisites:

- To access Avro file types, the avro-1.7.4.jar file must be added to the SAS_HADOOP_JAR_PATH environment variable. To access Parquet file types, the parquet-hadoop-bundle.jar file must be added to the SAS_HADOOP_JAR_PATH environment variable.
- If you have performed a default CLASSPATH installation, the HCatalog JAR files that are required to access the Hive metastore within a MapReduce job are automatically included in the Hadoop CLASSPATH. Otherwise, you must manually include the HCatalog JAR files in either the MapReduce library or the Hadoop CLASSPATH.

BY-Group Processing with Hadoop

When there is no BY statement in the thread program, the number of reducers is set to 0, and the program is run as a map-only task. When there is a BY statement in the thread program and the PROC DS2 statement uses the BYPARTITION=YES option, a MapReduce task runs, where the map task partitions the data, and the reducer task runs the DS2 thread program.

Note: The SAS In-Database Code Accelerator for Hadoop might not produce sorted BY groups when re-partitioning is involved.

For more information, see "BY-Group Processing When Running Thread Programs inside the Database" on page 181.

Using the DBCREATE_TABLE_OPTS Table Option

The DBCREATE_TABLE_OPTS table option is used to provide a free form string in the DATA statement. For the SAS In-Database Code Accelerator for Hadoop, you can use the DBCREATE_TABLE_OPTS table option to specify the output SerDe, the output delimiter of the Hive table, the output escaped by, and any other CREATE TABLE syntax allowed by Hive.

For more information, see "DBCREATE_TABLE_OPTS= Table Option" in *SAS DS2 Language Reference*.

SAS In-Database Code Accelerator for Teradata

When you use the SAS In-Database Code Accelerator for Teradata, the data program, the thread program, and their associated files (format files, packages, and so on) are published to the database. Both the data program and the thread program are executed inside the database.

Using the DS2ACCEL Option to Control In-Database Processing

The DS2ACCEL system option controls whether DS2 code is executed inside the database.

In the first maintenance release for SAS 9.4, the default behavior is to run the data and thread programs on the client machine (DS2ACCEL=NONE). You must set either the DS2ACCEL= system option to ANY or the DS2ACCEL= option in the PROC DS2 statement to YES for in-database processing to occur. The DS2ACCEL= option in the PROC DS2 statement overrides the DS2ACCEL system option.

Note: This is a change in behavior from the previous release in which the default value for the PROC DS2 INDB option (now named DS2ACCEL) caused the SAS In-Database Code Accelerator to automatically trigger in-database processing.

For more information, see "DS2ACCEL= System Option" on page 275 and "PROC DS2 Statement" in Chapter 19 of *Base SAS Procedures Guide*.

Considerations and Limitations

- The SAS In-Database Code Accelerator is available only for Greenplum, Hadoop, and Teradata.
- When you use the SAS In-Database Code Accelerator for Greenplum, only the thread program runs inside the database.
- When you use the SAS In-Database Code Accelerator for Hadoop and Teradata, both the data and thread program run inside the database if the output table from the data program resides in Hadoop or Teradata. You can use a different LIBNAME statement for the input and output table if the input and output librefs meet the following conditions:
  - The librefs are on the same Hadoop cluster or in the same Teradata database.
  - For Hadoop, both files must be accessible by Hive, or both files must be accessible in HDFS by means of an HDMD file.
  - When the connection strings are compared, they must be identical in value and case except for these values:
    - CATALOG (Teradata)
    - SCHEMA
    - HDFS_METADIR (Hadoop)
    - HDFS_TEMPDIR (Hadoop)
    - HDFS_PERMDIR (Hadoop)

  If the output table from the data program does not reside in Hadoop or Teradata, only the thread program is run inside the database.
- If the thread program is run inside the database, the number of threads is set by the SAS In-Database Code Accelerator. When this occurs, the THREADS= argument in the SET FROM statement in the data program has no effect.
- When a matrix is declared in a thread program, each thread program has its own, individual instance of a matrix. The DS2 matrix package does not support data partitioning between nodes or threads to perform parallel matrix operations. Instead, each thread performs the matrix operations on its own instance of the matrix.
- The DS2 program fails if you try to use an empty format that you defined with PROC FORMAT.
- Only one SET statement is allowed when using the SAS In-Database Code Accelerator. If more than one SET statement is used in the thread program, the thread program is not run inside the database. Instead, the thread program runs on the client.
- Thread and data programs that use packages are supported. However, using a HASH, HTTP, or SQLSTMT package causes the thread program to run on the client and not inside the database.
- In-database processing does not occur when the following methods are used to load data. Instead, the data and thread programs are run on the client.
  - Using a SET statement with embedded SQL code
  - Using an SQLSTMT package

- Using an initialized hash package
- Using an HTTP package

- Only one input table is allowed in the SET statement. If more than one input table is used in the SET statement, the thread program is not run inside the database. Instead, the thread program runs on the client.
- Using an unrecognized catalog in the SET statement causes the thread program to run on the client.
- If you use a HAVING clause to format output column data, the format is not applied to the output column data when the data is written back to a file. The format is specified in the output column metadata. The SAS/ACCESS Engine for Hadoop is currently unable to understand column format. Therefore, PROC PRINT or PROC CONTENTS do not print or display the contents with the format specified in the column's metadata.
- A Hive STRING data type is always converted to a VARCHAR data type using the following rules:
  - STRING -> VARCHAR(65355)
  - STRING + SASFMT:CHAR(*n*) -> VARCHAR(*n*)
  - STRING + SASFMT:VARCHAR(*n*) -> VARCHAR(*n*)
  - STRING + DBMAX_TEXT -> VARCHAR(DBMAX_TEXT)
- When using SAS Code Accelerator for Hadoop, the Hive user needs Read and Write access to the TempDir and the Destination Warehouse directories. In addition, the MapReduce user needs Read and Write permission.

BY-Group Processing When Running Thread Programs inside the Database

DS2 BY-group processing groups the rows from input tables and orders the rows by values of one or more columns in the BY statement.

With in-database processing, data is distributed on different data partitions. Each DS2 thread running inside the database has access to one data partition. Each DS2 thread can group and order only the rows in the same data partition. Consequently, the data partition might have only part of the entire group of data. You must do a final aggregation in the main data program.

But, in some instances, it is necessary for each thread to process the entire group of data. The SAS In-Database Code Accelerator provides a way to redistribute the input table to the thread program with a BY statement so that the entire group of data resides on the same data partition.

The PROC DS2 statement BYPARTITION argument controls whether the input data is re-partitioned. By default, the input data for the DS2 program is automatically re-partitioned by the first BY variable. All of the BY groups are in the same data partition and processed by the same thread. Each thread does the BY processing for the entire group of data. You might not need to do the final aggregation in the main data program.

For more information, see “BY-Group Processing with the SET Statement” in Chapter 19 of *SAS DS2 Language Reference*, and the DS2 procedure in *Base SAS Procedures Guide*.

SAS In-Database Code Accelerator Examples

Example 1: Running a Thread inside the Database

The following is an example of a DS2 program whose data and thread programs are published and executed in database through the SAS In-Database Code Accelerator. The results from the thread program are processed by the data program inside the database.

```
options ds2accel=any;

libname teralib teradata server=terapin database=xxxxxx
   user=xxxxxx password=xxxxxx;

data teralib.indata;
   do i = 1 to 10;
      output;
   end;
run;

proc ds2;

thread th_pgm / overwrite=yes;
   retain isum 0;
   keep isum;
   dcl double x isum;

   method run();
      set teralib.indata;
            x=i+1;
            isum=isum+i;
            end;

   method term();
      output;
    end;

endthread;
run;

data out(overwrite=yes);
   retain fsum 0;
   retain nrows 0;
   keep fsum nrows;

   dcl thread th_pgm m;
   method run();
      /* The THREADS= argument in the SET FROM statement has no effect */
      /* if the SAS In-Database Code Accelerator is used to access a */
      /* database table. */
      set from m threads=1;
      fsum =fsum + isum;
      nrows = nrows + 1;
```

```
   end;

   method term();
      output;
   end;
enddata;
run;
quit;
```

Example 2: Using User-Defined Formats

The following example uses formats that are defined in PROC FORMAT. Those formats that are referred to in the thread program are used to create an XML file. In addition to the data and programs, the format XML file is published to the database. The format XML file is used when running DS2 inside the database.

```
options ds2accel=any;

libname teralib teradata server=terapin database=xxxxxx
   user=xxxxxx password=xxxxxx;

%let libname=teralib;

data &libname..indata_fmt;
   do i = 1 to 10;
   output;
end;
run;

proc format;
   value yesno 1='YES' 0='NO';
run;

proc format;
   value $x '1'='YES' '0'='NO';
run;

proc ds2;
drop thread th_pgm; run;

thread th_pgm;
   dcl double x;
   dcl char z w;
   method run();
      set &libname..indata_fmt;
      x=i+1;
      z=put(1, yesno.);
      w=put('0', $x.);
   end;
endthread;
run;

data out (overwrite=yes);
   dcl thread th_pgm m;
   method run();
```

```
      dcl double y;
      /* The THREADS= argument in the SET FROM statement has no effect */
      /* if the SAS In-Database Code Accelerator is used to access a */
      /* database table. It could have been omitted from the SET FROM*/
      /* statement. */
      set from m threads=10;
      y=x+1;
   end;
enddata;
run;
quit;
```

The following output table is produced.

Output 14.1 *Result Table for Example 2*

x	z	w	i
2	YES	NO	1
3	YES	NO	2
4	YES	NO	3
5	YES	NO	4
6	YES	NO	5
7	YES	NO	6
8	YES	NO	7
9	YES	NO	8
10	YES	NO	9
11	YES	NO	10

Example 3: Using User-Defined Formats and Packages

The following example uses user-defined formats and user-defined DS2 packages. In addition to the data and thread programs, the user-defined formats and the user-defined DS2 packages are published to the database.

```
options ds2accel=any;

libname db teradata user=XXXX password=XXXX
   server=terapin database=XXXX;

proc ds2;
data db.ipassdata / overwrite=yes;
   declare double  score;
   method init();
```

```
      declare int i;
      do i = 1 to 20;
         score = i * 5;
         output;
      end;
   end;
enddata;
run;
quit;

proc format;
   value lettergrade
      90-high = 'A'
      80-89   = 'B'
      70-79   = 'C'
      60-69   = 'D'
      low-59  = 'F';
run;

proc format;
   value passfail
      70-high = 'PASS'
      low-69  = 'FAIL';
run;

proc ds2;
package pkgGrade;
   method compute(double s) returns char(1);
      declare char(1) g;
      g = put(s, lettergrade.);
      return g;
   end;
endpackage;

package pkgPassFail;
   method compute(double s) returns char(4);
      declare char(4) g;
      g = put(s, passfail.);
      return g;
   end;
endpackage;

thread th_pgm;
    declare char(1) grade;
    declare char(4) pass;
    declare package pkgGrade g();
    declare package pkgPassFail pf();

    method run();
       set db.ipassdata;
       grade = g.compute(score);
       pass  = pf.compute(score);
    end;
endthread;

data outdata;
```

```
   dcl thread th_pgm m;
   method run();
      /* The THREADS= argument in the SET FROM statement has no effect */
      /* if the SAS In-Database Code Accelerator is used to access a */
      /* database table. */
      set from m threads=1;
   end;
enddata;
run;
quit;

proc print data=outdata; quit;
```

The following output table is produced.

Output 14.2 *Result Table for Example 3 (Partial Output)*

The SAS System

grade	pass	score
F	FAIL	5
F	FAIL	10
F	FAIL	15
F	FAIL	20
F	FAIL	25
F	FAIL	30
F	FAIL	35
F	FAIL	40
F	FAIL	45
F	FAIL	50
F	FAIL	55
D	FAIL	60
D	FAIL	65
C	PASS	70
C	PASS	75

Example 4: BY-Group Processing

The following example transposes customer data that has multiple records for each customer into one wide record for each customer. The SAS In-Database Code

Accelerator for Teradata redistributes the input data **`pivot_1m`** by the first BY variable **`Cust_Name6`**. All the rows with the same **`Cust_Name`** will be on the same data partition and transposed by one thread.

```
options ds2accel=any;

%let nobs=1000000;
libname td teradata server=terapin user=xxxxxx password=xxxxxx database=xxxxxx;

proc delete data=td.pivot_1m; run;
data td.pivot_1m (tpt=no fastload=yes dbcommit=100000);
   drop i;
   length Cust_Name $20;
   do i = 1 to &noobs;
       month_id = floor(rand('Uniform')*12)+1;
       month_visits = floor(rand('Uniform')*1000)+1;
       month_amount = (floor(rand('Uniform')*1000000)+1)/100;
       Cust_Name = "Name"||strip(mod(i,1000));
       output;
   end;
run;

%let inputdata=td.pivot_1m;

proc ds2;
   thread work.p_thread / overwrite=yes;
      dcl double i;
      vararray double amount[12];
      vararray double num_visits[12];
      keep Cust_Name amount1-amount12 num_visits1-num_visits12;
      retain amount1-amount12 num_visits1-num_visits12;
      method clear_array();
         do i=1 to 12 ;
         amount[i] = 0;
         num_visits[i] = 0;
       end;
   end;
   method run();
      set &inputdata;
       by Cust_Name;
       if first.Cust_Name then
          clear_array();
          amount[month_id] = month_amount + amount[month_id];
          num_visits[month_id] = month_visits + num_visits[month_id];
       if last.Cust_Name then
          output;
   end;
endthread;
run;

data td.pivot_results (overwrite=yes);
   dcl thread p_thread p;
   method run();
      set from p;
      output;
   end;
```

```
enddata;
run;
quit;
```

The following output table is produced (partial output).

Output 14.3 *Result Table for Example 4 (Partial Output)*

amount1	amount2	amount3	amount4	amount5	amount6
0.00	0.00	0.00	3942.11	0.00	0.00
0.00	7112.52	0.00	0.00	0.00	0.00
0.00	0.00	0.00	8531.73	0.00	0.00
0.00	0.00	0.00	0.00	0.00	4189.85
628.50	0.00	0.00	0.00	0.00	0.00
0.00	5436.21	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00

Part 4

In-Database DATA Step Processing

Chapter 15

DATA Step Processing in Hadoop

DATA Step Processing in Hadoop

In order to accelerate DATA step processing of data based in Hadoop, the DATA step has been enhanced to determine when the user code is appropriate for exporting to the Hadoop MapReduce facility. If you have installed and activated the SAS Embedded Process on a Hadoop cluster, it is possible for DATA step code to be executed in parallel against the input data residing on the HDFS file system.

Because of the single-source, shared-nothing nature of MapReduce processing and the immutable nature of HDFS files, only a subset of the full DATA step syntax can be passed through for parallel execution. The DATA step can be run inside Hadoop for scoring with the following limitations:

- Only one input file and one output file are allowed.
- The input file and output file are in Hadoop.
- Only functions and formats that are supported by the DS2 language compile successfully.
- Some DATA step statements are not allowed, such as those pertaining to input and output.

For more information, see "Requirements for DATA Step Processing" on page 192 and "Restrictions in DATA Step Processing" on page 192.

To enable the DATA step to be run inside Hadoop, set the DSACCEL= system option to ANY.

If a SAS program does not meet the requirements for running in Hadoop, the code executes in your Base SAS session. In this case, SAS reads and writes large tables over the network.

You can determine whether your code is non-compliant for Hadoop by setting the system option MSGLEVEL=I. When MSGLEVEL=I, SAS writes log messages that identify the non-compliant code.

Requirements for DATA Step Processing

In order to run a DATA step program in Hadoop, the following is required:

- The DSACCEL= system option is set to ANY.

  For more information about the DSACCEL= system option, see *SAS System Options: Reference*.

- The code must contain a LIBNAME statement using the SAS/ACCESS HADOOP engine.

  For more information about the Hadoop LIBNAME statement, see *SAS/ACCESS for Relational Databases: Reference*.

- The input and output files must use the same libref for the HADOOP engine.
- The DATA statement must be followed immediately by the SET statement.

  This example demonstrates these requirements:

  ```
  options dsaccel=any;

  libname hdone hadoop;
  data hdone.out;
  set hdone.in;
  /* DATA step code */
  run;
  ```

- The SAS Embedded Process must be running on the cluster where the input and output files exist.

  For more information, see “Determining the Status of the SAS Embedded Process” in Chapter 5 of *SAS In-Database Products: Administrator's Guide* and “Starting the SAS Embedded Process” in Chapter 5 of *SAS In-Database Products: Administrator's Guide*.

Restrictions in DATA Step Processing

Here are the restrictions for using the DATA step in Hadoop:

- More than one SET statement is not supported.
- These statements are not supported:
  - BY (or FIRST. and LAST. variables)
  - CONTINUE
  - DISPLAY
  - FILE
  - INIFILE
  - INPUT
  - LEAVE

- MERGE
- MODIFY
- OUTPUT
- PUT
- REMOVE
- RENAME
- REPLACE
- RETAIN
- UPDATE
- WHERE
- WINDOW
- The ABORT statement has these restrictions:
  - The ABORT statement does not accept arguments.
  - The ABORT statement is not supported within functions. It is valid only in the main program.
-
- The sub-setting IF statement is not supported.
- The INPUT function does not support the question mark (?) and double question mark (??) modifiers.
- No SET statements options are allowed.
- You can use only SAS formats and functions that are supported by the DS2 language. For more information, see *SAS DS2 Language Reference*.
- Some CALL routines are not supported. Routines are supported if there is an equivalent function.
- Component objects are not supported.
- Scoring input variables cannot be modified.
- Large models can consume large amounts of memory on the client side. It is recommended that you set the MEMSIZE= system option to MAX.

Example: DATA Step Program for Hadoop

This example demonstrates executing a DATA step program in Hadoop.

```
/* Enable DATA step parallel processing using the system option */
/* and enable messages to view non-compliant code in the SAS log */
options dsaccel=any msglevel=i;

/* Create a libref for Hadoop files */
libname griddlm hadoop user=myuser pw=hd12345
   HDFS_TEMPDIR="/user/temp"
   HDFS_DATADIR="/userdata"
   HDFS_METADIR="/user/meta"
```

```
   config="C:\sasuser\scoring\hd1\conf\testconfig.xml"
   HPA_TEMPDIR_KEEP=YES;

/* Create a libref for the input data that is stored on disk. */
libname y '/myScoreData/';

/* Load the input table*/
data griddlm.intr;
   set y.intrid;
run;

/* Execute the score code using the Hadoop files. */
/* Both files must use the same libref. */
data griddlm.introut3;
   set griddlm.intr;
   /* Execute the score code. */
   if sum > 1000
      then score=1;
run;
```

Part 5

Format Publishing and the SAS_PUT( ) Function

Chapter 16

Deploying and Using SAS Formats inside the Database

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. It then 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 Aster, DB2, Greenplum, Netezza, and Teradata.

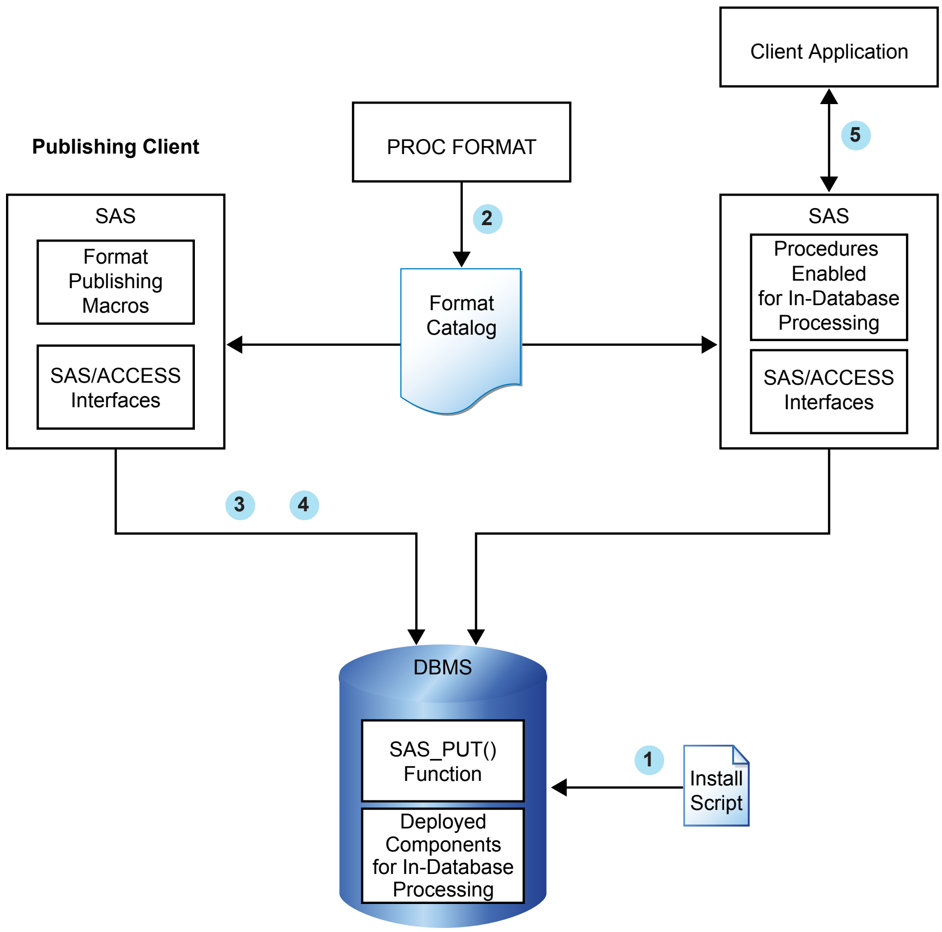
How It Works

By using the SAS formats publishing macro for DB2, Greenplum, Netezza, and Teradata, 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.

For Aster, the SAS_PUT( ) function is installed as part of the SAS Embedded Process. For more information, see the *SAS In-Database Products: Administrator's Guide*.

Figure 16.1 *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 5.

   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 and run the format publishing macro. For DB2, Greenplum, Netezza, and Teradata, 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 your database 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.

Format Publishing with User-Defined Functions and the SAS Embedded Process

There are two methods by which format publishing is processed inside the database:

- user-defined functions

  Formats are converted by the publishing macros into format functions that are similar to any user-defined functions in the database.

  In-database processing of formats by means of user-defined functions is supported by DB2 under UNIX, Greenplum, Netezza, and Teradata.

- SAS Embedded Process

  The SAS Embedded Process is a SAS server process that is installed and runs inside the database to read and write data from the database. The advantage of using the SAS Embedded Process is that a single function or a stored procedure is used instead of multiple, user-defined functions.

  Format publishing using the SAS Embedded Process is supported for Aster, DB2, Greenplum, Netezza, and Teradata.

  The SAS Embedded Process is one of the deployed components for in-database processing. For more information, see the *SAS In-Database Products: Administrator's Guide*.

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 16.1 *Special Characters in Directory Names*

Character	How to Represent
blank*	%str()
***	%str(*)
;	%str(;)
, (comma)	%str(,)

Character	How to Represent
=	%str(=)
+	%str(+)
-	%str(–)
>	%str(>)
<	%str(<)
^	%str(^)
|	%str(|)
&	%str(&)
#	%str(#)
/	%str(/)
~	%str(~)
%	%str(%%)
'	%str(%')
"	%str(%")
(	%str(%()
)	%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 16.2 *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`

Directory	Code Representation
`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 and not 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.
- Avoid using PICTURE formats with the MULTILABEL option. You cannot successfully create a CNTLOUT= data set when PICTURE formats are present. This is 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 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 280 and the "SQLREDUCEPUT= System Option" on page 281.
- 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 17
Deploying and Using SAS Formats in Aster

User-Defined Formats in the Aster Database

Introduction to User-Defined Formats in Aster

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

For more information about the %INDAC_PUBLISH_FORMATS macro, see "Publishing SAS Formats in Aster" on page 206. For more information about the SAS_PUT( ) function, see "Using the SAS_PUT( ) Function in the Aster Database" on page 213.

Aster Limitations and Restrictions When Using the FMTCAT= Option

Formats as labels and the DATATYPE= option cannot be used with formats that are exported to Aster.

Publishing SAS Formats in Aster

Overview of the Publishing Process

The SQL/MR function is the framework for enabling execution of user-defined functions within Aster through an SQL interface. A SAS SQL/MR function, SAS_PUT( ), supports format publishing in Aster. The SAS_PUT( ) function is installed as part of the in-database deployment package. For more information, see the *SAS In-Database Products: Administrator's Guide*.

The %INDAC_PUBLISH_FORMATS macro creates the user-defined format files that are needed by the SAS_PUT( ) function and publishes those files to the Aster database.

This macro makes many formats that SAS supplies available inside Aster. In addition to formats that SAS supplies, you can use the FMTCAT= option to publish the PROC FORMAT definitions that are contained in a single SAS format catalog. The process of publishing a PROC FORMAT catalog entry converts the *value-range-set(s)* into embedded data in Aster.

The %INDAC_PUBLISH_FORMATS macro performs the following tasks:

- takes the format catalog and produces a sasput_*type_fmtname*.xml file for each user-defined format that is in the format catalog
- uses the SAS/ACCESS Interface to Aster to insert the format files into either the NC_INSTALLED_FILES table under the PUBLIC schema (Aster 4.5) or the NC_USER_INSTALLED_FILES table under a specified schema (Aster 4.6)

Note: Files larger than 32k are automatically divided into 32k chunks of data and then are concatenated back together by performing multiple updates.

Note: If there are no user-defined formats, you do not need to run the %INDAC_PUBLISH_FORMATS macro. The formats that SAS supplies are installed in either the NC_INSTALLED_FILES table (Aster 4.5) or the NC_USER_INSTALLED_FILES table (Aster 4.6) when the SAS Formats Library for Aster is installed.

When the user accesses a SAS format through the SQL interface, the SAS_PUT( ) function retrieves the specified format's XML file and activates the SAS Embedded Process to perform the formatting. For more information, see "Using the SAS_PUT( ) Function in the Aster Database" on page 213.

Running the %INDAC_PUBLISH_FORMATS Macro

To run the %INDAC_PUBLISH_FORMATS macro, follow these steps.

1. Start SAS and submit this command in the Program Editor or the Enhanced Editor:

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

For more information, see the "INDCONN Macro Variable" on page 207.

2. Run the %INDAC_PUBLISH_FORMATS macro.

   For more information, see "%INDAC_PUBLISH_FORMATS Macro Syntax" on page 208.

Messages are written to the SAS log that indicate the success or failure of the creation of the XML format files.

INDCONN Macro Variable

The INDCONN macro variable is used to provide credentials to connect to Aster. 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_FORMATS macro is invoked.

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

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

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

USER=*username*
specifies the Aster 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 user ID.

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

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

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

DATABASE=*databasename*
specifies the Aster 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 in the INDCONN macro variable.

SERVER=*servername*
specifies the Aster 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 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 4.6. For Aster 4.5, the format XML files are published to the PUBLIC schema.
Requirement	Any schema that is used must be in the search path.

TIP The INDCONN macro variable is not passed as an argument to the %INDAC_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.

%INDAC_PUBLISH_FORMATS Macro Syntax

%INDAC_PUBLISH_FORMATS

(<DATABASE=*database–name*>
<, FMTCAT=*format-catalog-filename* | ALL>
<, FMTLIST=*format-name* <…*format-name*> | ALL>
<, ACTION=CREATE | REPLACE | DROP>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DATABASE=*database-name*

specifies the name of an Aster database to which the format files are published to either the NC_INSTALLED_FILES table (Aster 4.5) or the NC_USER_INSTALLED_FILES table (Aster 4.6). This argument lets you publish the sasput_*type_fmtname*.xml format files to a shared database where other users can access them.

Restriction	If you specify DSN= in the INDCONN macro variable, do not use the DATABASE argument. For more information, see "Running the %INDAC_PUBLISH_FORMATS Macro" on page 206.
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* | ALL

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 Aster.

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 Aster.
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 202

FMTLIST=*format-name* <…*format-name*> | ALL

specifies a list of formats that are created, replaced, or dropped.

Default	ALL
Requirements	Format names must be separated with a space.
	Character format names must begin with a dollar sign ($); for example, $EMPNAME.
Interaction	When ACTION=CREATE or REPLACE, the list of formats that are in the specified format catalog (FMTCAT=) are added to either the NC_INSTALLED_FILES table (Aster 4.5) or the NC_USER_INSTALLED_FILES table (Aster 4.6). When ACTION=DROP and FMTCAT=ALL, all the formats listed in FMTLIST are dropped. If ACTION=DROP and FMTCAT=*format-catalog-filename*, only those listed formats that exist in the format catalog are dropped.

ACTION=CREATE | REPLACE | DROP

specifies that the macro performs one of these actions:

CREATE

creates a sasput_*type_fmtname*.xml file for each user-defined format in the format catalog.

Tip If a format file already exists, an error occurs.

REPLACE

overwrites the current sasput_*type_fmtname*.xml file if it is already registered or creates a new sasput_*type_fmtname*.xml file, if one is not registered.

DROP

causes the sasput_*type_fmtname*.xml files to be dropped from either the NC_INSTALLED_FILES table (Aster 4.5) or the NC_USER_INSTALLED_FILES table (Aster 4.6) in the database.

Interaction If FMTCAT=ALL, all user-defined format files are dropped.

Default	CREATE
Tip	If the format files was defined previously and you specify ACTION=CREATE, you receive warning messages from Aster. If the format files were defined previously and you specify ACTION=REPLACE, a message is written to the SAS log indicating that the format file has been replaced.

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 200

Format Publishing Macro Example

```
%let indconn = server=acbase user=user1 password=open1 dsn=ncluster;
   %indac_publish_formats(fmtcat= fmtlib.formats);
```

This sequence of macros generates an XML file for each format. The format data types that are supported are numeric and character. The naming convention for the XML file is sasput_*type*_*fmtname*.xml, where *type* is the format data type (N for numeric formats or C for character formats), and *fmtname* is the format name.

After the format files are installed, you can invoke user-defined formats in Aster by using the SAS_PUT( ) function. For more information, see "Using the SAS_PUT( ) Function in the Aster Database" on page 213.

Aster Format Files

Overview of Aster Format Files

The %INDAC_PUBLISH_FORMATS macro produces a format file for each user-defined format in the format catalog. These files are inserted into either the NC_INSTALLED_FILES table under the PUBLIC schema (Aster 4.5) or the NC_USER_INSTALLED_FILES table under a specified schema (Aster 4.6). The naming convention for the file is **sasput_*type*_*fmtname*.xml**, where *type* is the format data type (N for numeric formats or C for character formats), and *fmtname* is the format name.

For an example, see "Example of a Format File" on page 211.

There are three ways to see the format files that are created:

- You can log on to the database using the Aster command line processor and submit an SQL statement. The following example assumes that three format files were created in Aster 4.6.

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

  All the format files are listed:

  ```
  name
  -------------------------
  sasput_n_dinar.xml
  sasput_n_ruble.xml
  sasput_c_lowcase.xml
  ```

- From SAS, you can use SQL procedure code that produces output in the LST file.

  ```
  proc sql noerrorstop;
     connect to aster (user=username password=password dsn=dsnname schema=schemaname);
  select *
     from connection to aster
     (select filename, fileowner, uploadtime
     from schemaname.nc_user_installed_files where
     filename like 'sasput_%');
     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*.

- You can look at the SAS log. A message that indicates whether the format files are successfully or not successfully created is printed to the SAS log.

Example of a Format File

Here is an example of an Aster format file. This is a partial listing.

```
<?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>9.02.02M0P01152009</VersionLong>
      <CreationDateTime>2009-11-13T15:19:55</CreationDateTime>
   </HEADER>

   <TABLE name="N_DIVFMT">
      <TABLE-HEADER>
         <Provider>SAS Institute Inc.</Provider>
         <Version>9.2</Version>
         <VersionLong>9.02.02M0P01152009</VersionLong>
         <CreationDateTime>2009-11-13T15:19:55</CreationDateTime>
         <ModifiedDateTime>2009-11-13T15:19:55</ModifiedDateTime>

         <Protection />
         <DataSetType />
         <DataRepresentation />
         <Encoding>UTF-8</Encoding>
         <ReleaseCreated />
         <HostCreated />
         <FileName>c:\jaseco\tmp\SASWORK\920\_TD22220\#LN00024</FileName>

         <Observations length="187" />
         <Compression compressed="No" number="1" length="252" />
         <Variables number="21" />
      </TABLE-HEADER>

      <COLUMN name="FMTNAME" order="1" label="Format name">
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>32</LENGTH>
         <Offset>0</Offset>
      </COLUMN>

      <COLUMN name="START" order="2" label="Starting value for format">
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>16</LENGTH>
         <Offset>32</Offset>
      </COLUMN>
```

```
<COLUMN name="END" order="3" label="Ending value for format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>16</LENGTH>
   <Offset>48</Offset>
</COLUMN>

<COLUMN name="LABEL" order="4" label="Format value label">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>21</LENGTH>
   <Offset>64</Offset>
</COLUMN>

<COLUMN name="MIN" order="5" label="Minimum length">
   <TYPE>numeric</TYPE>
   <DATATYPE>float</DATATYPE>
   <Offset>85</Offset>
</COLUMN>

... <more column definitions> ...

<ROW>
   <DELTA-RECORD key="DIVFMT" />
   <FMTNAME>DIVFMT</FMTNAME>
   <START>1</START>
   <END>1</END>
   <LABEL>New England</LABEL>
   <MIN>1</MIN>
   <MAX>40</MAX>
   <DEFAULT>15</DEFAULT>
   <LENGTH>15</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>
   <FMTNAME>DIVFMT</FMTNAME>
   <START>2</START>
   <END>2</END>
   <LABEL>Middle Atlantic</LABEL>
   <MIN>1</MIN>
   <MAX>40</MAX>
   <DEFAULT>15</DEFAULT>
   <LENGTH>15</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>

         ... <more row definitions>...

      </TABLE>
   </LIBRARY>
```

Using the SAS_PUT( ) Function in the Aster Database

Overview of the SAS_PUT( ) Function

The SAS_PUT( ) function executes the format files using the SAS Embedded Process in Aster. The SAS_PUT( ) function is installed in the NC_INSTALLED_FILES table under the PUBLIC schema. For more information, see the *SAS In-Database Products: Administrator's Guide*.

The SAS_PUT( ) function is available to use in the SELECT clause in any SQL expression in the same way that Aster SQL/MR functions are used.

This is the syntax of the SAS_PUT( ) function.

SELECT SAS_PUT(*value* , '*fmtname*') FROM *input-table*;

Arguments

value
specifies the name of the value that the format is applied to.

fmtname
specifies the name of the format.

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

Implicit Use of the SAS_PUT( ) Function

Mapping PUT Function Calls to SAS_PUT( )

After you install the SAS_PUT( ) function and formats that SAS supplies in libraries inside the Aster database, and after you 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 Aster understands.

This example illustrates how the PUT function is mapped to the SAS_PUT( ) function using implicit pass-through. The SELECT DISTINCT clause executes inside Aster, and the processing is distributed across all available data nodes. Aster formats the price values with the $DOLLAR8.2 format and processes the SELECT DISTINCT clause using the formatted values.

```
options sqlmapputto=sas_put;

libname dblib aster 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 aster user="sas" password="sas" server="sl96208"
   database=sas connection=shared;

NOTE: Libref DBLIB was successfully assigned, as follows:
      Engine:        ASTER
      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
                ;
```

```
ASTER_0: Prepared: on connection 0
SELECT * FROM sas."mailorderdemo"


ASTER_1: Prepared: on connection 0
 select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
 as char(8)) as "PRICE_C" from "sas"."mailorderdemo"

ASTER: trforc: COMMIT WORK
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.

ASTER_2: Executed: on connection 0
 select distinct cast(sas_put("sas"."mailorderdemo"."PRICE", 'DOLLAR8.2')
 as char(8)) as "PRICE_C" from "sas"."mailorderdemo"

ASTER: trget - rows to fetch: 9
ASTER: trforc: COMMIT WORK

               Test SAS_PUT using Implicit Passthru                        9
                                         3:42 Thursday, April 25, 2013

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

         quit;
```

Considerations with Implicit Use of SAS_PUT( )

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT. This ensures that the SQL processor maps your PUT functions to the SAS_PUT( ) function and that the SAS_PUT( ) reference is passed through to Aster. SAS_PUT is the default value for the SQLMAPPUTTO= system option.
- 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 an Aster VARCHAR(*n*) is defined to be a null-terminated string.

Explicit Use of the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in an SQL Query

If you use explicit pass-through (direct connection to Aster), 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 214 and explicitly uses the SAS_PUT( ) function call.

```
proc sql noerrorstop;
   title1 'Test SAS_PUT using Explicit Passthru;
   connect to aster(user=sas password=XXX database=sas server=sl96208);

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

disconnect from aster;
quit;
```

The following lines are written to the SAS log.

```
 proc sql noerrorstop;
 title1 'Test SAS_PUT using Explicit Passthru ';
 connect to aster(user=sas password=XXX database=sas server=sl96208);

 select * from connection to aster
    (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, April 25, 2013

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

disconnect from aster;
quit;
```

Considerations with Explicit Use of SAS_PUT( )

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;
```

Aster Permissions

For Aster 4.5, the person who runs the format publishing macros needs no permissions, because all functions and files are published to the PUBLIC schema.

For Aster 4.6, the person who runs the format publishing macros needs the following permissions, 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_FORMATS 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*.

Chapter 18

Deploying and Using SAS Formats in DB2 under UNIX

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 219. For more information about the SAS_PUT( ) function, see "Using the SAS_PUT( ) Function in the DB2 Database" on page 226.

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 *value-range-set(s)* 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 you can 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 and submit this command in the Program Editor or the Enhanced Editor:

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

   For more information, see the “INDCONN Macro Variable” on page 221.

2. Run the %INDB2_PUBLISH_FORMATS macro.

   For more information, see “%INDB2_PUBLISH_FORMATS Macro Syntax” on page 222.

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

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 how the host name was cached when SFTP *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 SFTP 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 SFTP command, see “DB2 Installation and Configuration Steps” in Chapter 3 of *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 the SERVERUSERID (SFTP user ID) and the private key that need to be added to the pageant.exe (Windows) or SSH agent (UNIX). In order for the SFTP process to be successful, Pageant must be running on Windows, and the SSH agent must be running on UNIX.

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=FENCED | UNFENCED>
<, 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 220.

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 are 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 202

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 format table contains the columns shown in the following table.

Table 18.1 *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 batch-file 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 batch-file 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 batch-file 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 200

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

```
%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 226.

Using the SAS_PUT( ) Function in the DB2 Database

Implicit Use of the SAS_PUT( ) Function

Mapping PUT Function Calls to SAS_PUT( )

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. The SELECT DISTINCT clause executes inside DB2, and the processing is distributed across all available data nodes. DB2 formats the sales values with the $DOLLAR12.2 format and processes the SELECT DISTINCT clause using the formatted values.

```
%let mapconn=user=sas1 password=sas31 database=indb;
libname dblib db2 &mapconn;

data dblib.shoes;
set sashelp.shoes;
run;

options sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
title 'Test SAS_PUT using Implicit PassThru/LIBNAME ';
select distinct
       PUT(SALES, Dollar8.2)AS SALES_C from dblib.SHOES;
quit;
```

These lines are written to the SAS log.

```
1726   options sastrace=',,,d' sastraceloc=saslog;
1727
1728   proc sql noerrorstop;
1729   title 'Test SAS_PUT using Implicit PassThru/LIBNAME ';
1730   select distinct
1731          PUT(SALES, Dollar8.2)AS SALES_C from dblib.SHOES;
DB2: AUTOCOMMIT turned ON for connection id 0 1854 1309265953 setconlo 0 SQL
  1855 1309265953 du_prep 0 SQL
DB2_363: Prepared: on connection 0 1856 1309265953 du_prep 0 SQL
SELECT * FROM SHOES FOR READ ONLY  1857 1309265953 du_prep 0 SQL
  1858 1309265953 du_prep 0 SQL
DB2: COMMIT performed on connection 0. 1859 1309265953 du_comm 0 SQL
  1860 1309265953 du_prep 0 SQL
DB2_364: Prepared: on connection 0 1861 1309265953 du_prep 0 SQL
 select distinct cast(SAS_PUT(TXT_1."SALES", 'DOLLAR8.2') as char(8))
      as SALES_C from SHOES TXT_1
```

```
FOR READ ONLY  1862 1309265953 du_prep 0 SQL
  1863 1309265953 du_prep 0 SQL
DB2: COMMIT performed on connection 0. 1864 1309265953 du_comm 0 SQL
  1865 1309265953 du_exec 0 SQL
DB2_365: Executed: on connection 0 1866 1309265953 du_exec 0 SQL
Prepared statement DB2_364 1867 1309265953 du_exec 0 SQL
  1868 1309265953 du_exec 0 SQL
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
    1869 1309265953 fetch 0
SQL
1732   quit;
```

Considerations with Implicit Use of SAS_PUT( )

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT. This ensures that the SQL processor maps your PUT functions to the SAS_PUT( ) function and that 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("dblib"."shoes"."SALES", 'DOLLAR12.2')
     as char(12)) as "SALES_C" from "dblib"."shoes"
  ```

  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.

Explicit Use of the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in an SQL Query

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 226 and explicitly uses the SAS_PUT( ) function call.

```
%let mapconn=user=sasts password=xxxx database=indb;
libname dblib db2 &mapconn;

data dblib.shoes;
set sashelp.shoes;
run;

options sastrace=',,,d' sastraceloc=saslog;

proc sql noerrorstop;
title 'Test SAS_PUT using Explicit Passthru ';
connect to db2 (user=sas3 password=sas31 database=indb);
select * from connection to db2
  (select distinct (sas_put("SALES",'DOLLAR12.2')) as "SALES_C" from SHOES);
disconnect from db2;
```

```
quit;
```

The following lines are written to the SAS log.

```
1733
1734   proc sql noerrorstop;
1735   title 'Test SAS_PUT using Explicit Passthru ';
1736   connect to db2 (user=db2 password=XXXXXXXXXXXX database=indb);
DB2: AUTOCOMMIT is YES for connection 4 1870 1309265953 ducon 0 SQL
1737   select * from connection to db2
1738     (select distinct (sas_put("SALES",'DOLLAR12.2')) as "SALES_C" from
SHOES);
  1871 1309265953 du_prep 0 SQL
DB2_366: Prepared: on connection 4 1872 1309265953 du_prep 0 SQL
select distinct (sas_put("SALES",'DOLLAR12.2')) as "SALES_C" from SHOES 1873
1309265953 du_prep 0
SQL
  1874 1309265953 du_prep 0 SQL
DB2: COMMIT performed on connection 4. 1875 1309265953 du_comm 0 SQL
  1876 1309265953 du_exec 0 SQL
DB2_367: Executed: on connection 4 1877 1309265953 du_exec 0 SQL
Prepared statement DB2_366 1878 1309265953 du_exec 0 SQL
  1879 1309265953 du_exec 0 SQL
1739   disconnect from db2;
1740   quit;
```

Considerations with Explicit Use of SAS_PUT( )

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("sales", 'dollar12.2') as char(12)) as "sales_c",
   from shoes;
```

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 19

Deploying and Using SAS Formats in Greenplum

User-Defined Formats in the Greenplum Database

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

For more information about the %INDGP_PUBLISH_FORMATS macro, see “Publishing SAS Formats in Greenplum” on page 231. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in Greenplum” on page 236.

Publishing SAS Formats in Greenplum

Overview of the Publishing Process

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

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

This macro also makes many formats that SAS supplies available inside Greenplum. In addition to formats that SAS supplies, you can 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 *value-range-set(s)* into embedded data in Greenplum.

The %INDGP_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 Greenplum commands that are necessary to register the SAS_PUT( ) function in 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 publish the SAS_PUT( ) function to the Greenplum database

The SAS_PUT( ) function is registered in Greenplum 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 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 format functions with the same name in multiple databases and schemas. Because all format object files are stored in the ***full-path-to-pkglibdir*/SAS** directory, the publishing macro uses the database, schema, and model name as the object filename to avoid potential naming conflicts.

Running the %INDGP_PUBLISH_FORMATS Macro

To run the %INDGP_PUBLISH_FORMATS macro, follow these steps:

1. Start SAS and submit one of the following commands in the Program Editor or the Enhanced Editor:

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

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

   For more information, see the "INDCONN Macro Variable" on page 233.

2. Run the %INDGP_PUBLISH_FORMATS macro.

   For more information, see "%INDGP_PUBLISH_FORMATS Macro Syntax" on page 234.

   Messages are written to the SAS log that indicate whether the SAS_PUT( ) function and format functions were successfully created.

INDCONN Macro Variable

The INDCONN macro variable is used as credentials to connect to Greenplum. You must specify the user, password, and either a DSN name or a server and database name. The schema name is optional. You must assign the INDCONN macro variable before the %INDGD_PUBLISH_FORMATS macro is invoked.

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

USER=*username* PASSWORD=*password* DSN=*dsnname* <SCHEMA=*schemaname*> <PORT=*port-number*>

USER=*username* PASSWORD=*password* SERVER=*servername* DATABASE=*databasename* <SCHEMA=*schemaname*> <PORT=*port-number*>

Arguments

USER=*username*
specifies the Greenplum 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 Greenplum 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 Greenplum ODBC data source to which you want to connect.

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.

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.

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.

PORT=*port-number*
specifies the psql port number.

Default: 5432

Requirement The server-side installer uses psql, and psql default port is 5432. If you want to use another port, you must have the UNIX or database administrator change the psql port.

TIP The INDCONN macro variable is not passed as an argument to the %INDGP_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.

%INDGP_PUBLISH_FORMATS Macro Syntax

%INDGP_PUBLISH_FORMATS

(<DATABASE=*database-name*>
<, FMTCAT=*format-catalog-filename*>
<, FMTTABLE=*format-table-name*>
<, ACTION=CREATE | REPLACE | DROP>
<, OUTDIR=*diagnostic-output-directory*>
);

Arguments

DATABASE=*database-name*

specifies the name of a Greenplum database to which the SAS_PUT( ) function and the format functions 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 "Running the %INDGP_PUBLISH_FORMATS Macro" on page 232.

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 are made available in Greenplum.

Defaults 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 Greenplum.

Interactions 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 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*.

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

Table 19.1 *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 the SAS_PUT( ) function.

REPLACE
overwrites the current SAS_PUT( ) function, if a SAS_PUT( ) function is already registered.

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

Default CREATE

Tip If the SAS_PUT( ) 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.

See "Special Characters in Directory Names" on page 200

Format Publishing Macro Example

```
%let indconn = user=user1 password=xxxx dsn=dsnx34 schema=block;
%indgp_publish_formats(fmtcat=work.formats);
```

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

```
CREATE OR REPLACE FUNCTION dbitest.homeeq_5_em_classification
(
float8,
float8,
float8,
float8,
float8,
varchar(32),
float8,
float8,
varchar(32),
float8,
float8
)
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 it is installed, you can use SQL to call the SAS_PUT( ) function in Greenplum. For more information, see “Using the SAS_PUT( ) Function in Greenplum” on page 236.

Using the SAS_PUT( ) Function in Greenplum

Implicit Use of the SAS_PUT( ) Function

Mapping PUT Function Calls to SAS_PUT( )

After you install the formats that SAS supplies in libraries inside the Greenplum 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 (the default) and you submit your program from a SAS session, the SAS SQL processor maps PUT function calls to SAS_PUT( ) function references that Greenplum understands.

This example illustrates how the PUT function is mapped to the SAS_PUT( ) function using implicit pass-through. The SELECT DISTINCT clause executes inside Greenplum, and the processing is distributed across all available data nodes. Greenplum formats the **id** values with the ANIMAL 20.0 format and processes the SELECT DISTINCT clause using the formatted values.

```
/* implicit pass-thru query */
options sqlgeneration=dbms sqlreduceput=none;
```

```
options sastrace=',,,d' sastraceloc=saslog
        sql_ip_trace=(note,source) msglevel=i;

proc sql noerrorstop reduceput=none details="reduce_put_bench$";
create table fmt_ipout as
   select distinct id, put(a,ANIMAL.) len=50 as fmtresult
     from dblib.sample ;
quit;
options sastrace=',,,,'
        sql_ip_trace=none msglevel=n;
```

This is a partial listing of the lines that are written to the SAS log.

```
/*
GREENPL_1: Prepared:
SELECT * FROM SAMPLE FOR READ ONLY

NOTE: XOG: Put Ping Query
NOTE: SELECT SAS_PUT('ANIMAL', '$IS-INTRINSIC') AS X, SAS_PUT('ANIMAL',
      '$FMT-META') AS Y FROM (SELECT COUNT(*) AS C FROM  SAMPLE WHERE 0=1)

GREENPL_2: Prepared:
select distinct TXT_1."id", cast(SAS_PUT(TXT_1."a", 'ANIMAL20.0') as char(20))
   as fmtresult from SAMPLE TXT_1

SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: passed down query:
select distinct TXT_1."id", cast(SAS_PUT(TXT_1."a", 'ANIMAL20.0') as char(20))
   as fmtresult from SAMPLE TXT_1
SQL_IP_TRACE: The SELECT statement was passed to the DBMS.

GREENPL_3: Executed:
Prepared statement GREENPL_2

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

Considerations with Implicit Use of SAS_PUT( )

Be aware of these items:

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

  ```
  select distinct TXT_1."id",
     cast(SAS_PUT(TXT_1."a", 'ANIMAL20.0') as char(20)) as fmtresult
     from SAMPLE TXT_1
  ```

  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 Greenplum VARCHAR(*n*) is defined to be a null-terminated string.

Explicit Use of the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in an SQL Query

If you use explicit pass-through (direct connection to Greenplum), 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 236 and explicitly uses the SAS_PUT( ) function call.

```
options sastrace=',,,d' sastraceloc=saslog
        sql_ip_trace=(note,source) msglevel=i;

proc sql noerrorstop;
connect to greenplm (&exconn) ;
create table fmt_epout as
select * from connection to greenplm (
select id, sas_put(a,'ANIMAL' ) as FMTRESULT
from sample
);
quit;
options sastrace=',,,,'
        sql_ip_trace=none msglevel=n;
```

This is a partial listing of the lines that are written to the SAS log.

```
/*
GREENPL_4: Prepared:
select id, sas_put(a,'ANIMAL' ) as FMTRESULT from sample

GREENPL_5: Executed:
Prepared statement GREENPL_4
*/
```

Considerations with Explicit Use of SAS_PUT( )

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("id", 'animal20.0') as char(20)) as "id",
   from sample;
```

Greenplum Permissions

You must have Greenplum superuser permissions to execute the %INDGP_PUBLISH_FORMATS macro that publishes the SAS_PUT( ) function and the format functions. Greenplum requires superuser permissions to create C functions in the database.

Without these permissions, the publishing of the SAS_PUT( ) function and user-defined 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 20
Deploying and Using SAS Formats in Netezza

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 242. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in the Netezza Data Warehouse ” on page 247.

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 Latin 1:

- Picture formats are not supported. The picture format supports only Latin 1 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 *value-range-set(s)* 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 and submit this command in the Program or Enhanced Editor:

   ```
   %let indconn = server=myserver user=myuserid password=XXXX
      database=mydb <serveruserid=myserveruserid>;
   ```

   For more information, see the “INDCONN Macro Variable” on page 243.

2. Run the %INDNZ_PUBLISH_FORMATS macro.

   For more information, see “%INDNZ_PUBLISH_FORMATS Macro Syntax” on page 243.

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

INDCONN Macro Variable

The INDCONN macro variable is used as credentials to connect to Netezza. You must specify the 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_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 242.

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 241

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 format table contains the columns shown in the following table.

Table 20.1 *Format Table Columns*

Column Name	Description
FMTNAME	specifies the name of the format.

Column Name	Description
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.

Tip There are limited resources available in Netezza when you run in fenced mode. For example, there is a limit to the number of columns available.

See “Modes of Operation” on page 246

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 200

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 format that is published is isolated in a separate process in the Netezza database when it is invoked. An error does not cause the database to stop. It is recommended that you publish the format in fenced mode during acceptance tests.

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

```
%let indconn = server=netezbase user=user1 password=xxxx
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 247.

Using the SAS_PUT( ) Function in the Netezza Data Warehouse

Implicit Use of the SAS_PUT( ) Function

Mapping PUT Function Calls to SAS_PUT( )

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. 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.

```
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=xxxx 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, 2013

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

           quit;
```

Considerations with Implicit Use of SAS_PUT( )

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT. This ensures that the SQL processor maps your PUT functions to the SAS_PUT( ) function and that 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.

Explicit Use of the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in an SQL Query

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 247 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, 2013

                          PRICE_C
                          _______
                           $27.98
                           $10.00
                           $12.00
```

```
                              $13.59
                              $48.99
                              $54.00
                              $13.98
                               $8.00
                              $14.00

disconnect from netezza;
quit;
```

Considerations with Explicit Use of SAS_PUT( )

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_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 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 21

Deploying and Using SAS Formats in Teradata

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 252. For more information about the SAS_PUT( ) function, see "Using the SAS_PUT( ) Function in the Teradata EDW " on page 258.

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 use a character set encoding other than Latin 1, picture formats are not supported. The picture format supports only Latin 1 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. For more information about the SAS formats library, see "Deployed Components for Teradata" on page 9.

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 *value-range-set(s)* into embedded data in Teradata.

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 and submit this command in the Program or Enhanced Editor:

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

   For more information, see the "INDCONN Macro Variable" on page 253.

2. Run the %INDTD_PUBLISH_FORMATS macro.

   For more information, see "%INDTD_PUBLISH_FORMATS Macro Syntax" on page 253.

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

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.

Default Your current database

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.

Default The database specified in the INDCONN macro variable or your current database

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 252.
Tip	The format definitions and the SAS_PUT( ) function do not need to 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 280.

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 21.1 *Teradata 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 255

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 200

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

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).

```
%let indconn server="terabase" user="user1" password="open1" database="mydb";
%indtd_publish_formats(fmtcat= fmtlib.fmtcat);
```

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 258.

Data Types and the SAS_PUT( ) Function

The SAS_PUT( ) function supports direct use of the Teradata data types shown in the following table. 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 21.2 *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)
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, trailing blanks are trimmed in columns with a data type of CHARACTER when converting to a VARCHAR data type.

The SAS_PUT( ) function does not support direct use of the Teradata data types shown in the following table. 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 21.3 *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

Mapping PUT Function Calls to SAS_PUT( )

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.

Note: If you specify SQLMAPPUTTO=*database*.SAS_PUT, *database* must be the same as the database where the SAS_PUT function is mapped.

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

```
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, July 11, 2013

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

         quit;
```

Considerations with Implicit Use of SAS_PUT( )

Be aware of these items:

- The SQLMAPPUTTO= system option must be set to SAS_PUT. This ensures that the SQL processor maps your PUT functions to the SAS_PUT( ) function and that 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.

Explicit Use of the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in an SQL Query

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 258 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, July 25, 2013

                           PRICE_C
                           _______
                             $8.00
                            $10.00
```

```
                   $12.00
                   $13.59
                   $13.99
                   $14.00
                   $27.98
                   $48.99
                   $54.00

disconnect from teradata;
quit;
```

Considerations with Explicit Use of SAS_PUT( )

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, the $HEXw. format should not be associated with variables that are processed by the SAS_PUT( ) function implicitly. For more information, see "Data Types and the SAS_PUT( ) Function" on page 256.

  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". However, 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 the functions so that access rights can be customized as needed. In addition, you must have the following permissions to publish the 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 6

In-Database Procedures

Chapter 22
Running SAS Procedures inside the Database

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.

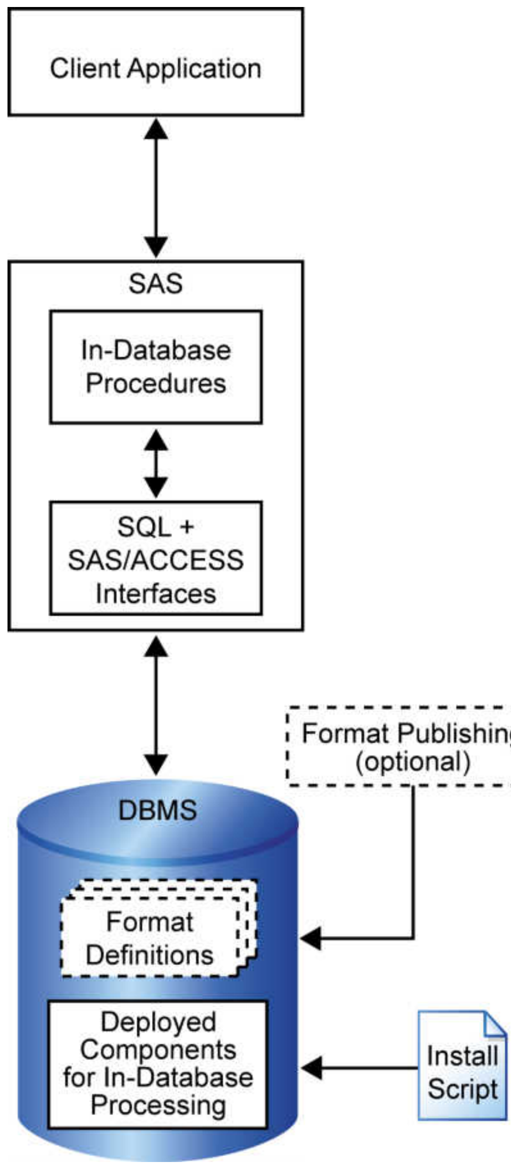
Using the new in-database technology, the procedures that are enabled for processing inside the data source generate more sophisticated queries. These queries allow the aggregations and analytics to be run inside the data source. Some of the in-database procedures generate SQL procedure syntax and use 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 data source, 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 data source. One example of a special SAS function is the SAS_PUT( ) function that enables you to execute PUT function calls inside the data source. 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 data source, and less data movement can occur. This could result in significant performance improvements.

This diagram illustrates the in-database procedure process.

Figure 22.1 *Process Flow Diagram*



Running In-Database Procedures

To run in-database procedures, the SQLGENERATION system option or the SQLGENERATION LIBNAME option must be set to DBMS or DBMS='*database-name*'.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the data source. By default, the

SQLGENERATION system option is set to NONE DBMS='ASTER DB2 GREENPLUM HADOOP NETEZZA ORACLE TERADATA SAPHANA'.

Conventional SAS processing is also used when specific procedure statements and options do not support in-database processing. For complete information, see "SQLGENERATION= System Option" on page 277 or the SQLGENERATION LIBNAME option in *SAS/ACCESS for Relational Databases: Reference*.

Procedures in Aster, DB2, Greenplum, Hadoop, Netezza, Oracle, and SAP HANA

The following Base SAS procedures have been enhanced for in-database processing inside Aster, DB2, Greenplum, Hadoop, Netezza, Oracle, and SAP HANA.

- FREQ
- RANK

  Note: PROC RANK in-database processing is not supported by Hadoop.
- REPORT
- SORT

  Note: Only the NODUPKEY option of PROC SORT is supported with in-database processing.

  Note: PROC SORT in-database processing is not supported by Hadoop.
- SUMMARY/MEANS
- TABULATE

For more information about running a specific procedure inside the data source, see the documentation for that procedure.

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.

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 use in-database procedures with user-defined formats that were published in the data source, 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 data source. 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 data source.

If this occurs, you can redefine the local format to match the published version and rerun the procedure inside the data source.

For more information about publishing user-defined formats, see the section on deploying and using formats for your data source in Part 3, “Format Publishing and the SAS_PUT( ) Function.”

Note: Format publishing of user-defined formats is not available for Hadoop, Oracle, and SAP HANA.

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.
- If you specify the ORDER=DATA option for input data, the procedure might produce different results for separate runs of the same analysis.
- The order of rows written to a data source 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 data source 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.

When SAS/ACCESS creates a data source table, SAS/ACCESS by default 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:
  - CONNECTION
  - CONNECTION_GROUP
  - DBCREATE_TABLE_OPTS
  - DBMSTEMP=YES
  - DBCONINIT
  - DBCONTERM
  - DBGEN_NAME=SAS

  - HDFS_METADIR
  - MODE=TERADATA
- LIBNAME concatenation prevents in-database processing.

Data Set-Related Options

These data set options and settings prevent in-database processing:

- DBCONDITION
- DBFORCE
- DBLINK (Oracle only)
- DBNULL
- DBTYPE
- NULLCHAR
- NULLCHARVAL
- OBS= and FIRSTOBS= on DATA= data set
- OUT= data set on DBMS and DATA= data set not on DBMS

  For example, if *data=work.foo* and *out=tera.fooout* where WORK is the Base SAS engine, in-database processing does not occur.
- RENAME= on a data set
- SCHEMA

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.

Additional Limitations That Can Prevent In-Database Processing

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 data source resolves function references. the data source searches in this order:
  1. fully qualified object name
  2. current data source
  3. SYSLIB

If you need to reference functions that are published in a nonsystem, nondefault data source, you must use one of these methods:

- Use explicit SQL.
- Use the DATABASE= LIBNAME option.
- Map the fully qualified name (*schema*.sas_put) in the external mapping.

- Oracle Version 10g supports only 4000 characters per input data item. If you are transcoding input data that has special characters, be aware that these characters might need more than one byte per character.

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.

- a note that says that the TYPE= attribute is not stored in DBMS tables. You see this note if you try to create a special SAS data set as a DBMS table for PROC MEANS or PROC SUMMARY.
- a note that says if the format was or was not found in the data source. You see this note if you use a format that SAS supplies or a user-defined format.

Part 7

System Options Reference

Chapter 23

System Options That Affect In-Database Processing

Dictionary

DS2ACCEL= System Option

Specifies whether DS2 code is enabled for parallel processing in supported environments using the SAS In-Database Code Accelerator.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: NONE

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in Chapter 1 of *SAS System Options: Reference*.

Syntax

DS2ACCEL=ANY | NONE

Arguments

ANY
enables DS2 code to execute in supported parallel environments.

NONE
disables DS2 code from executing in supported parallel environments.

Details

The SAS In-Database Code Accelerator enables you to publish a DS2 thread program to the database and execute the thread program in parallel inside the database. If you are using the SAS In-Database Code Accelerator for Teradata or Hadoop, the DS2 data program is also published and executed inside the database.

The DS2ACCEL= system option controls whether DS2 code is executed inside the database.

You can override the DS2ACCEL= system option by specifying the DS2ACCEL= option in the PROC DS2 statement.

See Also

- "Using the DS2ACCEL Option to Control In-Database Processing" in Chapter 21 of *SAS DS2 Language Reference*

Procedures:

- Chapter 19, "DS2 Procedure" in *Base SAS Procedures Guide*

DSACCEL= System Option

Specifies whether the DATA step is enabled for parallel processing in supported environments.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: The shipped default is NONE.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in Chapter 1 of *SAS System Options: Reference*.

Syntax

DSACCEL=ANY | NONE

Syntax Description

ANY
enables the DATA step to execute in supported parallel environments.

NONE
disables the DATA step from executing in supported parallel environments.

Details

SAS enables the DATA step to run, with limitations, in these environments:

- SAS LASR Analytic Server
- Inside Hadoop using SAS/ACCESS and SAS Embedded Process

You can use the MSGLEVEL= system option to control the message detail that appears in the SAS log for Hadoop MapReduce jobs:

- Specify MSGLEVEL=N to see only notes, warnings, and error messages.
- Specify MSGLEVEL=I to view additional Hadoop MapReduce messages.

See Also

- *SAS LASR Analytic Server: Reference Guide*
- *SAS In-Database Products: User's Guide*

System options:

- "MSGLEVEL= System Option" in *SAS System Options: Reference*

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

Categories: Data Access

System Administration: Performance

Default: (NONE DBMS='TERADATA DB2 ORACLE NETEZZA ASTER GREENPLM HADOOP SAPHANA')

Restrictions: Parentheses are required when this option value contains multiple keywords.

The maximum length of the option value is 4096 characters.

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.

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.

If you are using the Metadata LIBNAME Engine, the only valid SQLGENERATION= modifiers are NONE and DBMS. The engine ignores the DBMS=, EXCLUDEDB=, and EXCLUDEPROC= modifiers.

Requirement: You must specify NONE or DBMS as the primary state.

Interactions: Use this option with such procedures as PROC FREQ to indicate that SQL is generated for in-database processing of DBMS tables through supported SAS/ACCESS engines.

You can specify different SQLGENERATION= values for the DATA= and OUT= data sets by using different LIBNAME statements for each of these data sets.

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, Netezza, Oracle, SAP Hana, Teradata

Tip: After you set a required value (primary state), you can specify optional values (modifiers).

See: SQLGENERATION= LIBNAME option (includes examples), “Running In-Database Procedures” in *SAS In-Database Products: User’s Guide*

Syntax

SQLGENERATION=<(>NONE | DBMS
<DBMS='*engine1 engine2…enginen*'>
<EXCLUDEDB=*engine* | '*engine1…enginen*'>
<EXCLUDEPROC="*engine*='*proc1…procn*' *enginen*='*proc1…procn*' "><)>

SQLGENERATION=" "

Required Arguments

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.

" "
resets the value to the default that was shipped.

Optional Arguments

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*' "
prevents one or more specified SAS procedures from generating SQL for in-database processing for one or more specified SAS/ACCESS engines.

Details

Here is how SAS/ACCESS handles precedence between the LIBNAME and system option.

***Table 23.1** Precedence of Values for SQLGENERATION= LIBNAME and System Options*

LIBNAME Option	PROC EXCLUDE on System Option?	Engine Specified on System Option	Resulting Value	From (option)
not set	yes	NONE	NONE	system
		DBMS	EXCLUDEPROC	
NONE		NONE	NONE	LIBNAME
		DBMS		
DBMS		NONE	EXCLUDEPROC	system
		DBMS		
not set	no	NONE	NONE	
		DBMS	DBMS	
NONE		NONE	NONE	LIBNAME
		DBMS		
DBMS		NONE	DBMS	
		DBMS		

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. SAS procedures that are run on other databases do not generate SQL for in-database processing.

```
options sqlgeneration='';
options SQLGENERATION=(NONE DBMS='Teradata');
proc options option=sqlgeneration;
run;
```

For this example, SAS procedures generate SQL for Teradata and Oracle in-database processing. However, no SQL is generated for PROC1 and PROC2 in Oracle.

```
options sqlgeneration='';
options SQLGENERATION = (NONE DBMS='Teradata Oracle'
   EXCLUDEPROC="oracle='proc1 proc2'");
```

```
proc options option=sqlgeneration;
run;
```

SQLMAPPUTTO= System Option

Specifies whether the PUT function is mapped to the SAS_PUT() function for a database, possible also where the SAS_PUT() function is mapped.

Valid in: configuration file, SAS invocation, OPTIONS statement

Category: Files: SAS Files

Default: SAS_PUT

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, Netezza, Teradata

See: SQL_FUNCTIONS= LIBNAME option, *SAS In-Database Products: User's Guide*

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.

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 *database*.SAS_PUT argument to specify the database where the format definitions and the SAS_PUT() function have been published.

TIP The database name can be a multilevel name and it can include blanks.

Requirement If you specify a database name, you must enclose the entire argument in parentheses.

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 and formats that SAS supplies that you create using PROC FORMAT. The SAS_PUT() function supports the use of SAS formats. You can use it 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 *SAS SQL Procedure User's Guide*.

SQLREDUCEPUT= System Option

For the SQL procedure, specifies the engine type to use to optimize a PUT function in a query. The PUT function is replaced 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 “Restricted Options” in Chapter 1 of *SAS System Options: Reference*.

Syntax

SQLREDUCEPUT= ALL | NONE | DBMS | BASE

Syntax Description

ALL
specifies to consider the optimization of all PUT functions, regardless of the engine that is used by the query to access the data.

NONE
specifies to not optimize any PUT function.

DBMS
specifies to consider the optimization of all PUT functions in a query performed by a SAS/ACCESS engine. This is the default.

Requirement: The first argument to the PUT function must be a variable that is obtained by a table. The table must be accessed using a SAS/ACCESS engine.

BASE
specifies to consider the optimization of all PUT functions in a query performed by a SAS/ACCESS engine or a Base SAS engine.

Details

If you specify the SQLREDUCEPUT= system option, SAS optimizes the PUT function 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 are optimized if the SQLREDUCEPUT= option is 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 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 Chapter 5 of *SAS SQL Procedure User's Guide*

Procedure Statement Options:

- REDUCEPUT= option

System Options:

- “SQLCONSTDATETIME System Option” in *SAS SQL Procedure User's Guide*
- “SQLREDUCEPUTOBS= System Option” in *SAS SQL Procedure User's Guide*

Part 8

Appendix

Appendix 1
Scoring File Examples

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>
```

Recommended Reading

Here is the recommended reading list for this title:

- *Base SAS Procedures 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 DS2 Language Reference*
- *SAS In-Database Products: Administrator's Guide*
- *SAS Model Manager: User's Guide*
- *SAS/STAT User's Guide*
- *SAS Scalable Performance Data Server: User's Guide*

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

SAS Books
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

Index