# Contents

What’s New in SAS 9.4 In-Database Products ........................................... vii

## PART 1 Introduction 1

**Chapter 1 • SAS In-Database Processing** ................................................... 3
- Introduction to SAS In-Database Processing ........................................ 3
- Deployed Components for In-Database Processing ................................... 5
- Where to Go from Here ........................................................................... 10

## PART 2 SAS Scoring Accelerator 11

**Chapter 2 • Introduction to the SAS Scoring Accelerator** ............................. 13
- SAS Scoring Accelerator for SAS/ACCESS Databases .............................. 13
- SAS Scoring Accelerator for SPD Server ............................................... 15
- Scoring with User-Defined Functions and the SAS Embedded Process ...... 17
- Considerations When Creating or Modifying DATA Step Score Code .......... 17
- Special Characters in Directory Names .................................................. 20

**Chapter 3 • Exporting the Scoring Model Files from SAS Enterprise Miner** ..... 23
- Overview of the Score Code Export Node .............................................. 23
- Comparing the Score Code Export Node with Registering Models on the SAS Metadata Server .............................................. 24
- Using the Score Code Export Node in a Process Flow Diagram ............... 24
- Output Created by the Score Code Export Node ..................................... 25

**Chapter 4 • Analytic Store Scoring** ............................................................... 29
- Introduction to Analytic Store Scoring .................................................... 29
- How to Generate Analytic Store Scoring Files for the SAS Scoring Accelerator .................................................... 30

**Chapter 5 • SAS Scoring Accelerator for Aster** ........................................... 31
- Overview of Running Scoring Models in Aster ........................................ 31
- Running the %INDAC_PUBLISH_MODEL Macro ..................................... 32
- Scoring Files and Functions inside the Aster Database ............................ 37

**Chapter 6 • SAS Scoring Accelerator for DB2 under UNIX** ......................... 41
- Overview of Running Scoring Models in DB2 ........................................ 41
- Using Scoring Functions to Run Scoring Models .................................... 42
- Using the SAS Embedded Process to Run Scoring Models ....................... 47
- Running the %INDB2_PUBLISH_MODEL Macro ..................................... 52
- DB2 Permissions .................................................................................... 59

**Chapter 7 • SAS Scoring Accelerator for Greenplum** .................................. 61
- Overview of Running Scoring Models in Greenplum .............................. 61
- Using Scoring Functions to Run Scoring Models .................................... 62
- Using the SAS Embedded Process to Run Scoring Models ....................... 67
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>SAS Scoring Accelerator for Hadoop</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models in Hadoop</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Running Scoring Models in Hadoop</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>INDCONN Macro Variable</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>%INDH_PUBLISH_MODEL Macro Syntax</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>%INDH_RUN_MODEL Syntax</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Creating a Metadata File for the Input Data File</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Scoring Output</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>Reading and Writing HCatalog File Types</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>Using the MapReduce Job Logs to View DS2 Error Messages</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Hadoop Permissions</td>
<td>95</td>
</tr>
<tr>
<td>9</td>
<td>SAS Scoring Accelerator for Netezza</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models in Netezza</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Using Scoring Functions to Run Scoring Models</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>Using the SAS Embedded Process to Run Scoring Models</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>INDCONN Macro Variable</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>Running the %INDNZ_PUBLISH_MODEL Macro</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>Netezza Permissions</td>
<td>116</td>
</tr>
<tr>
<td>10</td>
<td>SAS Scoring Accelerator for Oracle</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>Oracle Permissions</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>How to Run a Scoring Model in Oracle</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>Creating a Model Table</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>Running the %INDOR_PUBLISH_MODEL Macro</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>Oracle Scoring Files</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>SASEPFUNC Table Function</td>
<td>125</td>
</tr>
<tr>
<td>11</td>
<td>SAS Scoring Accelerator for SAP HANA</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models in SAP HANA</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>How to Run a Scoring Model in SAP HANA</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>INDCONN Macro Variable</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Creating the Model Table</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>Running the %INDHN_PUBLISH_MODEL Macro</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Running the %INDHN_RUN_MODEL Macro</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>Scoring Output</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>SAP HANA Permissions</td>
<td>143</td>
</tr>
<tr>
<td>12</td>
<td>SAS Scoring Accelerator for SPD Server</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models in SPD Server</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Running Scoring Models in SPD Server</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>INDCONN Macro Variable</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>INDDATA Macro Variable</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>%INDSP_PUBLISH_MODEL Macro Syntax</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>%INDSP_RUN_MODEL Macro Syntax</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Scoring Output</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>SPD Server Permissions</td>
<td>154</td>
</tr>
<tr>
<td>13</td>
<td>SAS Scoring Accelerator for Teradata</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>Overview of Running Scoring Models in Teradata</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>Using Scoring Functions to Run Scoring Models</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>Using the SAS Embedded Process to Run Scoring Models</td>
<td>160</td>
</tr>
</tbody>
</table>
### Contents

- Running the %INDTD_PUBLISH_MODEL Macro .............................................. 171
- Teradata Permissions ................................................................................. 178

**Chapter 14 • SAS Scoring Accelerator and SAS Model Manager** .................. 179
- Using the SAS Scoring Accelerator with SAS Model Manager ................... 179

**PART 3 SAS In-Database Code Accelerator** .............................................. 181

- **Chapter 15 • Using the SAS In-Database Code Accelerator** .................... 183
  - Overview of the SAS In-Database Code Accelerator ............................... 183
  - Requirements for Using the SAS In-Database Code Accelerator .......... 184
  - SAS In-Database Code Accelerator for Greenplum ............................... 184
  - SAS In-Database Code Accelerator for Hadoop ................................. 185
  - SAS In-Database Code Accelerator for Teradata ................................. 188
  - Using the DS2ACCEL Option to Control In-Database Processing .......... 188
  - BY-Group Processing When Running Thread Programs inside the Database 189
  - Considerations and Limitations ............................................................... 189
  - SAS In-Database Code Accelerator Examples ...................................... 192

**PART 4 In-Database DATA Step Processing** .............................................. 201

- **Chapter 16 • DATA Step Processing in Hadoop** ........................................ 203
  - DATA Step Processing in Hadoop ......................................................... 203
  - Requirements for DATA Step Processing ............................................. 204
  - Restrictions in DATA Step Processing ................................................ 204
  - Example: DATA Step Program for Hadoop ......................................... 206

**PART 5 Format Publishing and the SAS_PUT( ) Function** .......................... 207

- **Chapter 17 • Deploying and Using SAS Formats inside the Database** ......... 209
  - Using SAS Formats and the SAS_PUT( ) Function ................................ 209
  - How It Works .......................................................................................... 210
  - Format Publishing with User-Defined Functions and the SAS Embedded Process . . 212
  - Special Characters in Directory Names ................................................ 212
  - Considerations and Limitations with User-Defined Formats ................. 214
  - Tips for Using the Format Publishing Macros ....................................... 214
  - Tips for Using the SAS_PUT( ) Function ............................................. 215
  - Determining Format Publish Dates ...................................................... 215

- **Chapter 18 • Deploying and Using SAS Formats in Aster** ....................... 217
  - User-Defined Formats in the Aster Database ......................................... 217
  - Publishing SAS Formats in Aster ......................................................... 218
  - Aster Format Files ................................................................................. 222
  - Using the SAS_PUT( ) Function in the Aster Database ....................... 225
  - Aster Permissions .................................................................................... 229

- **Chapter 19 • Deploying and Using SAS Formats in DB2 under UNIX** ....... 231
  - User-Defined Formats in the DB2 Database ........................................ 231
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 July 2015 release of SAS 9.4, the following changes and enhancements were made:
  - The SAS In-Database Code Accelerator now supports a SET statement with embedded SQL, a SET statement that specifies multiple input tables, and the MERGE statement.
  - In-database processing of PROC TRANSPOSE is preproduction for Teradata and Hadoop.
  - The SAS Scoring Accelerator for Hadoop, SAP HANA, and Teradata supports model scoring using analytic stores.
  - The run and publish model macros for the SAS Scoring Accelerator for Hadoop now support the SAS_HADOOP_CONFIG_PATH environment variable. This eliminates the need for a merged configuration file.
  - The installation and configuration of the SAS Embedded Process for Teradata has changed. The in-database deployment package is delivered to the client from the SAS Install Depot. The new process has a smaller client footprint and is a faster install process.
  - The SAS Embedded Process for Hadoop now supports IBM BigInsights, MapR, and Pivotal HD Hadoop distributions.
  - The installation and configuration of the SAS Embedded Process for Hadoop has changed significantly. For Cloudera and Hortonworks, Cloudera Manager and Ambari are used to install the SAS Embedded Process and the SAS Hadoop MapReduce JAR files. For IBM BigInsights, MapR, and Pivotal HD, the in-database deployment package is delivered to the client from the SAS Install Depot. In addition, the SAS Embedded Process and the SAS Hadoop MapReduce JAR files are installed with one script instead of in two separate scripts. The new process has a smaller client footprint and is a faster install process.
- In the February 2015 release of SAS 9.4, the following changes and enhancements 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.
HCatalog file formats are not supported on Pivotal HD v2.x or IBM BigInsights v3.x and later.

- 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 enhancements 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 enhancements 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 enhancements 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 enhancements were made:
  - The SAS Scoring Accelerator for SPD Server is available.

- In the June 2013 release of SAS 9.4, the following changes and enhancements 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**

**July 2015 Release of SAS 9.4: Changes and Enhancements**

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

- The SET statement can contain embedded SQL.
- You can specify multiple input tables in the SET statement.
- The SAS In-Database Code Accelerator for Hadoop supports reading and writing of HDFS-SPD Engine file formats.
- When a Hadoop data or thread program fails, write a message to the SAS log that contains a link to the MapReduce job log where you can find the error messages.

**February 2015 Release of SAS 9.4: Changes and Enhancements**

In the February 2015 release of SAS 9.4, the following changes and enhancements 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 enhancements 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

July 2015 Release of SAS 9.4: Changes and Enhancements

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

- The run and publish model macros for the SAS Scoring Accelerator for Hadoop now support the SAS_HADOOP_CONFIG_PATH environment variable. This eliminates the need for a merged configuration file. The INDCONN macro variable has a new argument, and two other arguments are no longer supported.
- The SAS In-Database Scoring Accelerator for Hadoop now uses HCatalog to process complex, non-delimited files. This enables the SAS In-Database Scoring Accelerator for Hadoop to support all HCatalog file formats, including Avro, ORC, RCFile, and Parquet. HCatalog file formats are not supported on Pivotal HD v2.x or IBM BigInsights v3.x and later.
- Scoring models using analytic stores is supported.
- In-database processing of PROC TRANSPOSE is preproduction.
- The SAS Embedded Process for Hadoop now supports IBM BigInsights, MapR, and Pivotal HD Hadoop distributions.
- The installation and configuration of the SAS Embedded Process for Hadoop has changed.
  - For Cloudera and Hortonworks, Cloudera Manager and Ambari are used to install the SAS Embedded Process and the SAS Hadoop MapReduce JAR files.
  - For IBM BigInsights, MapR, and Pivotal HD, the in-database deployment package is delivered to the client from the SAS Install Depot.
  - The SAS Embedded Process and the SAS Hadoop MapReduce JAR files are installed with one script instead of in two separate scripts. The new process has a smaller client footprint and is a faster install process.
  - The sasep-servers.sh file has changed names to the sasep-admin.sh file. Some of the scripts arguments are no longer needed and have been deleted. Other arguments have been added.

August 2014 Release of SAS 9.4: Changes and Enhancements

In the August 2014 release of SAS 9.4, the following changes and enhancements were made:

- 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, the following changes and enhancements were made:

- 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.
Netezza Changes

July 2015 Release of SAS 9.4: Changes and Enhancements

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

• If you have Netezza v7.0.3 or later, you can now publish formats and macros to different schemas.
• The SAS Embedded Process for Netezza has a new cartridge file that creates the NZRC database.

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.

SAP HANA Changes

July 2015 Release of SAS 9.4: Changes and Enhancements

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

• If you are using SAP HANA SPS9, the SAS Embedded Process for SAP HANA must be manually started. For previous versions, the SAS Embedded Process was automatically started by the SASAFL plug-in. In addition, a different procedure must be used to deploy the SASAFL plug-in.
• Scoring models using analytic stores is supported.
• Views can be used as input to an SAP HANA scoring model.

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.
SPD Server Changes

July 2013 Release of SAS 9.4: Changes and Enhancements

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

Teradata Changes

July 2015 Release of SAS 9.4: Changes and Enhancements

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

- If you have Teradata v14.10 or later, any object such as column names can be up to 128 characters.
- In-database processing of PROC TRANSPOSE is preproduction.
- Scoring models using analytic stores is supported.
- The installation and configuration of the SAS Embedded Process for Teradata has changed. The in-database deployment package is delivered to the client from the SAS Install Depot. The new process has a smaller client footprint and is a faster install 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.
In-Database Procedures

July 2015 Release of SAS 9.4: Changes and Enhancements

In the July 2015 release of SAS 9.4, the following changes and enhancements were made:

• In-database processing of PROC TRANSPOSE is preproduction for Teradata and Hadoop.
• In-database processing of Base Procedures is now supported for Impala and HAWQ.

SAS 9.4: Changes and Enhancements

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

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.

%INDACP F %INDB2PF %INDGPMM %INDNZPM
%INDACP M %INDB2PM %INDNZPC %INDORPM
%INDB2PC %INDGPPC %INDNZPF %INDTDPF
%INDB2PD %INDGPFF %INDNZPJ %INDTDPM

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.
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 providers. Using SAS in-database processing, you can run scoring models, some SAS procedures, DS2 thread programs, and formatted SQL queries inside the data source.

Note: For the sake of brevity, in this document, the term database is used to denote both relational database management systems and file systems.

The following table lists the SAS products needed to use these features.
### In-Database Feature

<table>
<thead>
<tr>
<th>In-Database Feature</th>
<th>Software Required</th>
<th>Supported Data Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>format publishing and the SAS_PUT( ) function</td>
<td>Base SAS</td>
<td>Aster</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to the data source</td>
<td>DB2 under UNIX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Netezza</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Teradata</td>
</tr>
<tr>
<td>scoring models</td>
<td>Base SAS</td>
<td>Aster</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to the data source</td>
<td>DB2 under UNIX</td>
</tr>
<tr>
<td></td>
<td>SAS Scoring Accelerator</td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td>SAS Enterprise Miner</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>SAS Factory Miner (analytic store scoring)*</td>
<td>Netezza</td>
</tr>
<tr>
<td></td>
<td>SAS Scalable Performance Data Server (optional)</td>
<td>Oracle</td>
</tr>
<tr>
<td></td>
<td>SAS Model Manager (optional)</td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPD Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Teradata</td>
</tr>
<tr>
<td>Base SAS procedures:</td>
<td>Base SAS</td>
<td>Aster</td>
</tr>
<tr>
<td>FREQ</td>
<td>SAS/ACCESS Interface to the data source</td>
<td>DB2 (UNIX and z/OS)</td>
</tr>
<tr>
<td><strong>RANK</strong></td>
<td></td>
<td>Greenplum</td>
</tr>
<tr>
<td>REPORT</td>
<td></td>
<td>Hadoop</td>
</tr>
<tr>
<td><strong>SORT</strong></td>
<td></td>
<td>Impala</td>
</tr>
<tr>
<td>SUMMARY/MEANS</td>
<td></td>
<td>Oracle</td>
</tr>
<tr>
<td>TABULATE</td>
<td></td>
<td>Netezza</td>
</tr>
<tr>
<td>***TRANSPOSE</td>
<td></td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Teradata</td>
</tr>
<tr>
<td>SAS/STAT procedures:</td>
<td>Base SAS (for CORR)</td>
<td>Teradata</td>
</tr>
<tr>
<td>CORR</td>
<td>SAS/ACCESS Interface to Teradata</td>
<td></td>
</tr>
<tr>
<td>CANCORR</td>
<td>SAS/STAT (for CANCORR, FACTOR,PRINCOMP, REG, SCORE, VARCLUS)</td>
<td></td>
</tr>
<tr>
<td>DMDB</td>
<td>SAS/ETS (for TIMESERIES)</td>
<td></td>
</tr>
<tr>
<td>DMINE</td>
<td>SAS Enterprise Miner (for DMDB, DMINE, DMREG)</td>
<td></td>
</tr>
<tr>
<td>DMREG</td>
<td>SAS Analytics Accelerator</td>
<td></td>
</tr>
</tbody>
</table>
## 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 (tkindsrvr-9.4_M3-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.

<table>
<thead>
<tr>
<th>In-Database Feature</th>
<th>Software Required</th>
<th>Supported Data Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS2 threaded programs</td>
<td>Base SAS</td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to the data source</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>SAS In-Database Code Accelerator</td>
<td>Teradata</td>
</tr>
<tr>
<td>DATA step scoring programs</td>
<td>Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to Hadoop</td>
<td></td>
</tr>
<tr>
<td>data quality operations</td>
<td>Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to Hadoop</td>
<td>Teradata</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to Teradata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS In-Database Code Accelerator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS Data Loader for Hadoop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS Data Quality Accelerator for Teradata</td>
<td></td>
</tr>
<tr>
<td>extract and transform data</td>
<td>Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to Hadoop</td>
<td>Teradata</td>
</tr>
<tr>
<td></td>
<td>SAS/ACCESS Interface to Teradata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS Data Loader for Hadoop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS Data Quality Accelerator for Teradata</td>
<td></td>
</tr>
</tbody>
</table>

* Analytic store scoring is supported only on Hadoop, SAP HANA, and Teradata.

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

*** In-database processing of PROC TRANSPOSE is preproduction only on Hadoop and Teradata. Additional licensing and configuration is required. For more information, see “Running PROC TRANSPOSE inside the Database (Preproduction)” on page 285.
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_M3-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 | Linux64>/` 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_M3-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_M3-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_M3-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, `sepcorehdpd-9.43000-\(n\).sh`). \(n\) is a number that indicates the latest version of the file.

The archive file is included in a ZIP file (en_sasep.zip) that is located in the `YourSASDepot/standalone_installs/SAS_Core_Embedded_Process_Package_for_Hadoop/9_43/Hadoop_on_Linux_x64/` directory.

The SAS Embedded Process and SAS Hadoop MapReduce JAR files 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_M3-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 \texttt{SAS-installation-directory/SASFormatsLibraryforNetezza/3.1/Netezza32bitTwinFin/} directory. The second archive file is located in the \texttt{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 \texttt{\%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 \texttt{\%INDNZ\_PUBLISH\_COMPILEUDF} macro 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_M3-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 \texttt{\%INDNZ\_PUBLISH\_JAZLIB} and \texttt{\%INDNZ\_PUBLISH\_COMPILEUDF} macros are typically run by your system or database administrator.

  For more information, see the \textit{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_M3-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 \texttt{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 \textit{SAS In-Database Products: Administrator's Guide}.  

8 Chapter 1 • SAS In-Database Processing
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_M3-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 \texttt{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 \textit{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 sepcoretera-9.43000-n.x86_64.rpm). \( n \) is a number that indicates the latest version of the file.

The first RPM file is located in the \texttt{SAS-installation-directory/SASFormatsLibraryforTeradata/3.1/TeradataonLinux/} directory. The second RPM file is included in a ZIP file (en_sasex.zip) that is located in the \texttt{YourSASDepot/standalone_installs/SAS_Core_EMBEDDED_PROCESS_PACKAGE_for_Teradata/9_43/Teradata_on_Linux/} 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 \textit{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:

<table>
<thead>
<tr>
<th>In-Database Processing Task</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish user-defined formats and use the SAS_PUT( ) function.</td>
<td>Chapter 17, “Deploying and Using SAS Formats inside the Database,” on page 209</td>
</tr>
<tr>
<td>Run procedures inside the database.</td>
<td>Chapter 23, “Running SAS Procedures inside the Database,” on page 279</td>
</tr>
<tr>
<td>Run DS2 thread programs inside the database.</td>
<td>Chapter 15, “Using the SAS In-Database Code Accelerator,” on page 183</td>
</tr>
<tr>
<td>Run DATA step scoring programs in Hadoop.</td>
<td>Chapter 16, “DATA Step Processing in Hadoop,” on page 203</td>
</tr>
<tr>
<td>Perform data quality operations or extract and transform data</td>
<td><em>SAS Data Loader for Hadoop: User's Guide</em></td>
</tr>
</tbody>
</table>
# Part 2

## SAS Scoring Accelerator

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Introduction to the SAS Scoring Accelerator</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Exporting the Scoring Model Files from SAS Enterprise Miner</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Analytic Store Scoring</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>SAS Scoring Accelerator for Aster</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>SAS Scoring Accelerator for DB2 under UNIX</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>SAS Scoring Accelerator for Greenplum</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>SAS Scoring Accelerator for Hadoop</td>
<td>79</td>
</tr>
<tr>
<td>9</td>
<td>SAS Scoring Accelerator for Netezza</td>
<td>97</td>
</tr>
<tr>
<td>10</td>
<td>SAS Scoring Accelerator for Oracle</td>
<td>117</td>
</tr>
<tr>
<td>11</td>
<td>SAS Scoring Accelerator for SAP HANA</td>
<td>129</td>
</tr>
<tr>
<td>12</td>
<td>SAS Scoring Accelerator for SPD Server</td>
<td>145</td>
</tr>
<tr>
<td>13</td>
<td>SAS Scoring Accelerator for Teradata</td>
<td>155</td>
</tr>
<tr>
<td>14</td>
<td>SAS Scoring Accelerator and SAS Model Manager</td>
<td>179</td>
</tr>
</tbody>
</table>
Chapter 2
Introduction to the SAS Scoring Accelerator

SAS Scoring Accelerator for SAS/ACCESS Databases ........................................ 13
   Overview of the SAS Scoring Accelerator for SAS/ACCESS Databases .............. 13
   How It Works for SAS/ACCESS Databases .................................................... 14

SAS Scoring Accelerator for SPD Server ............................................................ 15
   Overview of the SAS Scoring Accelerator for SPD Server .............................. 15
   How It Works for SAS SPD Server ................................................................... 16

Scoring with User-Defined Functions and the SAS Embedded Process .............. 17
   Considerations When Creating or Modifying DATA Step Score Code ............ 17
      How the SAS Scoring Accelerator Processes the DATA Step Score Code ...... 17
      Supported Language Elements and Syntax ................................................. 18

Special Characters in Directory Names .............................................................. 20

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.

In the July 2015 release of SAS 9.4, the SAS Scoring Accelerator for Hadoop, SAP HANA, and Teradata support model scoring using analytic stores. SAS Factory Miner’s HPSVM or HPFOREST components generate an analytic store file and the scoring model program that are needed for scoring. The analytic store file is used by the publishing macros instead of the property file that contains model inputs or outputs.

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 or SAS Factory 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 or analytic store file, 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 one of the following methods to generate scoring model information:

- 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.

- Use SAS Factory Miner to run either the HPFOREST or HPSVM components to create the analytic store files and score files for analytic store scoring.

  For more information, see *SAS Factory Miner: User’s Guide*.

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 145 and “%INDSP_PUBLISH_MODEL Macro Syntax” on page 148.

3. Run the run model macro. This macro runs the scoring model and creates an output table.
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 18.

For more information, see “Running Scoring Models in SPD Server” on page 145 and “%INDSP_PUBLISH_MODEL Macro Syntax” on page 148.

4 Use PROC SQL to query the output table data.

For more information, see “Scoring Output” on page 151.
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, al-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
ARCSIN
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*.

<table>
<thead>
<tr>
<th>Character</th>
<th>How to Represent</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank¹</td>
<td><code>%str( )</code></td>
</tr>
<tr>
<td>*²</td>
<td><code>%str(*)</code></td>
</tr>
<tr>
<td>;</td>
<td><code>%str(;)</code></td>
</tr>
<tr>
<td>,</td>
<td><code>%str(,)</code></td>
</tr>
<tr>
<td>=</td>
<td><code>%str(=)</code></td>
</tr>
<tr>
<td>+</td>
<td><code>%str(+)</code></td>
</tr>
<tr>
<td>-</td>
<td><code>%str(-)</code></td>
</tr>
<tr>
<td>&gt;</td>
<td><code>%str(&gt;)</code></td>
</tr>
<tr>
<td>&lt;</td>
<td><code>%str(&lt;)</code></td>
</tr>
<tr>
<td>^</td>
<td><code>%str(^)</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td><code>%str(&amp;)</code></td>
</tr>
<tr>
<td>#</td>
<td><code>%str(#)</code></td>
</tr>
<tr>
<td>/</td>
<td><code>%str(/)</code></td>
</tr>
<tr>
<td>~</td>
<td><code>%str(~)</code></td>
</tr>
<tr>
<td>%</td>
<td><code>%str(%%)</code></td>
</tr>
<tr>
<td>'</td>
<td><code>%str(’)</code></td>
</tr>
<tr>
<td>“</td>
<td><code>%str(“)</code></td>
</tr>
<tr>
<td>(</td>
<td><code>%str(())</code></td>
</tr>
<tr>
<td>)</td>
<td><code>%str(%)</code></td>
</tr>
<tr>
<td>Character</td>
<td>How to Represent</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td>~</td>
<td>%str(~)</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Code Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:\temp\Sales(part1)</td>
<td>c:\temp\Sales%str(%){part1}%str(%)</td>
</tr>
<tr>
<td>c:\temp\Drug &quot;trial&quot; X</td>
<td>c:\temp\Drug %str(%)trial%str(%) X</td>
</tr>
<tr>
<td>c:\temp\Disc's 50% Y</td>
<td>c:\temp\Disc%str(%)s 50%str(%) Y</td>
</tr>
<tr>
<td>c:\temp\Pay,Emp=Z</td>
<td>c:\temp\Pay%str(,)Emp%str(=)Z</td>
</tr>
</tbody>
</table>
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**

![Data Mining Process Flow Diagram](image)

**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**

<table>
<thead>
<tr>
<th>File or Folder</th>
<th>Description</th>
</tr>
</thead>
</table>
| score.sas      | SAS language score code created by SAS Enterprise Miner. This code can be used directly in a SAS program. A sample program looks like this:  
    data testout ;  
    set simpletest.scoredata ;  
    %include "c:\models\simpletest\score.sas";  
    run; |
<table>
<thead>
<tr>
<th>File or Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>score.xml</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Restriction:</strong> The maximum number of input variables for a scoring function is 128.</td>
</tr>
<tr>
<td>emoutput.xml</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The emoutput.xml file is not used by the scoring publishing macro.</td>
</tr>
<tr>
<td>scoredata.sas7bdat</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The scoredata.sas7bdat file is not used by the scoring publishing macro.</td>
</tr>
<tr>
<td>traindata.sas7bdat</td>
<td>A ten-row sample table of the training data set showing typical cases of the input attributes used to develop the score code.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The traindata.sas7bdat file is not used by the scoring publishing macro.</td>
</tr>
<tr>
<td>Format Catalog</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Role</th>
<th>Type</th>
<th>Prefix</th>
<th>Key</th>
<th>Suffix</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>N</td>
<td>P_</td>
<td>Target variable name</td>
<td></td>
<td>P_amount</td>
</tr>
<tr>
<td>Probability</td>
<td>N</td>
<td>P_</td>
<td>Target variable name</td>
<td>Predicted event</td>
<td>P_purchaseYES P_purchaseNO</td>
</tr>
<tr>
<td>Classification</td>
<td>$</td>
<td>I_</td>
<td>Target variable name</td>
<td></td>
<td>I_purchase</td>
</tr>
<tr>
<td>Expected Profit</td>
<td>N</td>
<td>EP_</td>
<td>Target variable name</td>
<td></td>
<td>EP_conversion</td>
</tr>
<tr>
<td>Expected Loss</td>
<td>N</td>
<td>EL_</td>
<td>Target variable name</td>
<td></td>
<td>EL_conversion</td>
</tr>
<tr>
<td>Return on Investment</td>
<td>N</td>
<td>ROI_</td>
<td>Target variable name</td>
<td></td>
<td>ROI_conversion</td>
</tr>
<tr>
<td>Decision</td>
<td>$</td>
<td>D_</td>
<td>Target variable name</td>
<td></td>
<td>D_conversion</td>
</tr>
<tr>
<td>Decision Tree Leaf</td>
<td>N</td>
<td><em>NODE</em></td>
<td></td>
<td></td>
<td><em>NODE</em></td>
</tr>
<tr>
<td>Cluster number or SOM</td>
<td>N</td>
<td><em>SEGMENT</em></td>
<td></td>
<td></td>
<td><em>SEGMENT</em></td>
</tr>
</tbody>
</table>

**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

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

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
Analytic Store Scoring

Introduction to Analytic Store Scoring

In the July 2015 release of SAS 9.4, the SAS Scoring Accelerator for Hadoop, SAP HANA, and Teradata support complex model scoring, using analytic stores. With complex models, the traditional descriptive code is no longer adequate to express such models due to the huge amount and variety of information that is needed. Analytic stores present themselves as the alternative to the old style of scoring with code generated by the underlying engine.

Three files are required for analytic store scoring by the SAS Scoring Accelerator:

• the analytic store file

  An analytic store is a binary file that contains information about the state of an analytic object. It stores information that enables you to load and restore the state of the analytic object and set it in a score-ready mode. The analytic store is transportable. That is, it can be produced on one host and consumed on others without the need of the traditional SAS export or import.

  By default, the filename is score.sasast. However, any filename can be specified.

• the score code

  The score code is a DS2 program.

  By default, the filename is score.sas. However, any filename can be specified.

• an XML file that contains the scoring variables and other properties that are used and created by the scoring code

  By default, the filename is score.xml. However, any filename can be specified.

For information about how to use the SAS Scoring Accelerator to perform analytic store scoring, see the SAS Scoring Accelerator section for your data source.
How to Generate Analytic Store Scoring Files for the SAS Scoring Accelerator

The scoring files that are required for analytic store scoring by the SAS Scoring Accelerator are generated by the SAS Factory Miner HPFOREST or HPSVM components and placed in a directory that is accessed by the scoring model macros.

In addition, the SAS Factory Miner HPFOREST or HPSVM components generate a ten-row sample of the scored data set and a ten-row sample table of the training data set.

For more information about the HPFOREST and HPSVM components, see SAS Factory Miner: User’s Guide.
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 nCluster to insert the three scoring files into a table. For more information, see “Scoring Files Table” on page 39.

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 39.

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 32.

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 34.

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=\texttt{username} PASSWORD=\texttt{password} DATABASE=\texttt{databasename}  
SERVER=\texttt{servername} <SCHEMA=\texttt{schemaname}>

**Arguments**

**USER=\texttt{username}**

specifies the Aster user name (also called the user ID) that is used to connect to the database.

**PASSWORD=\texttt{password}**

specifies the password that is associated with your Aster user ID.

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

**DSN=\texttt{datasourcename}**

specifies the configured Aster data source to which you want to connect.

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

**DATABASE=\texttt{databasename}**

specifies the Aster database that contains the tables and views that you want to access.

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

**SERVER=\texttt{servername}**

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

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

**SCHEMA=\texttt{schemaname}**

specifies the schema name for the database.

\textbf{Default} your default schema. To determine your default schema name, use the \texttt{show search path} command from the Aster Client Tool (ACT).

\textbf{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.

\textbf{Requirement} Any schema that is used must be in the search path.

\textbf{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-
derfined 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 20

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 37.

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

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 37.

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 `sascore_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.

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 `sascore_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.

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 32.

**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 37.

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

See “Special Characters in Directory Names” on page 20

---

**Model Publishing Macro Example**

```sas
%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 303.
• sasscore_score_io.xml. See “Example of an Input and Output Variables Scoring File” on page 323.
• sasscore_score_ufmt.xml. See “Example of a User-Defined Formats Scoring File” on page 330.

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 39.

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 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 INDCCONN macro variable. See Appendix 1, “Scoring File Examples,” on page 303 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:

  - `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"
  ```
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`.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model</td>
<td>VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the sasscore_modelname.ds2 file</td>
<td>BLOB(209708800)</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the sasscore_modelname_ufmt.xml file</td>
<td>BLOB(209708800)</td>
</tr>
<tr>
<td>ModelOwner</td>
<td>contains the name of the user who published the model</td>
<td>VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC</td>
</tr>
<tr>
<td>ModelUpdated</td>
<td>contains the date and time that the model was published</td>
<td>TIMESTAMP(6)</td>
</tr>
</tbody>
</table>

**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.

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

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.

```sql
select id, em_classification, em_eventprobability, em_probability
from sas_score (on score_intab model('reg') model_schema('mysch'));
```
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 42.

- 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 47.
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/sql/lib/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/sqlib/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/sqlib/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 46.

**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 54.

`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 46.

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/sql/lib/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"
,"STALKCSBR"
,"STALKRROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_CLASSIFICATION",
allmush1_em_eventprobability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCSBR"
,"STALKRROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_EVENTPROBABILITY",
allmush1_em_probability("BRUISES"
,"CAPCOLOR"
,"GILLCOLO"
,"GILLSIZE"
,"HABITAT"
,"ODOR"
,"POPULAT"
,"RINGNUMB"
,"RINGTYPE"
,"SPOREPC"
,"STALKCSBR"
,"STALKRROO"
,"STALKSAR"
,"STALKSHA"
,"VEILCOLO")
  as "EM_PROBABILITY"
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.

```sas
%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.

```sql
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.

```sql
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 48.

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 52 and “DB2 Scoring Files” on page 51.

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 50.

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model</td>
<td>VARCHAR(128) NOT NULL PRIMARY KEY</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the sasscore_modelname.ds2 file</td>
<td>BLOB(4M) NOT NULL</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the sasscore_modelname_ufmt.xml file</td>
<td>BLOB(4M)</td>
</tr>
<tr>
<td>ModelMetadata</td>
<td>Reserved by SAS for future use</td>
<td>BLOB(4M)</td>
</tr>
<tr>
<td>ModelUUID*</td>
<td>contains the UUID of the source model</td>
<td>VARCHAR (36)</td>
</tr>
<tr>
<td>Notes*</td>
<td>contains additional information that describes the source model</td>
<td>VARCHAR (512)</td>
</tr>
</tbody>
</table>

* 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 52.

2. Run the %INDB2_CREATE_MODELTABLE macro.

For more information, see “%INDB2_CREATE_MODELTABLE Macro Syntax” on page 49.

**%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 54

**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 48 and “Running the %INDB2_PUBLISH_MODEL Macro” on page 52.

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 'PROVIDER=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 49.

**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 303 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 52.

4. If you use the SAS Embedded Process, run the %INDB2_CREATE_MODELTABLE macro.
   For more information, see “Creating a Model Table” on page 48.

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 54.

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=userid 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
, MODELNAME=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 20

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 43.

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 43.
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 42

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 47

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 49.

DATASETP= 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 DATASETP= 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 49.

**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 52.

**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 58

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.

<table>
<thead>
<tr>
<th>Default</th>
<th>120 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>This argument is valid only when using the scoring functions. It has no effect if you specify MECHANISM=EP.</td>
</tr>
<tr>
<td>Interactions</td>
<td>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.</td>
</tr>
<tr>
<td>Tip</td>
<td>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.</td>
</tr>
<tr>
<td>See</td>
<td>INITIAL_WAIT= argument</td>
</tr>
</tbody>
</table>

**OUTDIR=** *diagnostic-output-directory*

specifies a directory that contains diagnostic files.

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

| Tip | This argument is useful when testing your scoring models. |
| See | “Special Characters in Directory Names” on page 20 |

**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 *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 *SAS In-Database Products: Administrator's Guide*. 
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 62.

• 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 67.
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 66.

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 75.

`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 66.

**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 norerrorstop;
connect to greenplm (<connection options>);
execute (create table scoretab {
    ID integer
,   EM_SEGMENT float
,   EM_EVENTPROBABILITY float
,   EM_PROBABILITY float
```

Using Scoring Functions to Run Scoring Models 63
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.

  ```sql
  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`.

  ```sql
  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.nsrename)= '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 `allmushl_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`.

```sql
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")
  as "EM_PROBABILITY";
```
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_eventprobability
(
  "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
)
```
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.

```sql
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 68.

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 73 and “Greenplum Scoring Files” on page 72.

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 70.

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model</td>
<td>VARCHAR(128) NOT NULL PRIMARY KEY</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the sasscore_modelname.ds2 file</td>
<td>BYTEA NOT NULL</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the sasscore_modelname_ufmt.xml file</td>
<td>BYTEA</td>
</tr>
<tr>
<td>ModelMetadata</td>
<td>Reserved by SAS for future use</td>
<td>BYTEA</td>
</tr>
<tr>
<td>ModelUUID*</td>
<td>contains the UUID of the source model</td>
<td>VARCHAR (36)</td>
</tr>
<tr>
<td>Notes*</td>
<td>contains additional information that describes the source model</td>
<td>VARCHAR (512)</td>
</tr>
</tbody>
</table>

* 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 73.

2. Run the %INDGP_CREATE_MODELTABLE macro.

   For more information, see “%INDGP_CREATE_MODELTABLE Macro Syntax” on page 69.

**%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 75

**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 68 and “Running the %INDGP_PUBLISH_MODEL Macro” on page 73.

Here is an example using PROC SQL.

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

%indgp_publish_model
dir= C:\models,
modalname= 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"

Chapter 7 • SAS Scoring Accelerator for Greenplum
"stalkcbr"
,"stalksbr"
from model.almush02),
'select models2, 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
<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 MODELS22, 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.

**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.

**model-name**

specifies the name of the model.

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.
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 303 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:
  ```sql
  select modelname from <schema.sas-model-table;`
  ```

- Run a PROC SQL query from SAS.
  ```sql
  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 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.
  ```bash
  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 _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 68.

- 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 73.
4. Run the %INDGP_PUBLISH_MODEL macro. For more information, see “%INDGP_PUBLISH_MODEL Macro Syntax” on page 75.

   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:
Arguments

**USER=</user_name>**
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 the 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 nonalphabetic 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 nonalphabetic 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
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

```sas
%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 20

**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 63.

** 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 63.

**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 62

**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 67

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 69.

**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 73.

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 63.

Tip  This argument is useful when testing your scoring models.

See  “Special Characters in Directory Names” on page 20

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 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 SAS In-Database Products: Administrator’s Guide.
Chapter 8
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.

You can create traditional scoring models by using the SAS Enterprise Miner Score Code Export node. In the July 2015 release of SAS 9.4, the SAS Scoring Accelerator for Hadoop supports analytic store scoring. SAS Factory Miner components, the HPFOREST and HPSVM components, generate the analytic store file and the SAS scoring model program, and format catalog that are used in analytic store scoring.

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 traditional scoring model by using SAS Enterprise Miner or an analytic store scoring model using the SAS Factory Miner HPFOREST or HPSVM component.

2. Start SAS.

   - 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 90 and PROC HDMD in SAS/ACCESS for Relational Databases: Reference.

4. Specify the Hadoop connection attributes.
   - %let indconn= user=myuserid;
   - For more information, see “INDCONN Macro Variable” on page 81.

5. Run the %INDHD_PUBLISH_MODEL macro.
   - With traditional model scoring, the %INDHD_PUBLISH_MODEL performs the following tasks using 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:
     - 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 SAS/ACCESS Interface to Hadoop to copy the sasscore_modelname.ds2 and sasscore_modelname_ufmt.xml scoring files to the HDFS.
   - With analytic store scoring, the %INDHD_PUBLISH_MODEL macro takes the files that are created by the SAS Factory Miner HPFOREST or HPSVM components: the DS2 scoring model program (score.sas file), the analytic store file (score.sasast file), and (if the training data includes SAS user-defined formats) a format catalog, and copies them to the HDFS.
   - For more information, see “%INDHD_PUBLISH_MODEL Macro Syntax” on page 82.
6. 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 85.

7. Submit an SQL query against the output file.
   For more information, see “Scoring Output” on page 92.

---

**INDCONN Macro Variable**

The INDCONN macro variable is used to provide credentials to connect to the Hadoop HDFS and MapReduce. 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.

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

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

**Arguments**

**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.

**Requirement**

Either the SAS_HADOOP_CONFIG_PATH environment variable must be defined or the HADOOP_CFG argument must be specified before you run the %INDHD_PUBLISH_MODEL or the %INDHD_RUN_MODEL macro.

**Interaction**

The HADOOP_CFG= argument takes precedence over the SAS_HADOOP_CONFIG_PATH environment variable.

**See**

SAS_HADOOP_CONFIG_PATH environment variable in the SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS.
%INDHD_PUBLISH_MODEL Macro Syntax

%INDHD_PUBLISH_MODEL
  (DIR=input-directory-path, MODELNAME=name, MODELDIR=hdfs-directory-path,
   DATASTEP=score-program-filename, XML=xml-filename,
   STORE=analytic-store-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, the analytic store file (if analytic store scoring), and the optional format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM components. This directory contains the score.sas file, the score.xml file or (if analytic store scoring) score.sasast 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=, STORE= (if analytic store scoring), and FMTCAT= (if needed) arguments.

See “Special Characters in Directory Names” on page 20

**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 path name.

See “Special Characters in Directory Names” on page 20
**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 or the HPSVM or HPFOREST component.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.sas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPSVM or HPFOREST component can be used.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The scoring model program file must be located in the DIR directory.</td>
</tr>
<tr>
<td>Interaction</td>
<td>If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPSVM or HPFOREST component, you do not need to specify the DATASTEP= argument.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>score.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used. This argument is not applicable for analytic store models generated by SAS Factory Miner.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The properties XML file must be located in the DIR directory.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
</tbody>
</table>

If you specify both the XML= and STORE= arguments, the XML= argument is ignored and a warning is printed to the SAS log. The XML= argument is used for traditional model scoring. The STORE= argument is used for analytic store scoring.

See “STORE=analytic-store-filename” on page 83

**STORE=analytic-store-filename**
specifies the filename of the analytic store.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.sasast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only analytic store files that are produced by the SAS Factory Miner HPFOREST or HPSVM components can be used.</td>
</tr>
<tr>
<td>Interaction</td>
<td>If an analytic store file exists in the input directory (DIR=), the %INDHD_PUBLISH_MODEL macro attempts to publish that object. Alternatively, you can specify the name of the analytic store in the STORE= argument.</td>
</tr>
</tbody>
</table>

If you specify both the XML= and STORE= arguments, the XML= argument is ignored and a warning is printed to the SAS log. The XML= argument is used for traditional model scoring. The STORE= argument is used for analytic store scoring.
See “DIR=input-directory-path” on page 82

“XML=xml-filename” on page 83

**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 or the SAS Factory Miner HPFOREST or HPSVM component can be used.

**Interactions**
If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component, 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*.

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

The analytic store file includes format catalog information. However, you can specify a different format catalog by using the FMTCAT= argument.

**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=\texttt{input-filename}.SASHDMD | INPUTTABLE=\texttt{input-table-name} \\
  , SCOREPGM=\texttt{model\_score\_program\_ds2\_file} \\
  | , STORE=\texttt{analytic\_store\_filename}<, CUSTJAR=\texttt{custom\_reader\_jar\_filename}> \\
  <, OUTDATADIR=\texttt{hdfs\_directory\_path}> \\
  <, OUTMETADIR=\texttt{hdfs\_directory\_path}> \\
  <, INFILETYPE=\texttt{type}> \\
  <, INPUTFILE=\texttt{input\_file\_name}> \\
  <, OUTFILEDELMITER=\texttt{file\_delimiter}> \\
  <, OUTTEXTQUALIFIER=\texttt{text\_qualifier}> \\
  <, OUTFILETYPE=\texttt{output\_file\_type}> \\
  <, OUTRECORDFORMAT=\texttt{output\_record\_format}> \\
  <, FORMATFILE=\texttt{user\_defined\_format\_filename}> \\
  <, FORCEOVERWRITE=\texttt{TRUE} | \texttt{FALSE}> \\
  <, KEEP=\texttt{variable\_keep\_list}> \\
  <, KEEPFILENAME=\texttt{keep\_list\_configuration\_filename}> \\
  <, TRACE=\texttt{YES} | \texttt{NO}> \\
  );

Arguments

\textbf{INMETANAME=\texttt{input-filename}.SASHDMD}

specifies the HDFS full path of the input metadata file (.sashdmd file).

\textbf{Restriction}

\texttt{INMETANAME} and \texttt{INPUTTABLE} are mutually exclusive.

\textbf{Requirement}

The metadata file must already exist or must be generated with PROC HDMD before running the \texttt{%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 \texttt{LIBNAME} statement that contains the \texttt{HDFS\_DATADIR=} and \texttt{HDFS\_METADIR} options. In this instance, metadata files are automatically generated.

\textbf{Interaction}

This file is read by the MapReduce job.

\textbf{See}

“Creating a Metadata File for the Input Data File” on page 90

\texttt{PROC HDMD} in \textit{SAS/ACCESS for Relational Databases: Reference}

\textbf{INPUTTABLE=\texttt{input-table-name}}

specifies the name of a Hive table that you want to use as input data.

\textbf{Restriction}

\texttt{INMETANAME} and \texttt{INPUTTABLE} are mutually exclusive.

\textbf{Requirement}

A SAS/ACCESS Hadoop engine libref must be defined (for example, \texttt{inputtable=hive.mytable}).

\textbf{Interaction}

The \texttt{%INDHD_RUN_MODEL} macro automatically detects the file type and creates a temporary HDMD file for the execution.
SCOREPGM=\textit{model\_score\_program\_ds2\_file}

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

\begin{itemize}
  \item Interaction \texttt{SCOREPGM=} and \texttt{STORE=} are mutually exclusive.

  \item See \textquoteright\textit{STORE=\textbackslash analytic\textbackslash store\textbackslash filename\textquoteright\ on page 86}
\end{itemize}

\texttt{STORE=}\textit{\textbackslash analytic\textbackslash store\textbackslash filename}

specifies the filename of the analytic store.

\begin{itemize}
  \item Default \texttt{score.sasast}

  \item Restriction Only analytic store files that are produced by the HPFOREST or HPSVM components can be used.

  \item Interaction \texttt{SCOREPGM=} and \texttt{STORE=} are mutually exclusive.

  \item See \textquoteright\textit{SCOREPGM=\textit{model\_score\_program\_ds2\_file\textquoteright\ on page 86}
\end{itemize}

\texttt{CUSTJAR=}\textit{\textbackslash custom\textbackslash reader\textbackslash jar\textbackslash filename}

specifies the user-provided JAR filename that contains custom readers.

\begin{itemize}
  \item Requirement The user provided JAR file should be store in a local folder.
\end{itemize}

\texttt{OUTDATADIR=}\textit{\textbackslash hdfs\textbackslash directory\textbackslash path}

specifies the name of the HDFS directory where the MapReduce job stores the output files.

\begin{itemize}
  \item Interaction The \textit{\textbackslash hdfs\textbackslash directory\textbackslash path\textquoteright\ overrides what is specified in the \texttt{<outputDir>} element in the input metadata file (.sashdmd file).
\end{itemize}

\texttt{OUTMETADIR=}\textit{\textbackslash hdfs\textbackslash directory\textbackslash path}

specifies the name of the HDFS directory where the MapReduce job stores the output file metadata.

\begin{itemize}
  \item Interaction The \textit{\textbackslash hdfs\textbackslash directory\textbackslash path\textquoteright\ overrides what is specified in the \texttt{<metaDir>} element in the input file metadata (.sashdmd file).
\end{itemize}

\texttt{INFILETYPE=}\textit{type}

specifies the type of input file. \textit{type} can be one of the following:

\begin{itemize}
  \item BINARY specifies a fixed record length file.

    \begin{itemize}
      \item Alias \texttt{FIXED}

      \item Note This type maps to the \texttt{com.sas.access.hadoop.ep.binar.FixedRecLenBinaryInputFormat} input format in the \texttt{<epInputFormat>} element in the input file metadata (.sashdmd file).
    \end{itemize}

  \item CUSTOM specifies a custom file.

    \begin{itemize}
      \item Note This type maps to the \texttt{com.sas.access.hadoop.ep.custom.CustomFileInputFormat}
    \end{itemize}
\end{itemize}
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).

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).

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).

SEQUENCE (HDMD)
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).

SEQUENCE
specifies a sequence 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).

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).

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=**\textit{input-filename}

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

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

**OUTFILEDELIMITER=**\textit{file-delimiter}

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

- `';`
- `'	`
- `^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=**\textit{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 OUTFILEDELIMITER and cannot be a newline (`0a'x`).

**Requirement** This option is valid only for DELIMITED. Other formats do not use it.

**OUTFILETYPE=**\textit{output-file-type}

specifies the output file type. \textit{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 84

**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:

```xml
<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.
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:

```sas
/***************************************************************
* 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:

```sas
/***************************************************************
* Assigns 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;
```
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 Hadoop Engine that specifies
* that the data and metadata 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 Hadoop 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 85.

#### 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.

```sas
/* Hadoop configuration file */
%let INDCONN=%str(HDFS_PORT=8120
    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;

---

**Reading and Writing HCatalog File Types**

**Overview of HCatalog File Types**

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 third maintenance release of SAS 9.4, the SAS In-Database Scoring Accelerator
for Hadoop uses HCatalog to read the native Hive file types Avro, ORC, Parquet, and
RCFile.

By default, an output file is delimited. You can use the %INDHD_RUN_MODEL
macro’s OUTRECORDFORMAT argument to write a binary file.

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 Scoring Accelerator for
  Hadoop is 0.13.0.
Support for HCatalog varies by vendor. For more information, see the documentation for your Hadoop vendor.

There are additional configuration steps that are needed when processing HCatalog files.

For more information, see “Additional Configuration Steps When Accessing Files That Are Processed Using HCatalog” on page 94.

**Additional Configuration Steps When Accessing Files That Are Processed Using HCatalog**

If you plan to access complex, non-delimited file types such as Avro or Parquet, through HCatalog, 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. In addition, you need to add the following HCatalog JAR files to the SAS_HADOOP_JAR_PATH environment variable:
  
  - `webhcat-java-client*.jar`
  - `hbase-storage-handler*.jar`
  - `hcatalog-server-extensions*.jar`
  - `hcatalog-core*.jar`
  - `hcatalog-pig-adapter*.jar`

- On the Hadoop cluster, the SAS Embedded Process for Hadoop install script automatically adds HCatalog JAR files to its configuration file. The HCatalog JAR files are added to the Embedded Process Map Reduce job class path during job submission.

  For information about installing and configuring the SAS Embedded Process for Hadoop, see *SAS In-Database Products: Administrator’s Guide*.

- You must include the HCatalog JAR files in your SAS_HADOOP_JAR_PATH environment variable.

  For more information about the SAS_HADOOP_JAR_PATH environment variable, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

- The hive-site.xml file must be in your SAS_HADOOP_CONFIG_PATH.

  For more information about the SAS_HADOOP_CONFIG_PATH environment variable, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

- If you are using IBM BigInsights v3.0 or Pivotal HD 2.x, do not use quotation marks around table names. The version of Hive that is used by IBM BigInsights v3.0 and Pivotal HD 2.x does not support quoted identifiers.
Using the MapReduce Job Logs to View DS2 Error Messages

When either MSGLEVEL=I or the SAS Scoring Accelerator for Hadoop fails, a message is written to the SAS log that contains an HTTP link to the MapReduce job log. Here is an example.

ERROR: Job job_1424277669708_2919 has failed. Please, see job log for details. Job tracking URL:
http://name.unx.company.com:8088/proxy/application_1424277669708_2919/

The HTTP link is to a site that contains the job summary. On the job summary page, you can see the number of failed and successful tasks. If you click on the failed tasks, you see a list of task attempts. A log is assigned to each attempt. Once in the log page, you are able to see the error messages.

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.
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 98.

- 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 102.

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 110. For more information about the scoring functions, see “Scoring Function Names” on page 98 and “Viewing the Scoring Functions” on page 99.

2. Use the scoring functions in any SQL query.

   For more information, see “Using Scoring Functions to Run Scoring Models” on page 98.

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 110.

`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`.

  ```plaintext
  nzsql database username password
  select function,createdate,functionssignature 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`.

  ```plaintext
  proc sql noerrorstop;
  connect to netezza (server=servername database=database
  username=username password=password);
  select * from connection to netezza
  (select function,createdate,functionssignature
   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`.

```sql
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",
   "STALKKROO",
   "STALKSAR",
   "STALKSHA",
   "VEILCOLO")
   as "EM_CLASSIFICATION",
   allmush1_em_eventprobability("BRUISES",
   "CAPCOLOR",
   "GILLCOLO",
   "GILLSIZE",
   "HABITAT",
   "ODOR",
   "POPULAT",
   "RINGNUMB",
   "RINGTYPE",
   "SPOREPC",
   "STALKCBR",
   "STALKKROO",
   "STALKSAR",
   "STALKSHA"
);```
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 99 was modified to create the SELECT statement in this example.

```sql
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 102.

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 110 and “Netezza Scoring Files” on page 108.

3. Execute the SAS_EP stored procedure to run the scoring model.
   For more information, see “SAS_EP Stored Procedure” on page 105.

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model.</td>
<td>NVARCHAR(128) NOT NULL</td>
</tr>
<tr>
<td>ModelSequence</td>
<td>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.</td>
<td>INTEGER NOT NULL</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the sasscore_modelname.ds2 file.</td>
<td>NVARCHAR (8000)</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the sasscore_modelname_ufmt.xml file.</td>
<td>NVARCHAR (8000)</td>
</tr>
<tr>
<td>ModelUUID*</td>
<td>contains the UUID of the source model.</td>
<td>VARCHAR (36)</td>
</tr>
<tr>
<td>Notes*</td>
<td>contains additional information that describes the source model.</td>
<td>VARCHAR (512)</td>
</tr>
</tbody>
</table>

* 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_MODELTABLE macro 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 schema=myschema;`
   
   For more information, see the “INDCONN Macro Variable” on page 109.

2. Run the %INDNZ_CREATE_MODELTABLE macro.
   
   For more information, see “%INDNZ_CREATE_MODELTABLE Macro Syntax” on page 104.
%INDNZ_CREATE_MODELTABLE Macro Syntax

%INDNZ_CREATE_MODELTABLE
(<DATABASE=database-name>
 <MODELTABLE=model-table-name>
 <DBSCHEMA=schema-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

Requirements The model table name cannot contain a period (.). Otherwise, an
error occurs.

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 111

DBSCHEMA=schema-name
specifies the name of a Netezza schema to which the model table is published.

Restriction This argument is supported only on Netezza v7.0.3 or later.

Interaction The schema that is specified by the DBSCHEMA= argument takes
precedence over the schema that you specify in the INDCONN macro
variable. If you do not specify a schema in the DBSCHEMA= argument or the INDCONN macro variable, the default schema for the
target database is used.

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 `ssa.score_modelname.ds2` and `ssa.score_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 116.

*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=xxxxx 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 106.
**SAS_EP Stored Procedure Syntax**

NZRC..SAS_EP

```sas
(' 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 104.
**INPUT_COLUMNS=</column><...<column>>**
specifies one or more columns from the input table that are passed to the SAS Embedded Process.

<table>
<thead>
<tr>
<th>Default</th>
<th>All columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If the column name is case sensitive, you must use double quotation marks (for example, &quot;Profit&quot; &quot;Sales&quot; &quot;Margin&quot;).</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Default</th>
<th>Current database distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If the column name is case sensitive, you must use double quotation marks (for example, &quot;Profit&quot;, &quot;Sales&quot;, &quot;Margin&quot;).</td>
</tr>
</tbody>
</table>

**OUTPUT_TEMPORARY= YES | NO**
specifies whether the output table is temporary.

<table>
<thead>
<tr>
<th>Default</th>
<th>NO</th>
</tr>
</thead>
</table>

**JOURNAL_TABLE=</journal-database>.<journaltable>>**
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"). |

**PARTITION_BY=</column><...<column>>**
specifies one or more columns by which to partition the input.

<table>
<thead>
<tr>
<th>Default</th>
<th>No partitioning occurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If the column name is case sensitive, you must use double quotation marks (for example, &quot;Profit&quot;, &quot;Sales&quot;, &quot;Margin&quot;).</td>
</tr>
</tbody>
</table>

**ORDER_BY=</column><...<column>>**
specifies one or more columns by which to order the input.

<table>
<thead>
<tr>
<th>Default</th>
<th>No reordering occurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If the column name is case sensitive, you must use double quotation marks (for example, &quot;Profit&quot;, &quot;Sales&quot;, &quot;Margin&quot;).</td>
</tr>
</tbody>
</table>

**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 303 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**.

```sql
nzsqli 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**.

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

select * from connection to netezza
  (select modelname, modelsequence
   from sasmodetablename
   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.
INDCONN Macro Variable

The INDCCONN 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 INDCCONN macro variable before the %INDNZ_PUBLISH_MODEL or the %INDNZ_CREATE_MODELTABLE macros are invoked.

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

```
SERVER=server USER=user PASSWORD=password DATABASE=database
<SCHEMA=schema-name>
```

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.

**SCHEMA=<‘>schema-name<‘>**

specifies the name of the schema where the models are published.

*Restriction* This argument is supported only on Netezza v7.0.3 or later.

*Interaction* The schema that is specified by the publishing macros’ DBSCHEMA= argument takes precedence over the schema that you specify in the INDCCONN macro variable. If you do not specify a schema in the DBSCHEMA= argument or the INDCCONN macro variable, the default schema for the target database is used.

*Tip* The INDCCONN 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.
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 109.

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

   For more information, see “Creating a Model Table” on page 102.

5. Run the %INDNZ_PUBLISH_MODEL macro.

   For more information, see “%INDNZ_PUBLISH_MODEL Macro Syntax” on page 111.

   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.
%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>
    <, DBCOMPIL=database-name>
    <, DBJAZLIB=database-name>
    <, DBSCHEMA=schema-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 20

**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 .ds2 and .xml 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 98.

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 98.

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 98

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 102

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.

Requirements

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 104.

The model table name cannot contain a period (.). Otherwise, an error occurs.

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.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.xml</th>
</tr>
</thead>
</table>

### 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.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>You must specify the DATABASE= argument if you use the SAS Embedded Process.</th>
</tr>
</thead>
</table>

### 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 110.

### 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.

<table>
<thead>
<tr>
<th>Default</th>
<th>SASLIB</th>
</tr>
</thead>
</table>

### 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.
Restriction  | This argument is ignored when MECHANISM=EP.

**DBSCHEMA=**schema-name

specifies the name of a Netezza schema to which the model is published.

Restriction  | This argument is supported only on Netezza v7.0.3 or later.

Interaction  | The schema that is specified by the DBSCHEMA= argument takes precedence over the schema that you specify in the INDCONN macro variable. If you do not specify a schema in the DBSCHEMA= argument or the INDCONN macro variable, the default schema for the target database is used.

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.

Interactions  | 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.
Running the `%INDNZ_PUBLISH_MODEL` Macro

<table>
<thead>
<tr>
<th>Default</th>
<th>FENCED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions</td>
<td>This argument is ignored when MECHANISM=EP.</td>
</tr>
<tr>
<td></td>
<td>The MODE= argument is supported for Netezza 6.0. The argument is</td>
</tr>
<tr>
<td></td>
<td>ignored for previous versions of Netezza.</td>
</tr>
<tr>
<td>Tip</td>
<td>There are limited resources available in Netezza when you run in</td>
</tr>
<tr>
<td></td>
<td>fenced mode. For example, there is a limit to the number of columns</td>
</tr>
<tr>
<td></td>
<td>available.</td>
</tr>
<tr>
<td>See</td>
<td>“Modes of Operation” on page 115</td>
</tr>
</tbody>
</table>

**IDCASE= UPPERCASE | LOWERCASE**  

Specifies whether the variable names in the generated sample SQL code (SampleSQL.txt) appear in uppercase or lowercase characters.

<table>
<thead>
<tr>
<th>Default</th>
<th>UPPERCASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>This argument is ignored when MECHANISM=EP.</td>
</tr>
<tr>
<td>Tip</td>
<td>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.</td>
</tr>
<tr>
<td>See</td>
<td>“Viewing the Scoring Functions” on page 99 for more information about the SampleSQL.txt file</td>
</tr>
</tbody>
</table>

**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 99.

| Tip           | This argument is useful when testing your scoring models.                |
| See           | “Special Characters in Directory Names” on page 20                      |

**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 *SAS In-Database Products: Administrator’s Guide.*
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 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 119.

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 121 and “Oracle Scoring Files” on page 124.

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 125.
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model</td>
<td>VARCHAR(128)</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the sasscore_modelname.ds2 file</td>
<td>BLOB not null</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the sasscore_modelname_ufmt.xml file</td>
<td>BLOB</td>
</tr>
<tr>
<td>ModelMetadata</td>
<td>reserved by SAS for future use</td>
<td>BLOB</td>
</tr>
<tr>
<td>ModelUUID*</td>
<td>contains the UUID of the source model</td>
<td>VARCHAR (36)</td>
</tr>
<tr>
<td>Notes*</td>
<td>contains additional information that describes the source model</td>
<td>VARCHAR (512)</td>
</tr>
</tbody>
</table>

* 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:

```sas
%let indconn = user=myuserid password=xxxxx path=ortest;
```

For more information, see the “INDCONN Macro Variable” on page 121.

2. Run the %INDOR_CREATE_MODELTABLE macro.

For more information, see “%INDOR_CREATE_MODELTABLE Macro Syntax” on page 120.
%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 122

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 contains 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 121.
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 122.

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

\[
\begin{align*}
\text{%INDOR_PUBLISH_MODEL} \\
\text{\texttt{(}} \text{DIR=} &\text{input-directory-path, MODELNAME=} \text{name} \\
\text{\texttt{<, MODELTABLE=} &\text{model-table-name}>} \\
\text{\texttt{<, DATASTEP=} &\text{score-program-filename}>} \\
\text{\texttt{<, XML=} &\text{xml-filename}>} \\
\text{\texttt{<, DATABASE=} &\text{database-name}>} \\
\text{\texttt{<, FMTCAT=} &\text{format-catalog-filename}>} \\
\text{\texttt{<, ACTION=} &\text{CREATE | REPLACE | DROP}>} \\
\text{\texttt{<, OUTDIR=} &\text{diagnostic-output-directory}>} \\
\text{\texttt{);}}
\end{align*}
\]

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 20

**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 124.

**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 120.

**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.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.xml</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>Default</th>
<th>The database specified in the INDCONN macro variable</th>
</tr>
</thead>
</table>

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 120.

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 121.

**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.

<table>
<thead>
<tr>
<th>Restriction</th>
<th>Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.</th>
</tr>
</thead>
</table>

| 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 124.

Tip This argument is useful when testing your scoring models.

See “Special Characters in Directory Names” on page 20

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 303 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`.

```sql
proc sql noerror stop;
  connect to oracle (user=username password=xxxxx 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
Here is an example using PROC SQL.

```sas
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.

```sas
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 127.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table-alias</code></td>
<td>specifies an alias for the input table name.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The table alias must be the same in the parallel hint and the FROM clause.</td>
</tr>
<tr>
<td><code>input-table</code></td>
<td>specifies the name of the input table that is used by the SASEPFUNC table function.</td>
</tr>
<tr>
<td><code>schema-name.model-table-name</code></td>
<td>specifies the fully qualified model table name.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The table name that you specify for this function must be the same table name that is used in the <code>%INDOR_CREATE.MODELTABLE</code> macro. For more information, see “Creating a Model Table” on page 119.</td>
</tr>
<tr>
<td><code>model-name</code></td>
<td>specifies the model name.</td>
</tr>
<tr>
<td>Requirement</td>
<td>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 122.</td>
</tr>
</tbody>
</table>
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 Oracle’s internal algorithms, it can be beneficial in some environments to change Oracle’s default behavior. There are two ways of doing so:

• Disable Oracle’s 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 Oracle’s internal load balancing for parallel execution does not harm a system in any way. However, influencing (changing) Oracle’s 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.
Overview of Running Scoring Models in SAP HANA ................................. 129
How to Run a Scoring Model in SAP HANA ............................................ 130
INDCONN Macro Variable ................................................................. 131
Creating the Model Table ................................................................. 133
  Overview .................................................................................. 133
  %INDHN_CREATE_MODELTABLE Run Process .................................. 133
  %INDHN_CREATE_MODELTABLE Macro Syntax ................................. 134
Running the %INDHN_PUBLISH_MODEL Macro ..................................... 135
  %INDHN_PUBLISH_MODEL Macro Run Process ............................... 135
  %INDHN_PUBLISH_MODEL Macro Syntax ...................................... 136
Running the %INDHN_RUN_MODEL Macro ........................................... 139
  %INDHN_RUN_MODEL Macro Run Process ..................................... 139
  %INDHN_RUN_MODEL Macro Syntax ............................................. 140
Scoring Output .............................................................................. 142
  Scoring Output Table ................................................................. 142
  Querying and Viewing the Scoring Output Table ............................ 143
SAP HANA Permissions .................................................................. 143

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.

You can create traditional scoring models by using the SAS Enterprise Miner Score Code Export node. In the July 2015 release of SAS 9.4, the SAS Scoring Accelerator for SAP HANA supports analytic store scoring. SAS Factory Miner components, the HPFOREST and HPSVM components, generate the analytic store file, the SAS scoring model program, and format catalog that are used in analytic store scoring.
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 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 133.

2. Run the %INDHN_PUBLISH_MODEL macro.
   - With traditional model scoring, the %INDHN_PUBLISH_MODEL macro performs the following tasks using 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:
     - 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 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.
   - With analytic store scoring, the %INDHN_PUBLISH_MODEL macro takes the files that are created by the SAS Factory Miner HPFOREST or HPSVM components: the DS2 scoring model program (score.sas file), the analytic store file (score.sasast file), and (if the training data includes SAS user-defined formats) a format catalog, and inserts their contents 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 135.

3. 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 139.

4. Submit an SQL query against the output table.
   For more information, see “Scoring Output” on page 142.
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 >
<SHEMA=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.

  **Interaction**
  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.

  **Interaction**
  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**
 préserves 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**
 préserves 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 "modelname.ds2" and "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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODELNAME</td>
<td>Contains the name of the model.</td>
<td>NVARCHAR(128) NOT NULL</td>
</tr>
<tr>
<td>MODELDS2</td>
<td>Contains the &quot;modelname.ds2&quot; file. Used by traditional model scoring.</td>
<td>NCLOB NOT NULL</td>
</tr>
<tr>
<td>MODELFORMATS</td>
<td>Contains the &quot;modelname_ufmt.xml&quot; file. Used by traditional model scoring.</td>
<td>NCLOB</td>
</tr>
<tr>
<td>MODELMETADATA</td>
<td>Contains the analytic store. Used by analytic store scoring.</td>
<td>NCLOB</td>
</tr>
<tr>
<td>MODELUUID*</td>
<td>Contains the UUID of the source model. Used by traditional model scoring.</td>
<td>NVARCHAR (36)</td>
</tr>
<tr>
<td>NOTES' *</td>
<td>Contains additional information that describes the source model.</td>
<td>NVARCHAR (512)</td>
</tr>
</tbody>
</table>

* 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 131.

2. Run the `%INDHN_CREATE_MODELTABLE` macro.

   For more information, see “%INDHN_CREATE_MODELTABLE Macro Syntax” on page 134.

   **%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 131.

   **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 136
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.
   For traditional scoring models, you can 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.

   For analytic store scoring models, you can also use the SAS Factory Miner HPFOREST or HPSVM component to create an analytic score file (score.sasast file) and the scoring model program (score.sas file).

2. 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 131.

3. Run the %INDHN_PUBLISH_MODEL macro.

   For more information, see “%INDHN_PUBLISH_MODEL Macro Syntax” on page 136.

   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>
  <, STORE= analytic-store-filename>
  <, DATABASE=database-name>
  <, FMTCAT=format-catalog-filename>
  <, ACTION=CREATE | REPLACE | DROP>
  <, OUTDIR=diagnostic-output-directory>
  );

Arguments

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

This is the directory that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM components. This directory contains the score.sas file, the score.xml file or (if analytic store scoring) score.sasast 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 filenames that are created by SAS Enterprise Miner, you must specify the DATASTEP=, XML=, STORE= (if analytic store scoring), and FMTCAT= arguments (if needed).

See
“Special Characters in Directory Names” on page 20

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 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 enclose the name in double quotation marks and set the INDCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.
DATASETP=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 or the SAS Factory Miner HPFOREST or HPSVM component.

Default score.sas

Restriction Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component can be used.

Interaction If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component, you do not need to specify the DATASETP= 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. This argument is not applicable for analytic store models generated by SAS Factory Miner.

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.

If you specify both the XML= and STORE= arguments, the XML= argument is ignored and a warning is printed to the SAS log. The XML= argument is used for traditional model scoring. The STORE= argument is used for analytic store scoring.

See “STORE=analytic-store-filename” on page 137

STORE=analytic-store-filename

specifies the filename of the analytic store.

Default score.sasast

Restriction Only analytic store files that are produced by the SAS Factory Miner HPFOREST or HPSVM components can be used.

Interactions If an analytic store file exists in the input directory (DIR=), the %INDHN_PUBLISH_MODEL macro attempts to publish that object. Alternatively, you can specify the name of the analytic store in the STORE= argument.

If you specify both the XML= and STORE= arguments, the XML= argument is ignored and a warning is printed to the SAS log. The
XML= argument is used for traditional model scoring. The STORE= argument is used to analytic store scoring.

**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 134.

**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 131.

**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 or the SAS Factory Miner HPFOREST or HPSVM component can be used.

**Interactions**

If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component, 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 analytic store file includes format catalog information. However, you can specify a different format catalog by using the FMTCAT= argument.

**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 20

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 131.

2. Run the %INDHN_RUN_MODEL macro.

   For more information, see “%INDHN_RUN_MODEL Macro Syntax” on page 140. 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 INDCCONN 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 134.

Interaction The database that is specified by the DATABASE argument takes precedence over the server, port, or instance value that you specify in the INDCCONN macro variable. For more information, see “INDCONN Macro Variable” on page 131.

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 INDCCONN macro variable’s PRESERVE_TAB_NAMES argument to YES.

See “INDCONN Macro Variable” on page 131

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 INDCCONN 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 134 and “%INDHN_RUN_MODEL Macro Syntax” on page 140.

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 131

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 141

EPTRACETABLE="schema">ep-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 141

“INDCONN Macro Variable” on page 131

**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 maximum string length for character input columns, such as CLOB and NCLOB, that otherwise do not have a specified maximum string length.

Default 1024

Restriction This argument applies only to CLOB and NCLOB columns.

Tip This argument can be used to avoid truncation of CLOB and NCLOB column values that contain more than 1024 characters.

## 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 140.
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`.

```sql
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 *SAS In-Database Products: Administrator’s Guide*. 
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_MODEL macros.

```sas
%let indconn = domain=mydomain server=myserver port=myport
                user=myuserid <password=XXXX>;
```

For more information, see “INDCONN Macro Variable” on page 147.

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 148.

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.

```sas
%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 147 and “INDDATA Macro Variable” on page 147.

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 150.

7. Submit an SQL query against the output table.

For more information, see “Scoring Output” on page 151.
**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.

**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
<, DATASET=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 20
**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.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.sas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node can be used.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The scoring model program file must be located in the INPUTDIR directory.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>score.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used.</td>
</tr>
<tr>
<td>Requirement</td>
<td>The properties XML file must be located in the INPUTDIR directory.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
</tbody>
</table>

**FMTCAT=**  
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.

<table>
<thead>
<tr>
<th>Default</th>
<th>FORMATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only format catalog files that are produced by the SAS Enterprise Miner Score Code Export node can be used.</td>
</tr>
<tr>
<td>Requirement</td>
<td>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 (<strong>libref.format-catalog-name</strong>), the catalog is expected to be found using the two-level name.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
</tbody>
</table>

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
does not cause 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=\textit{input-table-name}

specifies the name of the SPD Server table to be used as input to the scoring model.

\textbf{Restriction} The table must exist in the INDDATA domain.

\textbf{OUTDATA=\textit{output-table-name}}

specifies the name of the SPD Server table created as output from the scoring model.

\textbf{Default} \textit{modelname} \_\textit{OUT}

\textbf{Note} The table is created in the INDDATA domain.

\textbf{KEEP=\textit{variable-keep-list}}

specifies a list of variables to keep in the output table.

\textbf{Requirement} The variables in the list must be separated by spaces.

\textbf{Example} \texttt{keep=id \_em\_eventprobability \_em\_classification}

\textbf{OUTDIR=\textit{sample-sql-directory}}

specifies a directory that contains the sample SQL code (SampleSQL.txt).

\textbf{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.

\textbf{See} For more information about the SampleSQL.txt file, see “Querying Scoring Output Tables” on page 152.

“\textit{Special Characters in Directory Names}” on page 20

\textbf{THREADS=\textit{number-of-threads}}

specifies the number of threads to use when the model is run.

\textbf{Default} 1

\section*{Scoring Output}

\subsection*{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 \textit{modelname} \_\textit{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 \texttt{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 150.
Querying Scoring Output Tables

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

```sql
select * from mymodel_out;
```

```sql
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.

```sql
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.

```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";
```

Chapter 12 • SAS Scoring Accelerator for SPD Server
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_STALKKRO";
dcl double "S_G_STALKSAR";
dcl double "S_G_VEILCOLO";
dcl char(1) "U_TARGET";
dcl char(4) "_WARN_

method run();
set dataSPDS.almush01;
"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 saspsds;

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.
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 156.

- 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.

You can create traditional scoring models by using the SAS Enterprise Miner Score Code Export node. In the July 2015 release of SAS 9.4, the SAS Scoring Accelerator for Teradata supports analytic store scoring. SAS Factory Miner HPFOREST and
HPSVM components generate the analytic store file, the SAS scoring model program, and a format catalog that are used in analytic store scoring.

For more information, see Chapter 4, “Analytic Store Scoring,” on page 29.

For more information, see “Using the SAS Embedded Process to Run Scoring Models” on page 160.

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 171.

   For more information about the scoring functions that are created, see “Scoring Function Names” on page 156 and “Viewing the Scoring Functions” on page 157.

2. Use the scoring functions in any SQL query.

   For more information, see “Using Scoring Functions to Run a Scoring Model” on page 159.

---

**Scoring Function Names**

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

_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 171.

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

Using Scoring Functions to Run Scoring Models 157
• 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(
)
select id ,
as "EM_CLASSIFICATION",

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 157 was modified to create the SELECT statement in this example.

```sql
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 161.

2. Run the %INDTD_PUBLISH_MODEL macro.
   
   • With traditional model scoring, the %INDTD_PUBLISH_MODEL macro performs the following tasks using 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.
   
   • 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 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.

   With analytic store scoring, the %INDTD_PUBLISH_MODEL macro takes the files that are created by the SAS Factory Miner HPFOREST or HPSVM components: the DS2 scoring model program (score.sas file), the analytic store file (score.sasast file), and (if the training data includes SAS user-defined formats) a format catalog, and inserts their contents into the model table that was created using the %INDHN_CREATE_MODELTABLE macro.

   For more information, see “Running the %INDTD_PUBLISH_MODEL Macro” on page 171 and “Teradata Scoring Files” on page 170.

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

   For more information, see “SAS_SCORE_EP Stored Procedure” on page 163.
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`.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
<td>contains the name of the model</td>
<td>VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC</td>
</tr>
<tr>
<td>ModelDS2</td>
<td>contains the <code>sasscore_modelname.ds2</code> file</td>
<td>BLOB(209708800)</td>
</tr>
<tr>
<td>ModelFormats</td>
<td>contains the <code>sasscore_modelname_ufmt.xml</code> file</td>
<td>BLOB(209708800)</td>
</tr>
<tr>
<td>ModelOwner**</td>
<td>contains the name of the user who published the model</td>
<td>VARCHAR(128) CHARACTER SET UNICODE CASESPECIFIC</td>
</tr>
<tr>
<td>ModelUpdated**</td>
<td>contains the date and time that the model was published</td>
<td>TIMESTAMP(6)</td>
</tr>
<tr>
<td>ModelUUID*</td>
<td>contains the UUID of the source model</td>
<td>VARCHAR(36) CHARACTER SET UNICODE NOT CASESPECIFIC</td>
</tr>
<tr>
<td>Notes*</td>
<td>contains additional information that describes the source model</td>
<td>VARCHAR(512) CHARACTER SET UNICODE NOT CASESPECIFIC</td>
</tr>
</tbody>
</table>

* 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. This column is 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:

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

For more information, see the “INDCONN Macro Variable” on page 172.

2. Run the `%INDTD_CREATE_MODELTABLE` macro.

For more information, see “%INDTD_CREATE_MODELTABLE Macro Syntax” on page 162.

**%INDTD_CREATE_MODELTABLE Macro Syntax**

```sas
%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**: If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

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**: If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.
- **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 173

**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 178.

**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="grottov."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 164.

### 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.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

"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_PUBLISH_MODEL macro.

**Requirements**  
The model name must be the same as the one specified in the %INDTD_PUBLISH_MODEL macro’s MODELNAME argument.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.

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...

```sql
'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

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

"`output-table-name`"

specifies the name of the table to be created or the table that currently exists.

#### Requirement

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.

#### 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.

```plaintext
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.

**Requirements**
- If more than one column is specified, column names must be separated with spaces.
- If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum column name length is 128 characters, and it must be a valid Teradata column name. Otherwise, the maximum length is 30 characters.

**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**

**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.

**Requirements**
- If more than one column is specified, column names must be separated with commas.
- If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum column name length is 128 characters, and it must be a valid Teradata column name. Otherwise, the maximum length is 30 characters.

**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="database-name"."journal-table-name"**
specifies the name and location 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.

"database-name"
specifies the database where the journal table is created.

**Default**
- Current database

**Requirement**
- If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is
128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

"journal-table-name"
specifies the name of the journal table.

Requirement If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.

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).


ORDERBY=column-name,…, column-name
specifies one or more columns to use for the LOCAL ORDER BY (BY groups) clause.

Requirements If more than one column is specified, column names must be separated with commas.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum column name length is 128 characters, and it must be a valid Teradata column name. Otherwise, the maximum length is 30 characters.

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

Requirements If more than one column is specified, column names must be separated with commas.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum column name length is 128 characters, and it must be a valid Teradata column name. Otherwise, the maximum length is 30 characters.

SQLTRACE="database-name"."table-name"
specifies the name and location of a table to hold the generated SQL code.

"database-name"
specifies the database where the journal table is created.
If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

"table-name"

specifies the name of the table.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.

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 should be qualified with a database name. If a table name is not qualified with a database name, how the table name is resolved depends on which version of Teradata is used.

- 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 303 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`.

  ```sql
  bteq .logon myserver/myuserid,mypassword
  select modelname, modelowner, modeluuid 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,modeluuid
    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.

<table>
<thead>
<tr>
<th>Action Performed</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides the status of the SAS Embedded Process.</td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('status', :A); *</td>
</tr>
<tr>
<td></td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'status', :A); **</td>
</tr>
<tr>
<td></td>
<td>CALL SQLJ.SERVERCONTROL ('SAS', 'status', :A); ***</td>
</tr>
<tr>
<td>Stops new queries from being started. Queries that are currently running continue to run until they are complete.</td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('disable', :A); *</td>
</tr>
<tr>
<td></td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'disable', :A); **</td>
</tr>
<tr>
<td></td>
<td>CALL SQLJ.SERVERCONTROL ('SAS', 'disable', :A); ***</td>
</tr>
<tr>
<td>Enables new queries to start running.</td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('enable', :A); *</td>
</tr>
<tr>
<td></td>
<td>CALL DBCEXTENSION.SERVERCONTROL ('SAS', 'enable', :A); **</td>
</tr>
<tr>
<td></td>
<td>CALL SQLJ.SERVERCONTROL ('SAS', 'enable', :A); ***</td>
</tr>
</tbody>
</table>

* 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 traditional scoring model by using SAS Enterprise Miner or an analytic store scoring model using the SAS Factory Miner HPFOREST or HPSVM component.

2. Test your connection to Teradata with a local utility such as BTEQ.

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 172.

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

   For more information, see “Creating a Model Table” on page 161.

5. 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 173.

**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

**Requirements** You must specify the DATABASE= argument if you use the SAS Embedded Process.
If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

**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>
 <, STORE= analytic-store-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, the analytic store file (if analytic store scoring), and the format catalog are located.

This is the directory that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM components. This directory contains the score.sas file, the score.xml file or (if analytic store scoring) score.sasast 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= or (if analytic store scoring) STORE=, and (if needed) FMTCAT= arguments.

**See** “Special Characters in Directory Names” on page 20

**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 .ds2 and .xml scoring filenames.

**Restriction** The scoring function name is a combination of the model name and the output variable name. If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum scoring function name length is 128 characters. Otherwise, the maximum length is 30 characters. For more information, see “Scoring Function Names” on page 156.
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 156.

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 156

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 160

Default If either a score.sasast file exists or you specify STORE=analytic-store-filename, the default is EP. Otherwise, the default is STATIC.

Interaction If you specify MECHANISM=STATIC and either a score.sasast file exists or you specify STORE=analytic-store-filename, an error occurs.

See “STORE=analytic-store-filename” on page 175

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.
Requirements

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 162.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum table name length is 128 characters, and it must be a valid Teradata table name. Otherwise, the maximum length is 30 characters.

**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 or the SAS Factory Miner HPFOREST or HPSVM component.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.sas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only DATA step programs that are produced by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component can be used.</td>
</tr>
<tr>
<td>Interaction</td>
<td>If you use the default score.sas file that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component, you do not need to specify the DATASTEP= argument.</td>
</tr>
</tbody>
</table>

**XML=xml-filename**

specifies the name of the properties XML file that was created by the SAS Enterprise Miner Score Code Export node.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions</td>
<td>Only XML files that are produced by the SAS Enterprise Miner Score Code Export node can be used. This argument is not applicable for analytic store models generated by SAS Factory Miner.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
</tbody>
</table>

| 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. |

**STORE=analytic-store-filename**

specifies the filename of the analytic store.

<table>
<thead>
<tr>
<th>Default</th>
<th>score.sasast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Only analytic store files that are produced by the SAS Factory Miner HPFOREST or HPSVM components can be used.</td>
</tr>
<tr>
<td>Interactions</td>
<td>If an analytic store file exists in the input directory (DIR=), the <code>%INDTD_PUBLISH_MODEL</code> macro attempts to publish that object.</td>
</tr>
</tbody>
</table>
Alternatively, you can specify the name of the analytic store in the STORE= argument.

If you specify MECHANISM=STATIC and either a score.sasast file exists or you specify STORE=analytic-store-filename, an error occurs.

See

“DIR=input-directory-path” on page 173

“MECHANISM=STATIC | EP” on page 174

**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 162.

If you are using Teradata 14.10 or later, excluding the first release of Teradata 15, the maximum database name length is 128 characters, and it must be a valid Teradata database name. Otherwise, the maximum length is 30 characters.

**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 171.

**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 or the SAS Factory Miner HPFOREST or HPSVM component can be used.

**Interactions**

If you use the default format catalog that is created by the SAS Enterprise Miner Score Code Export node or the SAS Factory Miner HPFOREST or HPSVM component, 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 *Base SAS Procedures Guide.*
Note The analytic store file includes format catalog information. However, you can specify a different format catalog by using the FMTCAT= argument.

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 156.

Tip This argument is useful when testing your scoring models.

See “Special Characters in Directory Names” on page 20

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 for Publishing Formats and Scoring Models” in SAS In-Database Products: Administrator's Guide.
Chapter 14

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 15

Using the SAS In-Database Code Accelerator
Chapter 15
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.

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 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 189.

Note: The SAS In-Database Code Accelerator is available only for Greenplum, Hadoop, and Teradata.

Requirements for Using the SAS In-Database Code Accelerator

To use the SAS In-Database Code Accelerator, the following requirements must be met. Otherwise, the thread program is run in multiple threads on the client machine.

- The following products must be licensed at your site:
  - Base SAS
  - SAS In-Database Code Accelerator
    The SAS In-Database Code Accelerator for Hadoop is available by licensing SAS Data Loader. The SAS In-Database Code Accelerator for Teradata is available by licensing SAS In-Database Technologies for Teradata.
  - 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 tables 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.

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 or MapReduce 2.

*Note:* The SAS In-Database Code Accelerator for Hadoop supports only Cloudera 5.0, Hortonworks 2.0, IBM BigInsights 3.0, Pivotal HD 2.1, and MapR 4.0 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
- HDFS: SPD Engine**

*In the February 2015 release for SAS 9.4, the SAS In-Database Code Accelerator for Hadoop supports these file types.

**In the July 2015 release for SAS 9.4, the SAS In-Database Code Accelerator for Hadoop supports this file type.

*Note:* Only SPD Engine data sets whose architectures match the architecture of the Hadoop cluster (that is, 64-bit Solaris or Linux) run inside the database. Otherwise, the data and thread program run on the client machine.

**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. SASHDAT file types are not supported.

**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 15.1 Summary of HCatalog File I/O

<table>
<thead>
<tr>
<th>File Type</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimited</td>
<td>HDFS direct-read</td>
<td>HDFS direct-read</td>
</tr>
<tr>
<td></td>
<td>HCatalog if partitioned, skewed, or escaped</td>
<td>HCatalog if partitioned, skewed, or escaped</td>
</tr>
<tr>
<td>RCFile</td>
<td>HCatalog</td>
<td>HCatalog</td>
</tr>
<tr>
<td>ORC</td>
<td>HCatalog</td>
<td>HCatalog</td>
</tr>
<tr>
<td>Parquet</td>
<td>HCatalog*</td>
<td>CREATE TABLE AS SELECT**</td>
</tr>
<tr>
<td>sequence</td>
<td>HDMD</td>
<td>HCatalog</td>
</tr>
<tr>
<td></td>
<td>HCatalog if partitioned or skewed</td>
<td></td>
</tr>
<tr>
<td>Avro</td>
<td>HCatalog*</td>
<td>CREATE TABLE AS SELECT***</td>
</tr>
</tbody>
</table>

* 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](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](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 or later.
- 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, through HCatalog, 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. In addition, you need to add the following HCatalog JAR files to the `SAS_HADOOP_JAR_PATH` environment variable:

  - `webhcat-java-client*.jar`
  - `hbase-storage-handler*.jar`
  - `hcatalog-server-extensions*.jar`
  - `hcatalog-core*.jar`
  - `hcatalog-pig-adapter*.jar`

- On the Hadoop cluster, the SAS Embedded Process for Hadoop install script automatically adds HCatalog JAR files to its configuration file. The HCatalog JAR files are added to the Embedded Process Map Reduce job class path during job submission.

  For information about installing and configuring the SAS Embedded Process for Hadoop, see *SAS In-Database Products: Administrator’s Guide*.

- You must include the HCatalog JAR files in your `SAS_HADOOP_JAR_PATH` environment variable.

  For more information about the `SAS_HADOOP_JAR_PATH` environment variable, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

- The `hive-site.xml` file must be in your `SAS_HADOOP_CONFIG_PATH`.

  For more information about the `SAS_HADOOP_CONFIG_PATH` environment variable, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

- If you are using IBM BigInsights v3.0 or Pivotal HD 2.x, do not use quotation marks around table names. The version of Hive that is used by IBM BigInsights v3.0 and Pivotal HD 2.x does not support quoted identifiers.

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 189.
Using the MapReduce Job Logs to View DS2 Error Messages

The SAS In-Database Code Accelerator provides an HTTP job location when a job fails. In the July 2015 release for SAS 9.4, when the MSGLEVEL=I option is set and a job fails, a link to the HTTP location of the MapReduce logs is also produced. Here is an example.

```
ERROR: Job job_1424277669708_2919 has failed. Please, see job log for details. Job tracking URL :
       http://name.unx.company.com:8088/proxy/application_1424277669708_2919/
```

The HTTP link is to a site that contains the job summary. On the job summary page, you can see the number of failed and successful tasks. If you click on the failed tasks, you see a list of task attempts. A log is assigned to each attempt. Once in the log page, you are able to see the error messages.

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 December 2013 release, 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.
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 “Interleaving” in SAS DS2 Language Reference, and the DS2 procedure in Base SAS Procedures Guide.

Considerations and Limitations

Greenplum, Hadoop, and Teradata

- 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.
- 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 an SQLSTMT package
  - using an initialized hash package
  - using an HTTP package
In the July 2015 release for SAS 9.4, multi-table SET statements and a SET statement with embedded SQL code are allowed. Here is an example.

```
set dlib.invoice dlib.paysched;
```

**Note:** The librefs in the SET statement must be the same (for example, they must have the same schema, permissions, or use the same catalog). Otherwise, the data and thread programs are run on the client.

**Note:** Only one SET statement is allowed. 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.

**Note:** Using multi-table SET statements, embedded SQL, or the MERGE statement requires Hive.

**Note:** Use of the multi-table SET statement or a SET statement with embedded SQL with Hadoop requires Hive .13 or later.

- Using an unrecognized catalog in the SET statement causes the thread program to run on the client.
- In the July 2015 release for SAS 9.4, MERGE statements are allowed when using the SAS In-Database Code Accelerator.

**Note:** Tables with the SPD Engine or HDMD format do not support the MERGE statement.

**Note:** Use of the MERGE statement with Hadoop requires Hive .13 or later.

- Only one SET FROM statement is allowed in the data program. Otherwise, an error occurs.
- Some data sources choose their one preferred order for columns in the output table from DS2. For example, on Hive, the BYPARTITION columns are always moved to the end of the table. This is common as various data sources try to optimize their performance.

The order of declaration in a DS2 program might not be used as the order of columns in the data source. For example, if you use `keep K1 - K4;`, you might not get the columns as you expect or you might get an error because `K1` appears after `K4` in the CREATE TABLE statement.

### Greenplum

- Only the thread program runs inside the database.

### Hadoop

- Both the data and thread program can run inside the database if the output table from the data program resides in Hadoop.

**Note:** If the data program contains any data transformations beyond outputting the data, the data program is not run inside the database.

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.
- 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:
  • SCHEMA
  • HDFS_METADIR
  • HDFS_TEMPDIR
  • HDFS_PERMDIR

If the output table from the data program does not reside in Hadoop, only the thread program is run inside the cluster.

• 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)

• 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.

• When working with delimited files, data is textualized using the closest fitting format. The data is stored in a textualized manner. Therefore, some discrepancies might occur, and the transformation causes alteration of precision. For example, Hadoop would create an HDFS text representation of a floating point DOUBLE value. After retrieving the value, the resulting DOUBLE value could be slightly different from the starting value.

• The BYPARTITION=NO option in the PROC DS2 statement specifies that the input data is not re-partitioned even if there is a BY statement and enables two-stage aggregation. When using the SAS In-Database Code Accelerator for Hadoop, this option setting is ignored and the BYPARTITION=YES is used. Alternate thread stage aggregation techniques such as a hash object should be used instead of BYPARTITION=NO.

• Hadoop reserved keywords cannot be used for table names. Quoting table names that are Hadoop reserved keywords does not work.

**Teradata**

• Both the data and thread program run inside the database if the output table from the data program resides in Teradata.

_Considerations and Limitations_ 191

*Note:* If the data program contains any data transformations beyond outputting the data, the data program is not run inside the database.

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 in the same Teradata database.
• When the connection strings are compared, they must be identical in value and case except for these values:
  • CATALOG
  • SCHEMA

If the output table from the data program does not reside in Teradata, only the thread program is run inside the database.

---

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.

```sas
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;
The following output table is produced.

Output 15.1  Result Table for Example 2

<table>
<thead>
<tr>
<th>x</th>
<th>z</th>
<th>w</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>YES</td>
<td>NO</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>YES</td>
<td>NO</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>YES</td>
<td>NO</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>YES</td>
<td>NO</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>YES</td>
<td>NO</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>YES</td>
<td>NO</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>YES</td>
<td>NO</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>YES</td>
<td>NO</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>YES</td>
<td>NO</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>YES</td>
<td>NO</td>
<td>10</td>
</tr>
</tbody>
</table>

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.
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` are on the same data partition and transposed by one thread.

```sas
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 &nobs;
    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 15.3  Result Table for Example 4 (Partial Output)

<table>
<thead>
<tr>
<th>amount1</th>
<th>amount2</th>
<th>amount3</th>
<th>amount4</th>
<th>amount5</th>
<th>amount6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3942.11</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>7112.52</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>8531.73</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4189.85</td>
</tr>
<tr>
<td>628.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>5436.21</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Part 4

In-Database DATA Step Processing

Chapter 16
DATA Step Processing in Hadoop

203
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 204 and “Restrictions in DATA Step Processing” on page 204.

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:

```sas
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 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
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.
- If you create a table in Hadoop using the SAS/ACCESS HADOOP LIBNAME engine and the SAS Embedded Process and then drop an input variable with a DROP statement, missing values are assigned to any variable that is created from that input variable. Any variable that is dropped from the output table is also dropped from the input table. Dropping an input variable with the DROP statement is the same as using the DROP= data set option in the SET statement. The workaround is to create a separate DATA step to drop NEWX from the output table.

In this example, the variable \( x \) is dropped in the second data program. The new variable that is created in the data program, \( \text{new} x \), is created with a missing value.

```sas
/* This code cannot be run as stand-alone */
/* as options specific to your site */
/* are required on OPTION and LIBNAME statements. */

/* options for Hadoop jars and configuration files */

option set=SAS_HADOOP_CONFIG_PATH="/saswork/hadoop_files/cdh5config";
option set=SAS_HADOOP_JAR_PATH="/saswork/hadoop_files/cdh5jars";

/* This options enables the DATA step to run, with */
/* limitations, inside Hadoop using */
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 17
Deploying and Using SAS Formats inside the Database .................. 209

Chapter 18
Deploying and Using SAS Formats in Aster ............................... 217

Chapter 19
Deploying and Using SAS Formats in DB2 under UNIX .......... 231

Chapter 20
Deploying and Using SAS Formats in Greenplum ..................... 243

Chapter 21
Deploying and Using SAS Formats in Netezza ......................... 253

Chapter 22
Deploying and Using SAS Formats in Teradata ....................... 265
Chapter 17
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:

```
```
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.
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.
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>
<thead>
<tr>
<th>Character</th>
<th>How to Represent</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank*</td>
<td>%str()</td>
</tr>
<tr>
<td>**</td>
<td>%str(*)</td>
</tr>
<tr>
<td>;</td>
<td>%str(;)</td>
</tr>
<tr>
<td>, (comma)</td>
<td>%str(,)</td>
</tr>
</tbody>
</table>
**Special Characters in Directory Names**

<table>
<thead>
<tr>
<th>Character</th>
<th>How to Represent</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>%str(=)</td>
</tr>
<tr>
<td>+</td>
<td>%str(+)</td>
</tr>
<tr>
<td>-</td>
<td>%str(–)</td>
</tr>
<tr>
<td>&gt;</td>
<td>%str(&gt;)</td>
</tr>
<tr>
<td>&lt;</td>
<td>%str(&lt;)</td>
</tr>
<tr>
<td>^</td>
<td>%str(^)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>%str(&amp;)</td>
</tr>
<tr>
<td>#</td>
<td>%str(#)</td>
</tr>
<tr>
<td>/</td>
<td>%str(⁄)</td>
</tr>
<tr>
<td>~</td>
<td>%str(~)</td>
</tr>
<tr>
<td>%</td>
<td>%str(%)</td>
</tr>
<tr>
<td>'</td>
<td>%str(‘)</td>
</tr>
<tr>
<td>&quot;</td>
<td>%str(&quot;&quot; )</td>
</tr>
<tr>
<td>(</td>
<td>%str(())</td>
</tr>
<tr>
<td>)</td>
<td>%str(%)</td>
</tr>
<tr>
<td>¬</td>
<td>%str(¬)</td>
</tr>
</tbody>
</table>

* 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 17.2 Examples of Special Characters in Directory Names**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Code Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:\temp\Sales(part1)</td>
<td>c:\temp\Sales%str(()%part1%str(()%</td>
</tr>
<tr>
<td>c:\temp\Drug &quot;trial&quot; X</td>
<td>c:\temp\Drug %str(%)trial%str(%) X</td>
</tr>
<tr>
<td>c:\temp\Disc's 50% Y</td>
<td>c:\temp\Disc%str(%)s 50%str(%%) Y</td>
</tr>
</tbody>
</table>
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:

  ```sas
  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 297 and the “SQLREDUCEPUT= System Option” on page 298.

• 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:

```sas
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 18
Deploying and Using SAS Formats in Aster

User-Defined Formats in the Aster Database

Introduction to User-Defined Formats in Aster
Aster Limitations and Restrictions When Using the FMTCAT= Option

Publishing SAS Formats in Aster
Overview of the Publishing Process
Running the %INDAC_PUBLISH_FORMATS Macro
INDCONN Macro Variable
%INDAC_PUBLISH_FORMATS Macro Syntax
Format Publishing Macro Example

Aster Format Files
Overview of Aster Format Files
Example of a Format File

Using the SAS_PUT( ) Function in the Aster Database
Overview of the SAS_PUT( ) Function
Implicit Use of the SAS_PUT( ) Function
Explicit Use of the SAS_PUT( ) Function

Aster Permissions

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 218. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in the Aster Database” on page 225.
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 225.

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 219.

2. Run the %INDAC_PUBLISH_FORMATS macro.

   For more information, see “%INDAC_PUBLISH_FORMATS Macro Syntax” on page 220.

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:

\[
\text{USER=} \text{username} \quad \text{PASSWORD=} \text{password} \quad \text{DSN=} \text{dsnname} <\text{SCHEMA=} \text{schemaname}> \\
\text{USER=} \text{username} \quad \text{PASSWORD=} \text{password} \quad \text{DATABASE=} \text{databasename} \\
\text{SERVER=} \text{servername} <\text{SCHEMA=} \text{schemaname}>
\]

**USER=**

specifies the Aster user name (also called the user ID) that is used to connect to the database.

**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=**

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=**

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=**

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=**

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 218.

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 214
**FMTLIST=** *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 212
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 225.

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 223.

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
<?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>
    </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>
  </TABLE>
</LIBRARY>
```
<table>
<thead>
<tr>
<th>FMTNAME</th>
<th>START</th>
<th>END</th>
<th>LABEL</th>
<th>MIN</th>
<th>MAX</th>
<th>DEFAULT</th>
<th>LENGTH</th>
<th>FUZZ</th>
<th>PREFIX</th>
<th>MULT</th>
<th>FILL</th>
<th>NOEDIT</th>
<th>TYPE</th>
<th>SEXCL</th>
<th>EEXCL</th>
<th>HLO</th>
<th>DECSEP</th>
<th>DIG3SEP</th>
<th>DATATYPE</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVFMT</td>
<td>1</td>
<td>1</td>
<td>New England</td>
<td>1</td>
<td>40</td>
<td>15</td>
<td>15</td>
<td>1E-12</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>Middle Atlantic</td>
<td>1</td>
<td>40</td>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>PRICE_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8.00</td>
</tr>
<tr>
<td>$10.00</td>
</tr>
<tr>
<td>$12.00</td>
</tr>
<tr>
<td>$13.59</td>
</tr>
<tr>
<td>$13.99</td>
</tr>
<tr>
<td>$14.00</td>
</tr>
<tr>
<td>$27.98</td>
</tr>
<tr>
<td>$48.99</td>
</tr>
<tr>
<td>$54.00</td>
</tr>
</tbody>
</table>

quit;

Considerations with Implicit Use of SAS_PUT() 227

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.

```sql
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 226 and explicitly uses the SAS_PUT( ) function call.

```sql
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.

```sql
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.

```sql
select distinct
```
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 19
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 231. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in the DB2 Database” on page 238.

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
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:
   ```sas
   %let indconn = server=yourserver user=youruserid password=yourpwd
database=yourdb schema=yourschema serveruserid=yourserveruserid;
   ``
   For more information, see the “INDCONN Macro Variable” on page 233.

2. Run the `%INDB2_PUBLISH_FORMATS` macro.
   For more information, see “%INDB2_PUBLISH_FORMATS Macro Syntax” on page 234.

   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 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.
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.

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.

The INDCCONN 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

ARGS

DATABASE=database-name
<FMTCAT=format-catalog-filename>
<FORMATTABLE=format-table-name>
<ACTION=CREATE | REPLACE | DROP>
<MULTIPLE=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 INDCCONN macro variable. For more information, see “Running the %INDB2_PUBLISH_FORMATS Macro” on page 232.

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 214

**FMTTABLE=**<br>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 19.1 Format Table Columns**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTNAME</td>
<td>specifies the name of the format.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>specifies the origin of the format. SOURCE can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>SAS supplied by SAS</td>
</tr>
<tr>
<td></td>
<td>PROCFMT User–defined with PROC FORMAT</td>
</tr>
</tbody>
</table>

**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.

**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 212

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

```sas
%let indconn = server=db2base user=user1 password=open1
database=mydb schema=myschema;
%indb2_publishFormats(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 238.
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.

```sas
%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
      PUT(SALES, Dollar8.2)AS SALES_C from dblib.SHOES;
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
```
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.

  ```sql
  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 238 and explicitly uses the SAS_PUT( ) function call.

```sql
%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.

```
quit;
1733
1734 proc sql noerrorstop;
1735 title 'Test SAS_PUT using Explicit Passthru';
1736 connect to db2 (user=db2 password=XXXXXXXXXXXX database=indb);
1737 DB2: AUTOCOMMIT is YES for connection 4 1870 1309265953 ducon 0 SQL
1738 select * from connection to db2
1739 (select distinct (sas_put("SALES","DOLLAR12.2")) as "SALES_C" from SHOES);
1740 DB2_366: Prepared: on connection 4 1872 1309265953 du_prep 0 SQL
1741 select distinct (sas_put("SALES","DOLLAR12.2")) as "SALES_C" from SHOES 1873
1742 1309265953 du_prep 0 SQL
1743 DB2: COMMIT performed on connection 4. 1874 1309265953 du_comm 0 SQL
1744 DB2_367: Executed: on connection 4 1876 1309265953 du_exec 0 SQL
1745 Prepared statement DB2_366 1877 1309265953 du_exec 0 SQL
1746 1309265953 du_exec 0 SQL
1747 disconnect from db2;
1748 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 20
Deploying and Using SAS Formats in Greenplum

User-Defined Formats in the Greenplum Database

Publishing SAS Formats in Greenplum

Overview of the Publishing Process

Running the %INDGP_PUBLISH_FORMATS Macro

INDCONN Macro Variable

%INDGP_PUBLISH_FORMATS Macro Syntax

Format Publishing Macro Example

Using the SAS_PUT( ) Function in Greenplum

Implicit Use of the SAS_PUT( ) Function

Explicit Use of the SAS_PUT( ) Function

Greenplum Permissions

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 243. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in Greenplum” on page 248.

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 245.

2. Run the %INDGP_PUBLISH_FORMATS macro.

   For more information, see “%INDGP_PUBLISH_FORMATS Macro Syntax” on page 246.

   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=userid
```

```
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=userid**

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 244.

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.

Interaction  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.

Interaction  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 20.1 Format Table Columns**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTNAME</td>
<td>specifies the name of the format.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>specifies the origin of the format. SOURCE can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>SAS supplied by SAS</td>
</tr>
<tr>
<td></td>
<td>PROCFMT User-defined with PROC FORMAT</td>
</tr>
</tbody>
</table>

**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 212
Format Publishing Macro Example

\[
\%let \text{indconn} = \text{user=\text{user1} password=\text{xxxx dsn=\text{dsn34 schema=block}};}
\%\text{indgp_publish_formats(fmtcat=work.formats)};
\]

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 %\text{INDGP_PUBLISH_FORMATS} macro also produces a text file of Greenplum CREATE FUNCTION commands that are similar to these:

\[
\text{CREATE OR REPLACE FUNCTION dbitest.homeeq_5_em_classification}
\text{(}
\text{float8, float8, float8, float8, float8, varchar(32), float8, float8, varchar(32), float8, float8, float8}
\text{)}
\text{RETURNS varchar(33) AS '/usr/local/greenplum-db-3.3.4.0/lib/postgresql/SAS/sample_dbitest_homeeq_5.so', 'homeeq_5_em_classification'}
\]

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 248.

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 \text{id} values with the ANIMAL 20.0 format and processes the SELECT DISTINCT clause using the formatted values.

\[
\text{/* implicit pass-thru query */ }
\text{options sqlgeneration=dbms sqlreduceput=none;}
\]
options sastrace=',',,d' sastraceloc=saslog
        sql_ip_trace=(note,source) msglevel=i;

proc sql noerrorstop reducetputnone details="reduce_put_bench$";
create table fmt_ipout as
    select distinct id, put(a,ANIMAL.) len=50 as fmtresult
    from dblib.sample ;
quit;

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 248 and explicitly uses the SAS_PUT( ) function call.

```sql
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.

```sql
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*. 
User-Defined Formats in the Netezza Data Warehouse

Introduction to User-Defined Formats in Netezza ............................................ 253
Netezza Considerations and Limitations When Using the FMTCAT= Options . . . 254

Publishing SAS Formats in Netezza ................................................................. 254
Overview of the Publishing Process ................................................................. 254
Running the %INDNZ_PUBLISH_FORMATS Macro ......................................... 254
INDCONN Macro Variable .................................................................................. 255
%INDNZ_PUBLISH_FORMATS Macro Syntax ..................................................... 255
Modes of Operation ............................................................................................ 259
Format Publishing Macro Example .................................................................... 259

Using the SAS_PUT( ) Function in the Netezza Data Warehouse .................... 260
Implicit Use of the SAS_PUT( ) Function ......................................................... 260
Explicit Use of the SAS_PUT( ) Function ........................................................ 262

Netezza Permissions .......................................................................................... 263

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 254. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in the Netezza Data Warehouse ” on page 260.
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> <schema=myschema>;

   For more information, see the “INDCONN Macro Variable” on page 255.
2. Run the `%INDNZ_PUBLISH_FORMATS` macro.

For more information, see “%INDNZ_PUBLISH_FORMATS Macro Syntax” on page 255.

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:

```plaintext
SERVER=server USER=userid PASSWORD=password DATABASE=database
<SCHEMA=schema-name>
```

**Arguments**

**SERVER=server**

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

**USER=userid**

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.

**SCHEMA=<'schema-name'>**

specifies the name of the schema where the formats are published.

**Restriction**

This argument is supported only on Netezza v7.0.3 or later.

**Interaction**

The schema that is specified by the `%INDNZ_PUBLISH_FORMATS` macro’s `DBSCHEMA=` argument takes precedence over the schema that you specify in the INDCONN macro variable. If you do not specify a schema in the `DBSCHEMA=` argument or the INDCONN macro variable, the default schema for the target database is used.

**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>
  <, SCHEMACOMPILE=compile-schema-name>
  <, DBJAZLIB=database-name>
  <, SCHEMAJAZLIB=compile-schema-name>
  <, DBSCHEMA=compile-schema-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 254.

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 SQLMAPUTO= 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 publishing the SAS_COMPILEUDF function, see the SAS In-Database Products: Administrator's Guide.

SCHEMACOMPILE=compile-schema-name
specifies the name of the Netezza schema where the SAS_COMPILEUDF function was published.

Default An empty string

Restriction This argument is supported only on Netezza v7.0.3 or later.

Tip The SAS_COMPILEUDF function is referenced as <DBCOMPILE>..<SCHEMACOMPILE>..SAS_COMPILEUDF.

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.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Restriction</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASLIB</td>
<td>SASLIB</td>
<td>This argument is supported only on TwinFin systems.</td>
<td>For more information about publishing the SAS Formats Library for Netezza, see the <em>SAS In-Database Products: Administrator's Guide</em>.</td>
</tr>
<tr>
<td>SCHEMAJAZLIB=jazlib-schema-name</td>
<td>An empty string</td>
<td>This argument is supported only on Netezza v7.0.3 or later.</td>
<td>For more information about publishing the SAS Formats Library for Netezza, see the <em>SAS In-Database Products: Administrator's Guide</em>.</td>
</tr>
<tr>
<td>DBSCHEMA=schema-name</td>
<td></td>
<td>This argument is supported only on Netezza v7.0.3 or later.</td>
<td></td>
</tr>
</tbody>
</table>
Table 21.1 Format Table Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTNAME</td>
<td>specifies the name of the format.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>specifies the origin of the format. SOURCE can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>SAS</td>
</tr>
<tr>
<td></td>
<td>PROCFMT</td>
</tr>
</tbody>
</table>

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 259

**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 212

---

**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**

```plaintext
%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:

```plaintext
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 260.
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.

```sas
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

<table>
<thead>
<tr>
<th>PRICE_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10.00</td>
</tr>
<tr>
<td>$12.00</td>
</tr>
<tr>
<td>$13.59</td>
</tr>
<tr>
<td>$48.99</td>
</tr>
<tr>
<td>$54.00</td>
</tr>
<tr>
<td>$8.00</td>
</tr>
<tr>
<td>$14.00</td>
</tr>
<tr>
<td>$27.98</td>
</tr>
<tr>
<td>$13.99</td>
</tr>
</tbody>
</table>

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 260 and explicitly uses the SAS_PUT( ) function call.

```
options sqlmapputo=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 sqlmapputo=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

<table>
<thead>
<tr>
<th>PRICE_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$27.98</td>
</tr>
<tr>
<td>$10.00</td>
</tr>
<tr>
<td>$12.00</td>
</tr>
</tbody>
</table>
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.

```sql
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 22
Deploying and Using SAS Formats in Teradata

User-Defined Formats in the Teradata EDW

Introduction to User-Defined Formats in Teradata

Teradata Limitations and Restrictions When Using the FMTCAT= Option

Publishing SAS Formats in Teradata

Overview of the Publishing Process

Running the %INDTD_PUBLISH_FORMATS Macro

INDCONN Macro Variable

%INDTD_PUBLISH_FORMATS Macro Syntax

Modes of Operation

Format Publishing Macro Example

Data Types and the SAS_PUT( ) Function

Using the SAS_PUT( ) Function in the Teradata EDW

Implicit Use of the SAS_PUT( ) Function

Explicit Use of the SAS_PUT( ) Function

Tips When Using the SAS_PUT( ) Function in Teradata

Teradata Permissions

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 266. For more information about the SAS_PUT( ) function, see “Using the SAS_PUT( ) Function in the Teradata EDW” on page 272.

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=xxxxx
database=mydb;

   For more information, see the “INDCONN Macro Variable” on page 267.

2. Run the %INDTD_PUBLISH_FORMATS macro.

   For more information, see “%INDTD_PUBLISH_FORMATS Macro Syntax” on page 267.

   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=userid**
  - 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
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 266.

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 297.

**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>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FMTNAME</strong></td>
<td>specifies the name of the format.</td>
</tr>
<tr>
<td><strong>SOURCE</strong></td>
<td>specifies the origin of the format. SOURCE can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>SAS supplied by SAS.</td>
</tr>
<tr>
<td></td>
<td>PROCFMT User-defined with PROC FORMAT.</td>
</tr>
<tr>
<td><strong>PROTECTED</strong></td>
<td>specifies whether the format is protected. PROTECTED can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>YES Format was created with the MODE= option set to PROTECTED.</td>
</tr>
<tr>
<td></td>
<td>NO Format was created with the MODE= option set to UNPROTECTED.</td>
</tr>
</tbody>
</table>
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 269

**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 212

**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).

```sas
%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:

```sql
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 272.

---

**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 22.2 Teradata Data Types Supported by the SAS_PUT( ) Function

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>BYTEINT</td>
</tr>
<tr>
<td></td>
<td>SMALLINT</td>
</tr>
<tr>
<td></td>
<td>INTEGER</td>
</tr>
<tr>
<td></td>
<td>BIGINT*</td>
</tr>
<tr>
<td></td>
<td>DECIMAL (ANSI NUMERIC)*</td>
</tr>
<tr>
<td></td>
<td>FLOAT (ANSI REAL or DOUBLE PRECISION)</td>
</tr>
<tr>
<td>Date and time</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Character</td>
<td>CHARACTER†</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>LONG VARCHAR</td>
</tr>
</tbody>
</table>

* 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 22.3 Teradata Data Types Not Supported by the SAS_PUT( ) Function

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI date and time</td>
<td>INTERVAL</td>
</tr>
<tr>
<td></td>
<td>TIME WITH TIME ZONE</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>GRAPHIC server character set</td>
<td>GRAPHIC</td>
</tr>
<tr>
<td></td>
<td>VARGRAPHIC</td>
</tr>
<tr>
<td></td>
<td>LONG VARGRAPHIC</td>
</tr>
<tr>
<td>Binary and large object</td>
<td>CLOB</td>
</tr>
<tr>
<td></td>
<td>BYTE</td>
</tr>
<tr>
<td></td>
<td>VARBYTE</td>
</tr>
<tr>
<td></td>
<td>BLOB</td>
</tr>
</tbody>
</table>
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.

```sql
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.

```sql
libname dblib teradata user="sas" password="sas" server="s196208"
```
NOTE: Libref DBLIB was successfully assigned, as follows:

Engine: TERADATA
Physical Name: s196208

/*-- 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
3:42 Thursday, July 11, 2013

<table>
<thead>
<tr>
<th>PRICE_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8.00</td>
</tr>
<tr>
<td>$10.00</td>
</tr>
<tr>
<td>$12.00</td>
</tr>
<tr>
<td>$13.59</td>
</tr>
<tr>
<td>$13.99</td>
</tr>
<tr>
<td>$14.00</td>
</tr>
<tr>
<td>$27.98</td>
</tr>
<tr>
<td>$48.99</td>
</tr>
<tr>
<td>$54.00</td>
</tr>
</tbody>
</table>

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.

```sql
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 272 and explicitly uses the SAS_PUT( ) function call.

```sql
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.

```sql
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

13:42 Thursday, July 25, 2013

<table>
<thead>
<tr>
<th>PRICE_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8.00</td>
</tr>
<tr>
<td>$10.00</td>
</tr>
</tbody>
</table>
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.

```sql
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 270.

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 23

Running SAS Procedures inside the Database ................. 279
## 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 23.1 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='TERADATA DB2 ORACLE NETEZZA ASTER GREENPLUM HADOOP SAPHANA IMPALA HAWQ'.

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 294 or the SQLGENERATION LIBNAME option in SAS/ACCESS for Relational Databases: Reference.

Procedures in Aster, DB2, Greenplum, Hadoop, HAWQ, Impala, Netezza, Oracle, and SAP HANA

The following Base SAS procedures have been enhanced for in-database processing inside Aster, DB2, Greenplum, Hadoop, HAWQ, Impala, 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
• TRANSPOSE (Pre-production)
  Note: PROC TRANSPOSE in-database processing is supported only by Hadoop. Additional licensed products and configuration are required. For more information, see "Running PROC TRANSPOSE inside the Database (Pre-production)" on page 285.

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
- TRANSPOSE (Pre-production)

Note: Additional licensed products and configuration are required. For more information, see “Running PROC TRANSPOSE inside the Database (Preproduction)” on page 285.

- 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 \texttt{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 \texttt{OBS} option and the \texttt{FIRSTOBS} option prevents in-database processing.
- If you specify the \texttt{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 \texttt{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 \texttt{ORDER BY} clause to get consistent row order. Another option is to use the \texttt{SORT} procedure to create an ordinary SAS data set and use the \texttt{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 \texttt{SAS\_PUT( )} function and formats are published to the DBMS. For more information, see the \texttt{SAS Analytics Accelerator for Teradata: Guide}.

These BY statement option settings apply to the in-database procedures:

- The \texttt{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.

---

**Running PROC TRANSPOSE inside the Database (Preproduction)**

**Overview**

In the July 2015 release of SAS 9.4, the TRANSPOSE procedure works within a third-party database through the SAS In-Database Code Accelerator. This in-database processing allows PROC TRANSPOSE to achieve high performance through the distributed computation provided by a massively parallel processing (MPP) database.

The TRANSPOSE procedure performs a dynamic transformation of the data in which the characteristics of the output table, specifically the number and names of the variables as well as their types, are determined from the variable values as well as the characteristics of the input table. This dynamic behavior is achieved by a two-pass
process. During the first pass, rows of the input table are examined to determine the characteristics of the output table. During the second pass, the work of transposing the data is performed. Parallel operation within the MPP database speeds up both the first and second pass. In the first pass, rows are examined in parallel and in place, according to the manner in which they are already partitioned across the nodes of a cluster. In the second pass, rows are repartitioned to form BY groups that are then processed independently and in parallel.

Both the first and second passes of the in-database processing are performed by executing a DS2 program within the SAS Embedded Process, which resides within the nodes of the cluster. The database provides the SAS Embedded Process with the ability to read data from tables and write data to tables. The SAS Embedded Process provides an execution context for the DS2 program. Because the two passes of work are expressed in the DS2 language, columns of the tables are cast to variables that have DS2 data types. Data type support within DS2 is more extensive than that provided by the traditional SAS system. Therefore, the ability of the TRANSPOSE procedure to preserve data types and values of input data with the transposed output data is enhanced when the TRANSPOSE procedure is executed inside the database.

**Requirements**

The TRANSPOSE procedure can perform in-database processing if the following requirements are met:

- The following products are licensed at your site:
  - third maintenance release of SAS 9.4
  - SAS In-Database Code Accelerator (Hadoop or Teradata)
  - SAS/ACCESS Interface to your data provider (Hadoop or Teradata)
- The SAS Embedded Process is properly installed and configured on the cluster.
  For more information, see the *SAS In-Database Products: Administrator's Guide*.
- A client is installed and configured for the data provider.
- The input and output files are in Teradata or Hadoop.
  For in-database processing to occur, the data to be transposed must reside within the database. When the results of the transposition are directed to a table in the database, in-database processing results in very little network traffic between SAS and the data provider. In this case, only the metadata that is required to determine the characteristics of the output table is transferred back to SAS. Large-volume data transfers, which can be very slow, are avoided.
- A valid SAS/ACCESS LIBNAME statement must be provided to communicate with the data provider.
  For more information about the SAS/ACCESS LIBNAME statement, see *SAS/ACCESS for Relational Databases: Reference*.
- The following data provider client and SAS environment variables must be set:
  - For the Teradata client, the COPLIB environment variable.
  - For Hadoop, the SAS_HADOOP_JAR_PATH and SAS_HADOOP_CONFIG_PATH environment variables.
  For more information about the COPLIB environment variable, see your Teradata documentation. For more information about the SAS Hadoop environment variables, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*. 
• The SQLGENERATION system option is set to DBMS and the INDB=YES option is specified in the TRANSPOSE procedure statement.

Currently, in addition to the SQLGENERATION system option being set to allow in-database processing, the INDB=YES option must be specified in the TRANSPOSE statement. In the future, this requirement might be relaxed or deprecated.

For more information about the SQLGENERATION system option, see SAS/ACCESS for Relational Databases: Reference. For more information about the TRANSPOSE statement, see Base SAS Procedures Guide.

• One or more variables is specified in a BY statement.

In an MPP environment, the volume of data processed is expected to be large. To benefit from distributed computation, the data must be sub-divided among nodes in the MPP cluster such that it can be processed in parallel, independently on each node. For the TRANSPOSE procedure, this parallelism is achieved by partitioning the data across the nodes using one or more BY variables. Ideally, the cardinality of the BY variable or combination of BY variables should be large enough so that each BY group is small enough that it can be processed effectively on one node of the cluster. An attempt to process large BY groups can fail if the memory resources required exceed those available on a single node.

For more information about the BY statement, see the TRANSPOSE procedure in Base SAS Procedures Guide

• One or more variables is specified in an ID statement.

Mathematical transposition depends on a defined order of the rows and columns in a matrix. However, in data processing, a defined order of the rows and columns does not necessarily exist. SAS data sets, accessed through the Base engine, maintain the order of observations within the file containing the data so that observations can be read from the data set in an order that is consistent from one read to the next. Therefore, when running PROC TRANSPOSE, you should perform a transposition in which the first observation of an input data set is transposed to the first result variable of an output data set, the second observation to the second result variable, and so on. However, when reading a table or the results of a query through a SAS/ACCESS engine into SAS, no such consistency of order is guaranteed with an MPP data source. Further, non-deterministic ordering is very common in an MPP environment because of the variations introduced by parallel processing.

Because there is no guaranteed consistent ordering of input rows from a database, the TRANSPOSE procedure requires the specification of one or more ID variables for in-database processing. The ID variable (or variables) serve to map an input row to an output column. The assignment of input row to output column is based on the value of the ID variable(s) and is accomplished by forming the name of the output column from the value of the ID variable(s). Simply described, the names are formed by concatenating the formatted values of the ID variables in order of the appearance of the variables in the ID statement and then post-processing the concatenation to ensure that the result is a valid SAS name. The SAS name is validated according to SAS variable naming rules and the value of the VALIDVARNAME option. The unique set of these output column names and representative raw values of the ID variables from which they were formed are discovered by a DS2 program, executed within the SAS EP, and returned to SAS during the first pass of transposition inside the database. Subsequently, these names are used in the creation of the second DS2 program, which performs the actual transposition within the DBMS.

For more information about the ID statement, see the TRANSPOSE procedure in Base SAS Procedures Guide.

• The LET option is specified in the TRANSPOSE procedure statement.
When executing in the traditional fashion in Base SAS, the TRANSPOSE procedure stops with an error when it detects that two input rows within a BY group have ID variable values that form the same output variable name. This behavior is designed to prevent unintentional loss of data during a transposition by disallowing two input rows to be transposed to a single output variable. When both rows form the same name, if not prevented, the values of the second rows overwrite the values of the first row and result in data loss. In the case that the loss of data is acceptable and intentional, the LET option can be specified in the TRANSPOSE procedure statement. Specifying the LET option allows execution to continue, issuing a warning for the duplicate observation, and letting the values of the last observation that formed the duplicate name to overwrite previously transposed values. However, to achieve high performance when executing inside the database, the information gathered and transferred back to SAS regarding the variable names formed from the ID variables is minimized. Therefore, the information is insufficient to detect these duplicate input rows to output variable mappings. Not being detectable, execution does not stop, and error or warning messages are not issued for duplicates. Because of this, the LET option, specified in the TRANSPOSE procedure statement, is required for in-database processing. This requirement is meant to serve as an explicit acknowledgment of the potential for data loss by the user.

Because of the non-deterministic nature of row ordering within a database or file system, if two or more input rows within a BY group map to a single output column, the column values in the transposed output table are also not deterministic and can vary from one run to the next. The values surviving in the output table are associated with the last row mapped from the input table, but the last row mapped can vary due to differences in apparent row ordering. This variation can make comparison of the results of two different transpositions of the same input data difficult. Because data can be lost, even though a user is forced to acknowledge this potential through the use of the LET option, and output results can vary from run to run in the presence of duplicates, the best practice is to ensure that BY and ID variables are chosen such that there is a well-defined, one-to-one mapping between input observations and output variables. Such assurance can be gained through knowledge and inspection of the data being analyzed.

For more information about the LET option, see the TRANSPOSE procedure in Base SAS Procedures Guide.

### Data Type Conversion

In accordance with traditional TRANSPOSE behavior, when no VAR statement is specified, the analysis variables participating in the transposition are those that are numeric in the Base SAS context. To involve other variables in the transposition, explicitly specify them in a VAR statement.

The Base SAS language has two types of variables: numeric and character. Database management and file systems, on the other hand, have many different variable data types. When the TRANSPOSE procedure is run inside the database or file system, the SAS In-Database Code Accelerator is the interface between the two systems. The SAS In-Database Code Accelerator supports more data types than traditional Base SAS but likely fewer than the number of data types that are supported by the database or file system. When running within the SAS Embedded Process, DS2 converts and preserves data types as best it can. When processed inside the database, the TRANSPOSE procedure runs DS2 programs within the SAS Embedded Process. The DS2 programs uses neither the SAS variable types nor the database or file system data types. It uses DS2 data types.
The work of the TRANSPOSE procedure introduces an additional difficulty. The difficulty is the determination of a data type for output variables that can preserve the values of the input variables. A common data type must be determined for all output variables to hold the transposed results of the input variables. If all input analysis variables (those listed in the VAR statement or implied by the lack of one) are of the same data type, then the data type of all result variables in the output is exactly the same as the input data type.

For example, if all input variables to be transposed are of the double-precision, floating-point data type, then all result variables in the output are double-precision, floating-point. However, if the input analysis variables are not of the same type, then a common type capable of preserving the values of all input variables must be determined. For example, similar to traditional TRANSPOSE operation, if there are two input analysis variables, and one is double-precision, floating-point while the other variable is character, then all result output variables must be of the character type. Using DS2 in the SAS Embedded Process complicates this common type determination because DS2 supports more than just the two traditional SAS variable types. For in-database processing, the TRANSPOSE procedure first attempts to preserve the data type of the input analysis variables in the output. If that is not possible, the TRANSPOSE procedure attempts to determine a common representative type that allows the full precision of input values to be preserved in the output. For numeric data types, if no common output type can preserve full numerical precision, then the TRANSPOSE procedure uses the approximate double-precision, floating-point data type to ensure that the magnitude and range of values are preserved.

**In-Database Processing Results**

If in-database processing is successful, you will see the following message appear in the SAS log after submission of the TRANSPOSE procedure.

NOTE: The transposition was performed in the DBMS

Some procedure and data set options can prevent in-database processing. If in-database processing is requested but a transposition cannot be performed inside the database, then traditional execution proceeds in the SAS session.

To determine why in-database processing was not attempted, set the MSGLEVEL=I system option and look for additional WARNING messages in the SAS log. For more information about MSGLEVEL, see “Using the MSGLEVEL Option to Control Messaging” on page 289.

If in-database processing is requested but fails, and the reason for the failure is not clear, then set the SQL_IP_TRACE=ALL system option, re-run the job, and look for additional details in the SAS log.

---

**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 database.
  
  *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 24
System Options That Affect In-Database Processing ............. 293
Chapter 24
System Options That Affect In-Database Processing

Dictionary

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.

Notes: This system option is new starting with the first maintenance release for SAS 9.4.
This option can be restricted by a site administrator. For more information, see
“Restricted Options” in 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 IMPALA HAWQ')
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. The procedure can then 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, HAWQ, Impala, Netezza, Oracle, SAP HANA, Teradata

Note: Support for Impala and HAWQ was added in the third maintenance release for SAS 9.4.

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' engine='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.

Note: As a best practice, run as many calculations in-database as possible.
Processing that is run in-database generally results in better performance.

" "

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' engine='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 24.1 Precedence of Values for SQLGENERATION= LIBNAME and System Options

<table>
<thead>
<tr>
<th>LIBNAME Option</th>
<th>PROC EXCLUDE on System Option?</th>
<th>Engine Specified on System Option</th>
<th>Resulting Value</th>
<th>From (option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not set</td>
<td>yes</td>
<td>NONE</td>
<td>NONE</td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td>EXCLUDEPROC</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td></td>
<td>NONE</td>
<td>NONE</td>
<td>LIBNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBMS</td>
<td></td>
<td>NONE</td>
<td>EXCLUDEPROC</td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not set</td>
<td>no</td>
<td>NONE</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td></td>
<td>NONE</td>
<td>NONE</td>
<td>LIBNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBMS</td>
<td></td>
<td>NONE</td>
<td>DBMS</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

Here is the default that is shipped with the product.

```plaintext
options sqlgeneration='';
proc options option=sqlgeneration
run;
```

SAS procedures generate SQL for in-database processing for all databases except DB2 in this example.

```plaintext
options sqlgeneration='';
options sqlgenerations=(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.

```plaintext
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.

```plaintext
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 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:

```sql
select x from &lib..c where (put(bday, date9.) = put(today(), date9.));
```

Here, the SELECT clause is optimized.

```sql
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 *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 ........................................... 303
Appendix 1

Scoring File Examples

---

Example of a .ds2 Scoring File ............................................................... 303
Example of an Input and Output Variables Scoring File ....................... 323
Example of a User-Defined Formats Scoring File .................................... 330

---

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;
Appendix 1 • Scoring File Examples

```plaintext
{-#_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 ('KSH') 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_1 ') G_SERVICE = 2.0;
when ('REMOTE_') 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;
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;
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;
    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;
          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;
              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;
                    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;
                        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;
                              end;
_1_12 = -1.0;
_1_13 = -1.0;
_1_14 = -1.0;
_DM_FIND = 1.0;
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;
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;
else 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;
else do ;
if _DM12 = '8' then do ;
_1_7 = 1.0;
_DM_FIND = 1.0;
end;
else do ;
if _DM12 = '9' then do ;
Example of a .ds2 Scoring File

```plaintext
_1_8 = 1.0;
_DM_FIND = 1.0;
end;
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;
Example of a .ds2 Scoring File

```plaintext
_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 ;
```

314 Appendix 1 • Scoring File Examples

```plaintext
_else if _DM12 = '3' then do ;
    _3_2 = 1.0;
_else if _DM12 = '4' then do ;
    _3_3 = 1.0;
_else if _DM12 = '11' then do ;
    _3_7 = 1.0;
_else if _DM12 = '12' then do ;
    _3_8 = 1.0;
_else if _DM12 = '10' then do ;
    _3_6 = 1.0;
_else if _DM12 = '8' then do ;
    _3_5 = 1.0;
_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 = .;
Example of a .ds2 Scoring File
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';
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 + (9.47545464503040) * _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 + (-1.4625436452609) * _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 + (-4.7473053364677) * _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 + (0.92431512613497) * TEMP * \_1_8;
LP0 = LP0 + (3.33003316374041) * TEMP * \_1_9;
LP1 = LP1 + (1.2863079547562) * TEMP * \_1_9;
LP2 = LP2 + (4.52005620947533) * TEMP * \_1_9;
LP3 = LP3 + (1.2863079547562) * TEMP * \_1_9;
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 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (-16.2289623285747) * TEMP * \_2_1;
LP1 = LP1 + (3.87844062530087) * TEMP * \_2_1;
LP2 = LP2 + (13.5342255495752) * TEMP * \_2_1;
LP3 = LP3 + (0.1178447033604) * TEMP * \_2_1;
}\_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 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
LP2 = LP2 + (-2.57814751000107) * TEMP * \_2_0;
LP3 = LP3 + (5.18313976012504) * TEMP * \_2_0;
LP0 = LP0 + (4.96178727582277) * TEMP * \_2_0;
LP1 = LP1 + (7.19423934264755) * TEMP * \_2_0;
\_LP2 = \_LP2 + (-2.27711889809608) * \_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 + (0.688846491937) * \_TEMP * \_2_9;
\_LP1 = \_LP1 + (0.26881516596812) * \_TEMP * \_2_9;
\_LP2 = \_LP2 + (3.49366097063402) * \_TEMP * \_2_9;
\_LP3 = \_LP3 + (4.7752198306386) * \_TEMP * \_2_9;
\_LP0 = \_LP0 + (-3.11663563731206) * \_TEMP * \_2_10;
\_LP1 = \_LP1 + (-3.11663563731206) * \_TEMP * \_2_10;
\_LP2 = \_LP2 + (-1.93101189518423) * \_TEMP * \_2_10;
\_LP3 = \_LP3 + (-0.66813727595772) * \_TEMP * \_2_10;
\_LP0 = \_LP0 + (23.3492198306386) * \_TEMP * \_2_11;
\_LP1 = \_LP1 + (15.3692429684277) * \_TEMP * \_2_11;
\_LP2 = \_LP2 + (19.6299281653522) * \_TEMP * \_2_11;
\_LP3 = \_LP3 + (3.7767067535256) * \_TEMP * \_2_11;
\_LP0 = \_LP0 + (0.688846491937) * \_TEMP * \_2_12;
\_LP1 = \_LP1 + (0.26881516596812) * \_TEMP * \_2_12;
\_LP2 = \_LP2 + (3.49366097063402) * \_TEMP * \_2_12;
\_LP3 = \_LP3 + (4.7752198306386) * \_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 + (2.0643367366059) * \_TEMP * \_3_0;
\_LP0 = \_LP0 + (35.8542164366111) * \_TEMP * \_3_1;
\_LP1 = \_LP1 + (-7.0383225133459) * \_TEMP * \_3_1;
\_LP2 = \_LP2 + (-13.592536842049) * \_TEMP * \_3_1;
\_LP3 = \_LP3 + (-11.651202148638) * \_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 + (32.280194616028) * \_TEMP * \_3_3;
\_LP1 = \_LP1 + (5.1093592540826) * \_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 + (1.62190484241675) * \_TEMP * \_3_4;
\_LP3 = \_LP3 + (2.04551962977047) * \_TEMP * \_3_4;
\_LP0 = \_LP0 + (31.6597116255105) * \_TEMP * \_3_5;
\_LP1 = \_LP1 + (2.67824698076013) * \_TEMP * \_3_5;
\_LP2 = \_LP2 + (1.62190484383178) * \_TEMP * \_3_5;
\_LP3 = \_LP3 + (1.19293530007666) * \_TEMP * \_3_5;
\_LP0 = \_LP0 + (31.6597116340262) * \_TEMP * \_3_6;
LP0 = LP0 + (22.81947524222965) * 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 + (-37.114213383863) * TEMP * 5_10;
LP0 = LP0 + (34.2320056651284) * TEMP * 5_11;
LP1 = LP1 + (2.9953971574379) * TEMP * 5_11;
LP2 = LP2 + (-4.63754693643679) * TEMP * 5_11;
LP3 = LP3 + (-37.114213383863) * 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 + (-4.52842188369726) * TEMP * 5_12;
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 + (-6.2222997982126) * TEMP * 6_0;
LP0 = LP0 + (21.8797726373068) * TEMP * 6_1;
LP1 = LP1 + (2.87906958740973) * TEMP * 6_1;
LP2 = LP2 + (1.8366665646474) * 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 + (-0.6240001926188) * TEMP * 6_2;
LP3 = LP3 + (0.53569098310408) * TEMP * 6_2;
TEMP = 1.0;
LP0 = LP0 + (-3.44927846138227) * TEMP * 7_0;
LP1 = LP1 + (-6.37652016665453) * TEMP * 7_0;
LP2 = LP2 + (-4.2509493215537) * TEMP * 7_0;
LP3 = LP3 + (-4.5168563932432) * TEMP * 7_0;
LP0 = LP0 + (-6.4340808433648) * TEMP * 7_1;
LP1 = LP1 + (-0.80236520705753) * TEMP * 7_1;
LP2 = LP2 + (-1.292246327966) * TEMP * 7_1;
LP3 = LP3 + (-6.3228249961119) * 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 = -18.590997968937 + LP2;
if _LPMAX < _LP2 then _LPMAX = _LP2;
LP3 = -6.00322742797283 + LP3;
if _LPMAX < _LP3 then _LPMAX = _LP3;
LP0 = EXP(_LP0 - _LPMAX);
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
<Score>
  <Producer>
    <Name> SAS Enterprise Miner </Name>
  </Producer>
  <File>example_score_io.xml</File>
  <Data>
    <Table>
      <Column>LP1 = EXP(_LP1 - _LPMAX);</Column>
      <Column>_LP2 = EXP(_LP2 - _LPMAX);</Column>
      <Column>_LP3 = EXP(_LP3 - _LPMAX);</Column>
      <Column>_LPMAX = EXP(-_LPMAX);</Column>
      <Column>_P4 = 1.0 / (_LPMAX + _LP0 + _LP1 + _LP2 + _LP3);</Column>
      <Column>_P0 = _LP0 * _P4;</Column>
      <Column>_P1 = _LP1 * _P4;</Column>
      <Column>_P2 = _LP2 * _P4;</Column>
      <Column>_P3 = _LP3 * _P4;</Column>
      <Column>_P4 = _LPMAX * _P4;</Column>
      <Column>REGDR1: P_ATTACKU2R = _P0;</Column>
      <Column>_MAXP = _P0;</Column>
      <Column>_IY = 1.0;</Column>
      <Column>P_ATTACKR2L = _P1;</Column>
      <Column>if (_P1 - _MAXP > 1E-8) then do;</Column>
      <Column>_MAXP = _P1;</Column>
      <Column>_IY = 2.0;</Column>
      <Column>end;</Column>
      <Column>P_ATTACKPROBE = _P2;</Column>
      <Column>if (_P2 - _MAXP > 1E-8) then do;</Column>
      <Column>_MAXP = _P2;</Column>
      <Column>_IY = 3.0;</Column>
      <Column>end;</Column>
      <Column>P_ATTACKNORMAL = _P3;</Column>
      <Column>if (_P3 - _MAXP > 1E-8) then do;</Column>
      <Column>_MAXP = _P3;</Column>
      <Column>_IY = 4.0;</Column>
      <Column>end;</Column>
      <Column>P_ATTACKDOS = _P4;</Column>
      <Column>if (_P4 - _MAXP > 1E-8) then do;</Column>
      <Column>_MAXP = _P4;</Column>
      <Column>_IY = 5.0;</Column>
      <Column>end;</Column>
      <Column>I_ATTACK = REGDRF[_IY];</Column>
      <Column>U_ATTACK = REGDRU[_IY];</Column>
      <Column>EM_EVENTPROBABILITY = P_ATTACKU2R;</Column>
      <Column>EM_PROBABILITY = MAX(P_ATTACKU2R, P_ATTACKR2L, P_ATTACKPROBE, P_ATTACKNORMAL, P_ATTACKDOS);</Column>
      <Column>EM_CLASSIFICATION = I_ATTACK;</Column>
      <Column>_return: ;</Column>
      <Column>end;</Column>
    </Table>
  </Data>
</Score>
```
<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>
</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>
Example of an Input and Output Variables Scoring File

<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>
Appendix 1 • Scoring File Examples

```xml
<Description/>
<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>
        </DataMap>
      </Array>
    </Parameter>
  </ParameterList>
</Function>
```
Example of an Input and Output Variables Scoring File

<DataMap>
  <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 length="18">
  <Type>
    Parm
  </Type>
  <DataMap>
    <Element index="0">
      <Value>
        <Origin> AOV16_COUNT </Origin>
        <Type> double </Type>
      </Value>
    </Element>
  </DataMap>
</Array>
</Parameter>
Appendix 1 • Scoring File Examples
Example of an Input and Output Variables Scoring File

```
<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>
```
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
<?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>
    </TABLE-HEADER>
    <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>
  </COLUMN>
</TABLE>
</LIBRARY>
```
Example of a User-Defined Formats Scoring File

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Label</th>
<th>Type</th>
<th>Data Type</th>
<th>Length</th>
<th>Offset</th>
<th>Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>Ending value for format</td>
<td>character</td>
<td>string</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>LABEL</td>
<td>Format value label</td>
<td>character</td>
<td>string</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>Minimum length</td>
<td>numeric</td>
<td>double</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td>Maximum length</td>
<td>numeric</td>
<td>double</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Default length</td>
<td>numeric</td>
<td>double</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td>Format length</td>
<td>numeric</td>
<td>double</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FUZZ</td>
<td>Fuzz value</td>
<td>numeric</td>
<td>double</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Appendix 1 • Scoring File Examples

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Label</th>
<th>Type</th>
<th>Data Type</th>
<th>Length</th>
<th>Offset</th>
<th>Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX</td>
<td>Prefix characters</td>
<td>character</td>
<td>string</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MULT</td>
<td>Multiplier</td>
<td>numeric</td>
<td>double</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>FILL</td>
<td>Fill character</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NOEDIT</td>
<td>Is picture string noedit?</td>
<td>numeric</td>
<td>double</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>Type of format</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SEXCL</td>
<td>Start exclusion</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EEXCL</td>
<td>End exclusion</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

```xml
<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>
```
Example of a User-Defined Formats Scoring File

```
<ROW>
  <FMTNAME missing=" " />
  <START missing=" " />
  <END missing=" " />
  <LABEL missing=" " />
  <MIN missing=" " />
  <MAX missing=" " />
  <DEFAULT missing=" " />
  <LENGTH missing=" " />
  <FUZZ missing=" " />
  <PREFIX missing=" " />
</ROW>
```

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Label</th>
<th>Type</th>
<th>Data Type</th>
<th>Length</th>
<th>Offset</th>
<th>Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLO</td>
<td>Additional information</td>
<td>character</td>
<td>string</td>
<td>11</td>
<td>11</td>
<td>/</td>
</tr>
<tr>
<td>DECSEP</td>
<td>Decimal separator</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td>DIG3SEP</td>
<td>Three-digit separator</td>
<td>character</td>
<td>string</td>
<td>1</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Date/time/datetime?</td>
<td>character</td>
<td>string</td>
<td>8</td>
<td>8</td>
<td>/</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>Language for date strings</td>
<td>character</td>
<td>string</td>
<td>8</td>
<td>8</td>
<td>/</td>
</tr>
</tbody>
</table>
### Appendix 1 • Scoring File Examples

<table>
<thead>
<tr>
<th>DELTA-RECORD</th>
<th>FMTNAME</th>
<th>START</th>
<th>END</th>
<th>LABEL</th>
<th>MIN</th>
<th>MAX</th>
<th>DEFAULT</th>
<th>LENGTH</th>
<th>FUZZ</th>
<th>MULT</th>
<th>FILL</th>
<th>NOEDIT</th>
<th>TYPE</th>
<th>SEXCL</th>
<th>EEXCL</th>
<th>HLO</th>
<th>DECSEP</th>
<th>DIG3SEP</th>
<th>DATATYPE</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>key=&quot;ABC&quot;</td>
<td>ABC</td>
<td>1</td>
<td>1</td>
<td>yes</td>
<td>1</td>
<td>40</td>
<td>3</td>
<td>3</td>
<td>1E-12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>key=&quot;YESNO&quot;</td>
<td>YESNO</td>
<td>0</td>
<td>0</td>
<td>NO</td>
<td>1</td>
<td>40</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example of a User-Defined Formats Scoring File

```
<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>
  <EXCL>N</EXCL>
  <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 Data Loader for Hadoop: User's Guide
- SAS Enterprise Miner: High-Performance Procedures
- SAS In-Database Products: Administrator's Guide
- SAS/STAT User’s Guide
- SAS Scalable Performance Data Server: User’s Guide

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

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books
<table>
<thead>
<tr>
<th>Special Characters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%INDAC_PUBLISH_FORMATS macro</td>
<td>example 222, running 218, syntax 220</td>
</tr>
<tr>
<td>%INDAC_PUBLISH_MODEL macro</td>
<td>example 36, running 32, syntax 34</td>
</tr>
<tr>
<td>%INDB2_CREATE_MODELTABLE macro</td>
<td>running 48, syntax 49</td>
</tr>
<tr>
<td>%INDB2_PUBLISH_FORMATS macro</td>
<td>example 237, modes of operation 237, running 232, syntax 234</td>
</tr>
<tr>
<td>%INDB2_PUBLISH_MODEL macro</td>
<td>modes of operation 58, running 52, syntax 54</td>
</tr>
<tr>
<td>%INDGP_CREATE_MODELTABLE macro</td>
<td>running 68, syntax 69</td>
</tr>
<tr>
<td>%INDGP_PUBLISH_FORMATS macro</td>
<td>example 248, running 244, syntax 246</td>
</tr>
<tr>
<td>%INDGP_PUBLISH_MODEL macro</td>
<td>running 73, syntax 75</td>
</tr>
<tr>
<td>%INDHD_PUBLISH_FORMATS macro</td>
<td>syntax 82</td>
</tr>
<tr>
<td>%INDHD_RUN_MODEL macro</td>
<td>syntax 85</td>
</tr>
<tr>
<td>%INDHN_CREATE_MODELTABLE macro</td>
<td>running 133</td>
</tr>
<tr>
<td>%INDHN_PUBLISH_FORMATS macro</td>
<td>running 136, syntax 136</td>
</tr>
<tr>
<td>%INDHN_RUN_MODEL macro</td>
<td>syntax 139</td>
</tr>
<tr>
<td>%INDNZ_CREATE_MODELTABLE macro</td>
<td>running 103, syntax 104</td>
</tr>
<tr>
<td>%INDNZ_PUBLISH_FORMATS macro</td>
<td>example 259, modes of operation 259, running 254, syntax 255</td>
</tr>
<tr>
<td>%INDNZ_PUBLISH_MODEL macro</td>
<td>modes of operation 115, running 110, syntax 111</td>
</tr>
<tr>
<td>%INDOR_CREATE_MODELTABLE macro</td>
<td>running 119, syntax 120</td>
</tr>
<tr>
<td>%INDOR_PUBLISH_FORMATS macro</td>
<td>running 121, syntax 122</td>
</tr>
<tr>
<td>%INDOR_PUBLISH_MODEL macro</td>
<td>syntax 148</td>
</tr>
<tr>
<td>%INDSP_PUBLISH_FORMATS macro</td>
<td>running 161, syntax 162</td>
</tr>
<tr>
<td>%INDSP_RUN_MODEL macro</td>
<td>syntax 177</td>
</tr>
<tr>
<td>%INDTD_CREATE_MODELTABLE macro</td>
<td>running 171, syntax 173</td>
</tr>
<tr>
<td>%INDTD_PUBLISH_FORMATS macro</td>
<td>example 270, modes of operation 269, running 266, syntax 267</td>
</tr>
<tr>
<td>%INDTD_PUBLISH_MODEL macro</td>
<td>modes of operation 177, running 171, syntax 173</td>
</tr>
</tbody>
</table>

A
- analytic store scoring 29
- Hadoop 79
- HPFOREST component 30
- HPSVM component 30
SAP HANA 129
Teradata 155
ANALYZE_TABLE function
  overview 50
  syntax 50
  using 50
Aster
  deployed components for in-database processing 5
  format files 222
  in-database procedures 281
  permissions 37, 229
  publishing SAS formats 218
  SAS Embedded Process 5
  SAS System libraries 6
  SAS_PUT() function 218
  SAS_SCORE() function 39
  Scoring Accelerator 31
  user-defined formats 217

B
BY-group processing
  in-database procedures and 283
  SAS In-Database Code Accelerator 189

C
case sensitivity 43, 63, 99, 157

D
data mining models 23
  data set options
    in-database procedures and 284
  DATA step 293
  data types
    SAS_PUT() function (Teradata) 270
  DB2
    ANALYZE_TABLE function 50
    creating a model table 48
    deployed components for in-database processing 6
    in-database procedures 281
    permissions 59, 240
    publishing SAS formats 231
    SampleSQL.txt file 44
    SAS Embedded Process 6, 41
    SAS formats library 6
    Scoring Accelerator 41
    user-defined formats 231
    using scoring functions to run scoring models 42
    using the SAS Embedded Process to run scoring models 47
  DBCREATE_TABLE_OPTS table option 188
directory names
  special characters in 20, 212
DS2
  SAS In-Database Code Accelerator 183
DS2ACCEL system option 188
DSACCEL= system option 293

E
EM_output variables 31, 42, 62, 98, 156
extension nodes 28

F
fenced mode 58, 115, 237, 259
fixed variable names 27
format files for Aster 222
format publishing macros
  %INDAC_PUBLISH_FORMATS 218
  %INDB2_PUBLISH_FORMATS 232
  %INDGP_PUBLISH_FORMATS 244
  %INDNZ_PUBLISH_FORMATS 254
  %INDTD_PUBLISH_FORMATS 266
special characters in directory names 212
tips for using 214
forms
  determining publish dates 215
  publishing (Aster) 218
  publishing (DB2) 231
  publishing (Greenplum) 243
  publishing (Netezza) 254
  publishing (Teradata) 266
  SAS formats library (DB2) 6
  SAS formats library (Greenplum) 7
  SAS formats library (Netezza) 8
  SAS formats library (Teradata) 9
functions
  See also SAS_PUT() function
  ANALYZE_TABLE 50
  SAEPFUNC 125
  SAS_EP 70

G
GPPC 72
Greenplum
  creating a model table 68
  deployed components for in-database processing 7
GPPC 72
in-database procedures 281
permissions 78, 250
publishing SAS formats 243
SampleSQL.txt file 64
SAS Embedded Process 7, 61
SAS formats library 7
SAS In-Database Code Accelerator 184
SAS_EP function 70
Scoring Accelerator 61
user-defined formats 243
using the SAS Embedded Process to run scoring models 67

H

Hadoop
%INDHD_PUBLISH_MODEL macro 82
%INDHD_RUN_MODEL macro 85
%INDHN_RUN_MODEL macro 139
analytic store scoring 79
deployed components for in-database processing 7
DS2 error messages in Hadoop 95
file types for SAS In-Database Code Accelerator 185
file types for SAS In-Database Code Accelerator 185
HCatalog 186
in-database procedures 281
INDCONN macro variable 81
permissions 95
query output file 92
SAS Embedded Process 7
SAS In-Database Code Accelerator 185
Scoring Accelerator 79
scoring output file 92
Hadoop BY-group processing
SAS In-Database Code Accelerator 187
HCatalog 186
HPFOREST component 30
HPSVM component 30

I

in-database procedures 279
Aster 281
BY-groups 283
column names in Netezza 285
considerations and limitations 282
controlling messaging with
   MSGLEVEL option 289
data set options 284
DB2 281
generating SQL for 294
Greenplum 281
Hadoop 281
items preventing in-database processing 285

LIBNAME statement 284
Netezza 281
Oracle 281
PROC TRANSPOSE 285
row order 283
running 281
SAP HANA 281
SAS formats and 265
Teradata 282
in-database processing
deployed components for Aster 5
deployed components for DB2 6
deployed components for Greenplum 7
deployed components for Hadoop 7
deployed components for Netezza 8
deployed components for Oracle 8
deployed components for Teradata 9
using the SAS Embedded Process 17, 212
using user-defined functions 17, 212
INDCONN macro variable
Aster 32, 219
DB2 52, 233
Greenplum 73, 245
Hadoop 81
Netezza 109, 255
Oracle 121
SPD Server 147
Teradata 172, 267
INDDATA macro variable
SPD Server 147
INTRINSIC-CRDATE format 215

M

macros
%INDAC_PUBLISH_FORMATS 220
%INDAC_PUBLISH_MODEL 34
%INDB2_CREATE_MODELTABLE 49
%INDB2_PUBLISH_FORMATS 234
%INDB2_PUBLISH_MODEL 54
%INDGP_CREATE_MODELTABLE 69
%INDGP_PUBLISH_FORMATS 246
%INDGP_PUBLISH_MODEL 75
%INDHD_PUBLISH_MODEL 82
%INDHD_RUN_MODEL 85
%INDHN_PUBLISH_MODEL 136
%INDHN_RUN_MODEL 139
%INDNZ_CREATE_MODELTABLE 104
%INDNZ_PUBLISH_FORMATS 255
%INDNZ_PUBLISH_MODEL 111
%INDOR_CREATE_MODELTABLE 120
messaging controlling with MSGLEVEL option 289
model registration
Score Code Export node compared with SAS Metadata Server 24
model table
creating in DB2 48
creating in Greenplum 68
creating in Netezza 102
creating in Oracle 119
creating in SAP HANA 133
creating in Teradata 161
running
%INDB2_CREATE_MODELTABL E macro 48
running
%INDGP_CREATE_MODELTABL E macro 68
running
%INDHN_CREATE_MODELTABL E macro 133
running
%INDNZ_CREATE_MODELTABL E macro 103
running
%INDOR_CREATE_MODELTABL E macro 119
running
%INDTD_CREATE_MODELTABL E macro 161
modifying score code 17
MSGLEVEL system option controlling messaging with 289

N
names of scoring functions 43, 63, 98, 156
Netezza
creating a model table 102
deployed components for in-database processing 8
in-database procedures 281
permissions 116, 263
publishing SAS formats 254
SampleSQL.txt file 99
SAS Embedded Process 8, 97
SAS formats library 8
SAS_EP stored procedure 105
Scoring Accelerator 97
user-defined formats 253
using SAS Embedded Process to run scoring models 102
using scoring functions to run scoring models 98
nodes score code created by SAS Enterprise Miner nodes 28
user-defined 28

O
Oracle
creating a model table 119
deployed components for in-database processing 8
in-database procedures 281
permissions 118
SAS Embedded Process 8
SASEPFUNC function 125
Scoring Accelerator 117
using SAS Embedded Process to run scoring models 118
output files 25
output variables 26
EM_ 31, 42, 62, 98, 156
SPD Server 151
output, created by Score Code Export node 25

P
parallel processing 293
permissions
Aster 37, 229
DB2 59, 240
Greenplum 78, 250
Hadoop 95
Netezza 116, 263
Oracle 118
SAP HANA 143
SPD Server 154
Teradata 178, 275
PROC TRANSPOSE
data type conversion with in-database processing 288
in-database processing 285
in-database processing results 289
requirements for in-database processing 286
procedures
See in-database procedures
process flow diagrams
using Score Code Export node in 24
protected mode 269  
protected mode (Teradata) 177  
publishing client 14  
publishing macros  
%INDAC_PUBLISH_FORMATS 218  
%INDAC_PUBLISH_MODEL 34  
%INDB2_CREATE_MODELTABLE 49  
%INDB2_PUBLISH_FORMATS 232  
%INDB2_PUBLISH_MODEL 54  
%INDGP_CREATE_MODELTABLE 69  
%INDGP_PUBLISH_FORMATS 244  
%INDGP_PUBLISH_MODEL 75  
%INDHD_PUBLISH_FORMATS 82  
%INDHD_RUN_MODEL 85  
%INDHN_PUBLISH_FORMATS 104  
%INDHN_PUBLISH_MODEL 111  
%INDOR_CREATE_MODELTABLE 120  
%INDOR_PUBLISH_FORMATS 122  
%INDOR_PUBLISH_MODEL 122  
%INDSP_PUBLISH_FORMATS 148  
%INDSP_PUBLISH_MODEL 150  
%INDTD_CREATE_MODELTABLE 162  
%INDTD_PUBLISH_FORMATS 266  
%INDTD_PUBLISH_MODEL 173  
publishing process 31, 42, 62, 98, 145, 156  
publishing SAS formats  
Aster 218  
DB2 231  
determining format publish dates 215  
Greenplum 243  
Netezza 254  
special characters in directory names 212  
Teradata 266  
tips 214  
publishing scoring model files  
running %INDNZ_PUBLISH_MODEL macro 110  
running %INDOR_PUBLISH_MODEL macro 121  
running %INDAC_PUBLISH_MODEL macro 32  
running %INDB2_PUBLISH_MODEL macro 52  
running %INDGP_PUBLISH_MODEL macro 73  
running %INDTD_PUBLISH_MODEL macro 171  
PUT function  
in-database procedures and 265  
mapping to SAS_PUT function 297  
reducing, based on engine type 298  
R  
registering models  
Score Code Export node compared with SAS Metadata Server 24  
row order  
in-database procedures and 283  
S  
SA_EP function  
syntax 71  
SampleSQL.txt file  
DB2 44  
Greenplum 64  
Netezza 99  
SPD Server 152  
Teradata 158  
SAP HANA  
analytic store scoring 129  
creating a model table 133  
in-database procedures 281  
permissions 143  
query output file 143  
Scoring Accelerator 129  
scoring output table 142  
using SAS Embedded Process to run scoring models 130  
SAS Embedded Process  
Aster 6, 31  
DB2 6, 47  
Greenplum 7, 67, 72  
Hadoop 7  
Netezza 8, 102  
Oracle 8, 118  
SAP HANA 130  
Teradata 9, 160, 171  
SAS Enterprise Miner  
score code created by each node 28  
SAS formats  
%INDAC_PUBLISH_FORMATS macro 218  
%INDB2_PUBLISH_FORMATS macro 232  
%INDGP_PUBLISH_FORMATS macro 244  
%INDNZ_PUBLISH_FORMATS macro 254  
%INDTD_PUBLISH_FORMATS macro 266  
Aster 218
DB2 231
deploying 265
Greenplum 243
in-database procedures and 265
Netezza 254
SAS formats library (DB2) 6
SAS formats library (Greenplum) 7
SAS formats library (Netezza) 8
SAS formats library (Teradata) 9
SAS_PUT() function and 265
Teradata 266
SAS formats library (DB2) 6
SAS formats library (Greenplum) 7
SAS formats library (Netezza) 8
SAS formats library (Teradata) 9
SAS In-Database Code Accelerator 183
BY-group processing 189
considerations 189
DBCREATE_TABLE_OPTS table
option 188
DS2 error messages in Hadoop 188
DS2ACCEL system option 188
examples 192
Greenplum 184
Hadoop 185
Hadoop BY-group processing 187
Hadoop file compression 185
Hadoop file types 185
Hadoop HCatalog 186
overview 183
requirements 183
Teradata 188
SAS Metadata Server
compared with registering models with
Score Code Export node 24
SAS Model Manager
modifying DATA step score code 17
SAS Scoring Accelerator
analytic store scoring 29
Aster 31
components 14, 15
DB2 41
Greenplum 61
Hadoop 79
modifying DATA step score code 17
Netezza 97
Oracle 117
overview for SAS SPD Server 15
overview for SAS/ACCESS databases 13
process flow diagram for SAS/ACCESS
data sources 14
process flow diagram for SPD Server 16
SAP HANA 129
SAS Model Manager 179
SAS SPD Server 145
Teradata 155
SAS SPD Server
running scoring models 145
SAS System libraries
Aster 6
SAS_EP function
overview 70
using 70
SAS_EP stored procedure
overview 105
running 105
syntax 106
tips for using 107
SAS_PUT() function
Aster 218
data types in Teradata 270
DB2 231
explicit use of (Aster) 228
explicit use of (DB2) 239
explicit use of (Greenplum) 250
explicit use of (Netezza) 262
explicit use of (Teradata) 274
Greenplum 243
implicit use of (Aster) 226
implicit use of (DB2) 238
implicit use of (Greenplum) 248
implicit use of (Netezza) 260
implicit use of (Teradata) 272
mapping PUT function to 297
Netezza 254
Teradata 266
tips for using 215
tips for using in Teradata 275
SAS_SCORE_EP stored procedure
overview 163
running 163
syntax 164
tips for using 170
SAS_SCORE( ) function
installation 6
SAS_SCORE() function
overview 39
using 39
SAS/ACCESS LIBNAME statement
in-database procedures and 284
SASEPFUNC function
overview 125
syntax 126
using 125
score code
considerations when creating or
modifying 17
created by each node of SAS Enterprise
Miner 28
Score Code Export node 14, 15, 23
compared with registering models on
SAS Metadata Server 24
files exported by 23
output created by 25
using in process flow diagrams 24
scoring files
creating in Aster 31
creating in DB2 47
creating in Greenplum 67
creating in Netezza 102
creating in Oracle 118
creating in SAP HANA 130
creating in Teradata 160
data example 303, 323, 330
viewing (Aster) 37
viewing (DB2) 51
viewing (Greenplum) 72
viewing (Netezza) 108
viewing (Oracle) 124
viewing (Teradata) 170
scoring functions
names of 43, 63, 98, 156
scoring publishing macro and 14
using to run a DB2 scoring model 46
using to run a Greenplum scoring model 66
using to run a Netezza scoring model 101
using to run a Teradata scoring model 159
viewing 44, 63, 99, 157
scoring publishing macros
%INDAC_PUBLISH_MODEL 34
%INDB2_PUBLISH_MODEL 54
%INDGP_PUBLISH_MODEL 75
%INDH_PUBLISH_MODEL 82
%INDH_RUN_MODEL 85
%INDHN_PUBLISH_MODEL 136
%INDHN_RUN_MODEL 139
%INDNZ_PUBLISH_MODEL 139
%INDOR_PUBLISH_MODEL 111
%INDP_PUBLISH_MODEL 122
%INDP_PUBLISH_MODEL 148
%INDP_RUN_MODEL 150
%INDTD_PUBLISH_MODEL 173
overview 14
SFTP protocol 42
source data
generating SQL for in-database
processing of 294
SPD Server
%INDSP_PUBLISH_MODEL macro 148
%INDSP_RUN_MODEL macro 150
INDCONN macro variable 147
INDDATA macro variable 147
output variables 151
permissions 154
query output tables 152
run model macro overview 15
SampleSQL.txt file 152
Scoring Accelerator 145
scoring publishing macro overview 15
special characters in directory names 20, 212
SQL
Generating for in-database processing of
source data 294
SQLGENERATION system option 281
SQLGENERATION= system option 294
SQLMAPPUTTO= system option 297
SQLREDUCEPUT= system option 298
SSH-2 protocol 42
stored procedure
SAS_EP 105
SAS_SCORE_EP 163
T
Teradata
analytic store scoring 155
creating a model table 161
data types and SAS_PUT() function 270
deployed components for in-database
processing 9
in-database procedures 282
INDCONN macro variable 172
permissions 178, 275
publishing SAS formats 266
SampleSQL.txt file 158
SAS Embedded Process 9, 155
SAS formats library 9
SAS In-Database Code Accelerator 188
SAS_SCORE_EP stored procedure 163
Scoring Accelerator 155
tips for using the SAS_PUT() function 275
user-defined formats 265
using SAS Embedded Process to run
scoring models 160
using scoring functions to run scoring
models 156
threaded processing
SAS In-Database Code Accelerator 183
U
UFMT-CRDATE format 215
unfenced mode 58, 115, 237, 259
unprotected mode 269
unprotected mode (Teradata) 177
user-defined formats
  Aster 217
  DB2 231
  determining publish date 215
  Greenplum 243
  Netezza 253
  SAS_PUT() function 209
  Teradata 265
  user-defined nodes 28

V
variables
  EM_output variables 42, 62, 98, 101, 145, 156, 159
  fixed variable names 27
  output variables 26
Gain Greater Insight into Your SAS® Software with SAS Books.

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

support.sas.com/bookstore for additional books and resources.