SAS® Model Manager 14.3: Macro Reference
## Contents

**Chapter 1 • Introduction** .......................... 1
  Overview ................................................. 1
  Setting Metadata Connection System Options ............ 2
  Macro Variables ........................................ 4

**Chapter 2 • Model Management Access Macros** ............ 11
  Overview of Access Macros ............................. 11
  Using the Model Management Access Macros ........... 12
  Dictionary .............................................. 16

**Chapter 3 • Macros for Registering Models to the SAS Metadata Repository** ............. 47
  Using Macros to Register Models Not Created by SAS Enterprise Miner .......... 47
  Dictionary .............................................. 51

**Chapter 4 • Macros for Adding Folders, Projects, Versions, and Setting Properties** ........ 57
  Adding Folders, Projects, Versions, and Properties Using Macros ................. 57
  Dictionary .............................................. 62
  Example: Add a Folder, Project, and Version; Set Properties ...................... 67

**Chapter 5 • Macros for Generating Score Code** ............. 69
  Generating Score Code for COUNTREG Procedure Models ......................... 69
  Generating Score Code for SEVERITY Procedure Models .......................... 70
  Dictionary .............................................. 70

**Chapter 6 • Macros for Creating Performance Reports in Batch** .................... 99
  Overview of SAS Programs to Monitor Model Performance ...................... 100
  Prerequisites for Running Batch Performance Reports .......................... 101
  Report Output in Test and Production Modes .............................. 104
  Define the Report Specifications ................................ 106
  Extracting the Champion Model from a Channel ............................... 117
  SAS Code to Run Performance Reports ................................ 120

**Chapter 7 • Macros for Publishing and Scoring Models from within a Portfolio** ........... 129
  Overview of Portfolio Macros ................................ 129
  Prerequisites .......................................... 130
  Dictionary .............................................. 130

**Chapter 8 • Macro for Converting and Exporting Model DS2 Code** ............. 141
  Overview .............................................. 141
  Dictionary .............................................. 142

**Chapter 9 • Feature Contribution Index Macros** .................. 149
  Overview .............................................. 149
  How the Model Outcome Is Determined ....................................... 150
  Aggregation for Nominal Targets .......................................... 150
  Dictionary .............................................. 151
  Examples .............................................. 158

**Chapter 10 • Using SAS Viya Models with SAS Model Manager** ........... 175
  Overview .............................................. 175
  Prerequisites for Running Code Examples ...................................... 175
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure a SAS 9.4 Client to Work with SAS Viya</td>
<td>176</td>
</tr>
<tr>
<td>Create a Model in SAS Viya</td>
<td>176</td>
</tr>
<tr>
<td>Registering Models in SAS Model Manager</td>
<td>182</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Overview

SAS Model Manager provides a set of macros that you can use in your SAS programs to manage models that are within projects and portfolios, as well as many aspects of the model life cycle. You can add folders, projects, and versions; set properties; and register models in the model repository. As part of the model life cycle, you can score models, monitor performance, create reports, and publish models to the SAS Metadata Repository, a database, or Hadoop.

Note: If your SAS environment is in the locked-down state, your access to the file system of the host operating environment might be restricted. In order to run the SAS Model Manager macros, your system administrator must enable the HTTP access method by specifying ENABLE_AMS=HTTP in the LOCKDOWN statement. For more information, see “Locked-Down Servers” in SAS Intelligence Platform: Security Administration Guide and “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide.

Here are the types of macros that are available for use:

- Model Management access macros
- Macros for registering models
- Macros for adding folders, projects, and versions, and for setting properties
- Macros for generating score code
- Macros for creating performance reports in batch
- Macros for publishing and scoring models from within a portfolio
• Macros for computing the feature contribution indices for interval and nominal predictors
• Macro for converting and exporting model DS2 code

**TIP** You can also use the SAS Model Manager macros to register SAS Viya models into the model repository. For more information, see Chapter 10, “Using SAS Viya Models with SAS Model Manager,” on page 175.

### Setting Metadata Connection System Options

Before running macros in a SAS program that requires a connection to the SAS Metadata Server, specify the metadata connection system options. If these options are not specified, you will be prompted for the information. If you are running the macros in batch mode, you must specify the metadata server connection system options. Otherwise, you will receive a failure message.

Here is a sample OPTIONS statement that specifies these options:

```sas
options metaPort=8561
    metaServer=server-address
    metaRepository=Foundation
    metaUser=user-ID
    metaPass=password;
```

For more information, see “Connection Options ” in SAS Language Interfaces to Metadata.

Example 1: Specifying the metadata server connection system options when registering models to the SAS Metadata Repository using the `%AAMODEL` and `%AA_MODEL_REGISTER` macros.

```sas
options mlogic mprint symbolgen;

/* Set up the metadata connection system options. */

options metaPort=8561
    metaServer=server-address
    metaRepository=Foundation
    metaUser=user-ID
    metaPass=password;

libname mmlib "!sasroot\mmcommon\sample";

proc logistic data=mmlib.hmeq_train;
   class bad job reason;
   model bad = CLAGE CLNO DEBTINC DELINQ DEROG JOB NINQ REASON VALUE YOJ;
   store work.itemstore;
run;QUIT;

%aamodel;
%aa_model_register(
   ModelName=model_1,
   itemstore=work.itemstore,
   spk=Y,
)
Example 2: Using the metadata connection system options when accessing files in the model repository. This macro copies the specified model file to the specified location on a local or network computer.

```sas
/******************************************************/
/* Get the score code from a registered model and run */
/* it.                                                */
/******************************************************/
Options NOmlogic NOmprint NOspool;

/******************************************************/
/* Set up the metadata connection system options. */
/******************************************************/
options metaPort=8561
    metaServer=a123.us.company.com
    metaRepository=Foundation
    metaUser=myuserID
    metaPass=sasuser1;

/******************************************************/
/* Get the Model Management macro code. */
/******************************************************/
FILENAME MMAccess catalog 'sashelp.modelmgr.accessmacros.source';
%include MMAccess;

/******************************************************/
/* Specify the model component filename and */
/* destination. */
/******************************************************/
%let WorkPath = c:\myProject\1.0;
FILENAME dest '&WorkPath.\score.sas';

/******************************************************/
/* Set the return code to detect failure in case the */
/* macro load fails. */
/******************************************************/
%let _MM_RC = -1;

/******************************************************/
/* Get score code. */
/******************************************************/
%M_GetModelFile(name=score.sas,
    ModelId=//ModelManagerRepo/MMRoot/HomeEquity/HMEQ/1.0/
    DecisionTree, TextFile=dest);
```
Macro Variables

SAS Environment Macro Variables

The following table lists the macro variables that are used to set the SAS environment:

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_Password</td>
<td>the password of the user ID that is running the macro</td>
<td>mdlmgrpw2</td>
</tr>
<tr>
<td>_MM_User</td>
<td>the user ID of the user that is running the macro</td>
<td>mdlmgradmin</td>
</tr>
</tbody>
</table>

Scoring Test Macro Variables

The following table lists the macro variables that are used to run a scoring test:

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_InputDS</td>
<td>two-level SAS data set name for the input data source</td>
<td>inlib.hmeq_score_input</td>
</tr>
<tr>
<td>_MM_InputLib</td>
<td>the libref that is associated with the location of the input data source file</td>
<td>inlib</td>
</tr>
<tr>
<td><strong>Macro Variable Name</strong></td>
<td><strong>Description</strong></td>
<td><strong>Example Value</strong></td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>_MM_ModelID</td>
<td>the UUID of the model</td>
<td>4622bdda-ac1b-12d5-0196-021edec54347</td>
</tr>
<tr>
<td>_MM_OutputDS</td>
<td>two-level SAS data set name for the output data source</td>
<td>outlib.hmeq_score_output</td>
</tr>
<tr>
<td>_MM_OutputLib</td>
<td>the libref that is associated with the location of the output data source file</td>
<td>outlib</td>
</tr>
<tr>
<td>_MM_Password</td>
<td>the password of the user ID that is running the report</td>
<td>mdlmgrpw2</td>
</tr>
<tr>
<td>_MM_PerformanceDS</td>
<td>two-level SAS data set name for the performance data source</td>
<td>perflib.hmeq_perf_q1</td>
</tr>
<tr>
<td>_MM_PerformanceLib</td>
<td>the libref that is associated with the location of the performance data source file</td>
<td>perflib</td>
</tr>
<tr>
<td>_MM_TaskDir</td>
<td>the URL of the stored scoring test</td>
<td><a href="http://myserver.mycompany:7980/SASContentServer/repository/default/ModelManager/MMRoot/DDHMEQ/HMEQ/1.0/Scoring">http://myserver.mycompany:7980/SASContentServer/repository/default/ModelManager/MMRoot/DDHMEQ/HMEQ/1.0/Scoring</a></td>
</tr>
<tr>
<td>_MM_TestDS</td>
<td>two-level SAS data set name for the test data source</td>
<td>testlib.hmeq_test</td>
</tr>
<tr>
<td>_MM_TestLib</td>
<td>the libref that is associated with the location of the test source file</td>
<td>testlib</td>
</tr>
<tr>
<td>_MM_TrainDS</td>
<td>two-level SAS data set name for the train data source</td>
<td>trainlib.hmeq_train</td>
</tr>
<tr>
<td>_MM_TrainLib</td>
<td>the libref that is associated with the location of the train source file</td>
<td>trainlib</td>
</tr>
<tr>
<td>_MM_User</td>
<td>the user ID of the user that is running the report</td>
<td>mdlmgradmin</td>
</tr>
</tbody>
</table>
## Validating Model Report Macro Variables

The following tables lists the macro variables that are used to create model comparison reports, model profile reports, delta reports, dynamic lift reports, and user reports:

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_LocationInfo</td>
<td>the location information for a model</td>
<td>/MMRoot/Mortgages/HMEQ/1.0</td>
</tr>
<tr>
<td>_MM_ModelFlag</td>
<td>the value of the champion model flag</td>
<td>0 - champion model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - challenger model</td>
</tr>
<tr>
<td>_MM_ModelLabel</td>
<td>a label for a model</td>
<td>Decision Tree</td>
</tr>
<tr>
<td>_MM_ModelName</td>
<td>the name of the model</td>
<td>Tree</td>
</tr>
<tr>
<td>_MM_Password</td>
<td>the password of the user ID that is running the report</td>
<td>mdlmgrpw2</td>
</tr>
<tr>
<td>_MM_PosteriorVar</td>
<td>the model’s posterior variable name</td>
<td>EM_EVENTPROBABILITY</td>
</tr>
<tr>
<td>_MM_ProjectName</td>
<td>the name of the project</td>
<td>HMEQ</td>
</tr>
<tr>
<td>_MM_ReportFormat</td>
<td>the output format of the generated report</td>
<td>html</td>
</tr>
<tr>
<td>_MM_ReportLib</td>
<td>the libref for the Report node</td>
<td>report</td>
</tr>
<tr>
<td>_MM_ResourcesLib</td>
<td>the libref for the Resources node</td>
<td>resources</td>
</tr>
<tr>
<td>_MM_SampleSize</td>
<td>the size of a sample</td>
<td>1000</td>
</tr>
<tr>
<td>_MM_SampleSeed</td>
<td>the sample seed</td>
<td>12345</td>
</tr>
<tr>
<td>_MM_ScoreCodeType</td>
<td>the type of score code</td>
<td>SAS Program</td>
</tr>
<tr>
<td>_MM_TargetEvent</td>
<td>the target event value</td>
<td>1</td>
</tr>
<tr>
<td>_MM_TargetVar</td>
<td>the target variable name</td>
<td>bad</td>
</tr>
<tr>
<td>_MM_TaskDir</td>
<td>the URL of the stored report</td>
<td><a href="http://myserver.mycompany:7980/">http://myserver.mycompany:7980/</a></td>
</tr>
</tbody>
</table>
### Macro Variables

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_User</td>
<td>the user ID of the user that is running the report</td>
<td>mdlmgradmin</td>
</tr>
</tbody>
</table>

---

### Performance Monitoring Report Macro Variables

The following table lists the macro variables that are used to create performance monitoring reports:

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_Agg_Mail</td>
<td>specifies whether to send aggregated mail for performance monitoring with multiple data sources</td>
<td>Y or N</td>
</tr>
<tr>
<td>_MM_DateTime</td>
<td>the time that the performance task is to run</td>
<td>1Sep2013:05:00:00</td>
</tr>
<tr>
<td>_MM_Hpds2_Flg</td>
<td>enables high-performance monitoring if set it to 1, is used with the _MM_Hpdm_Performance macro variable</td>
<td>1</td>
</tr>
<tr>
<td>_MM_Hpdm_Performance</td>
<td>the configuration settings for high-performance monitoring</td>
<td>%nrstr(performance commit=10000 cpucount=ACTUAL dataserver='tera2650' timeout=120 host='tms2650' install='/opt/v940/laxno/TKGrid';)</td>
</tr>
<tr>
<td>_MM_ModelName</td>
<td>the name of the champion model</td>
<td>reg1</td>
</tr>
<tr>
<td>_MM_ModelID</td>
<td>the UUID of the champion model</td>
<td>7514d6e-ac1b-12d5-01e4-878abeb04505</td>
</tr>
<tr>
<td>_MM_ModelLocalPath</td>
<td>the location of the SAS Work library in the SAS Application Server</td>
<td>C:\DOCUME<del>1\ADMINI</del>1\LOCALALS~1\Temp\1\SAS Temporary Files\TD2032_BRDVM0199_</td>
</tr>
<tr>
<td>_MM_Password</td>
<td>the password of the user ID that is running the report</td>
<td>mdlmgrpw2</td>
</tr>
<tr>
<td>_MM_ProjectPath</td>
<td>the network path to the model project in the model repository</td>
<td>//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HMEQ</td>
</tr>
<tr>
<td>Macro Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>_MM_ProjectURLPath</td>
<td>the URL to the model project in the model repository</td>
<td><a href="http://myserver.mycompany.com:7980/SASContentServer/repository/default/ModelManager/MMRoot/HMEQ">http://myserver.mycompany.com:7980/SASContentServer/repository/default/ModelManager/MMRoot/HMEQ</a></td>
</tr>
<tr>
<td>_MM_ProjectUUID</td>
<td>the project UUID</td>
<td>27514d6e-ac1b-12d5-01e4-878abeb04505</td>
</tr>
<tr>
<td>_MM_Seg_Filter</td>
<td>filters the performance data for each sub-project from the top level performance datasource by using this macro variable</td>
<td>%nrstr(Location='USA')</td>
</tr>
<tr>
<td>_MM_ScoreCodeType</td>
<td>the type of score code</td>
<td>SAS Program</td>
</tr>
<tr>
<td>_MM_VersionName</td>
<td>the name of the default version</td>
<td>1.0</td>
</tr>
<tr>
<td>_MM_ReportDatasrc</td>
<td>the project’s performance data set</td>
<td>mmlib_hmeq_perf_q1</td>
</tr>
<tr>
<td>_MM_PreCode</td>
<td>one or more macro variables that set values to performance variables</td>
<td>%let _MM_EventProbVar=score; %let _MM_TargetVar=bad;</td>
</tr>
<tr>
<td>_MM_ResultURLPath</td>
<td>the URL to the version’s Resources node</td>
<td><a href="http://myserver.mycompany.com:7980/SASContentServer/repository/default/ModelManager/MMRoot/HMEQ/1.0/Resources">http://myserver.mycompany.com:7980/SASContentServer/repository/default/ModelManager/MMRoot/HMEQ/1.0/Resources</a></td>
</tr>
<tr>
<td>_MM_TimeLabel</td>
<td>the label that is used in reports to represent the time period of the data in the performance data set</td>
<td>Q1</td>
</tr>
<tr>
<td>_MM_Trace</td>
<td>indicates whether to write a trace log</td>
<td>ON or OFF</td>
</tr>
<tr>
<td>_MM_User</td>
<td>the user ID of the user that is running the report</td>
<td>mdlmgradmin</td>
</tr>
</tbody>
</table>

**Dashboard Report Macro Variables**

The following table lists the macro variables that are used to create dashboard reports:
### Model Retrain Report Macro Variables

The following table lists the macro variables that are used to retrain models:

<table>
<thead>
<tr>
<th>Macro Variable</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_Hpds2_Flg</td>
<td>enables high-performance monitoring if set to 1, is used with the _MM_Hpdm_Performance macro variable</td>
<td>1</td>
</tr>
<tr>
<td>_MM_Hpdm_Performance</td>
<td>the configuration settings for high-performance monitoring</td>
<td>%nrstr(performance commit=10000 cpucount=ACTUAL dataserver='tera2650' timeout=120 host='tms2650' install='/opt/v940/laxno/TKGrid';)</td>
</tr>
<tr>
<td>_MM_Password</td>
<td>the password of the user ID that is running the report</td>
<td>mdlmgrpw2</td>
</tr>
<tr>
<td>_MM_User</td>
<td>the user ID of the user who is running the report</td>
<td>mdlmgradmin</td>
</tr>
</tbody>
</table>
Chapter 2
Model Management Access Macros

Overview of Access Macros

The Model Management access macros provide a way to use SAS code to perform basic operations on a model repository. The Model Management access macros are a combination of SAS macros and Java libraries. The Model Management access macros and Java libraries are delivered with the SAS Model Manager software.

Here is a list of the Model Management access macros:

- %MM_AddModelFile Macro adds a model component file to a model that is already registered with SAS Model Manager.
- %MM_GetModelFile Macro retrieves a model file from the model repository and saves it to a specified destination.
- %MM_GetURL Macro retrieves the SAS Model Manager path to an object in the model repository and saves it in the global macro variable _MM_URL.
- %MM_Register Macro registers a model in the model repository. You can use the %MM_Register macro in the same SAS program that you create models using SAS Enterprise Miner to register the model for use with SAS Model Manager.
- %MM_RegisterByFolder Macro registers multiple models simultaneously to the model repository. Model files for a single model are contained in a subdirectory, and all subdirectories have the same parent directory.
- %MM_CreateModelDataset Macro
%MM_CreateModelDataset creates a data set that contains information for all models in a specified location. Model information can be retrieved in a data set for all models in MMRoot, an organizational folder, a project, a version, and a single model.

Note: The macros are in the modelmgr.sas7bcat file. The location of this file for Windows is `\sasinstalldir\SASFoundation\9.4\mmcommon\sashelp`. The default value for `sasinstalldir` in Windows is `C:\Program Files \SASHome`. The location of this file for UNIX is `/sasinstalldir/SASFoundation/9.4/sashelp`. The default value for `sasinstalldir` in UNIX is `/usr/local/SASHome`.

To use the Model Management access macros, you can structure your SAS program as follows:

1. Specify the metadata connection system options, if you are not already connected to the SAS Metadata Server. For more information, see “Setting Metadata Connection System Options” on page 2.
2. Create a fileref to the Model Management access macro catalog and include that fileref, using the `%INCLUDE` statement.
3. Set up librefs to access the necessary directories and filerefs to access the necessary files.
4. Set up macro variables as necessary.
5. Execute the macro.
6. Check for successful completion.

### Using the Model Management Access Macros

#### Global Macro Variables

Your SAS program and SAS Model Manager use global macro variables to pass information about the SAS environment and the model repository to the access macros. Some macros set these global macro variables. You can set any of these global macro variables in your SAS program. At the end of each macro execution, the global macro variable `_MM_RC` is set to a number that indicates either that the macro executed successfully or that there was an error.

Note: The `_MM_Service_Registry_URL` macro variable is no longer used. If your SAS program contains this macro variable, it will be ignored.

Here is a description of the Model Management global macro variables:

- **_MM_CId**
  contains the name of the current object identifier. _MM_CId is either the UUID or the SAS Model Manager path to the object in the model repository. You can use the `%MM_GetURL` to obtain a URL for any object in the model repository.

  The `%MM_Register` macro sets _MM_CId to contain the identifier for the registered model. The `%MM_AddModelFile` macros sets _MM_CId to the identifier for the model to which the file was added.

- **_MM_RC**
  contains one of the following return codes after processing a Model Management access macro:
### _MM_RC Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Access Macro Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All OK</td>
</tr>
<tr>
<td>1</td>
<td>Macro parameter error</td>
</tr>
</tbody>
</table>

#### _MM_ResourceURL

Contains the URL of the **Resources** folder. The _MM_Resource URL is set by the %MM_GetURL macro when the macro returns a version URL in the _MM_URL global macro variable.

#### _MM_URL

Contains a URL for a SAS Model Manager object. The %MM_GetURL macro returns a URL in the _MM_URL global macro variable.

#### _MM_User

Contains the name of a SAS Model Manager user on the server that is specified by the _MM_MulticastAddress global macro variable.

Default: the value of SAS automatic macro variable &SYSUSERID.

#### _MM_Password

Contains a password for the SAS Model Manager user. If you do not encode the password using the PWENCODE procedure, the password is printed in the SAS log.

**See:** “Encoding SAS Model Manager User Passwords” on page 122

**Note:** The _MM_User and _MM_Password access macros need to be specified only if you want to access the SAS Content Server with FILENAME or LIBNAME statements.

For a description of these macro variables as well as their default values, see “Global Macro Variables” on page 12.

If you are running macros in a SAS session that requires a connection to the SAS Metadata Server, you must specify the metadata connection system options before you run the macros. If these options are not specified, you will be prompted for the information. If you are running the macros in batch mode, you must include the metadata connection system options. Otherwise, you will receive a failure message. For more information, see “Setting Metadata Connection System Options” on page 2.

#### See Also

“Macro Variables” on page 4

### Accessing the Macros

Before you can use the access macros, your SAS program must access the catalog where the macros are located, and load the macros into memory. Here is example code to do this:

```plaintext
/*******************************************/
/* Specify the macro code location     */
/*******************************************/

Filename MMAccess catalog "sashelp.modelmgr.accessmacros.source";
```
Identifying Model Repository Objects

The access macros use an identifier to specify a unique object such as the version or a model, in the model repository. The identifier can be in the form of a Universally Unique Identifier (UUID) or a SAS Model Manager path.

- A UUID is a case sensitive, 36-character string that uniquely identifies the repository object. An example UUID is cca1ab08-0a28-0e97-0051-0e3991080867.

  If you need to find the UUID or the exact SAS Model Manager path for an object, you can look it up in SAS Model Manager on the System tab of the Models Properties page. The UUID and path values are listed there.

- The format for a SAS Model Manager path is //repository/MMRoot/folder/project/version/Models/model.

  The name of the repository is defined during installation. The names of the folder, project, version, and model that follow in the path are user-defined. SAS Model Manager path specifications always use the forward slash character (/) as a separator.

  For example, a version path might look like //ModelManagerDefaultRepo/MMRoot/HomeEquity/HMEQ/1.0.

  You use the _MM_CId global macro variable to pass a model repository identifier to an access macro. For more information, see “_MM_CId” on page 12.

Identifying Files Used by Access Macros

All Model Management access macros that accept SAS file references require the file references to point to a single physical file. File references in the form libref.filename must resolve to a single physical file. Specific logical library references in the form libref must resolve to a directory or a folder.

Concatenated library references cannot be used.

Here is a list of libraries to which you must assign a libref in your SAS programs:

- the directory that contains your model files
- the directory that contains the training data
- the directory that contains your input, output, and target data sets

Model Management access macros use the libref SMMMMODEL to access model component files, as in this example:

    libname smmmodel "c:\myModel\HMEQ\scorecode";

You can define the libref SMMMMODEL at the beginning of your SAS program and use it to access model component files in any of the Model Management access macros that your program executes.
Here is a list of files that you can identify with a fileref in your SAS programs:

- a catalog fileref to the Model Management access macro code
- the source path and filename for a single file to be registered by the %MM_AddModelFile macro
- the source path and filename for a SAS Enterprise Miner package file to be registered by the %MM_Register macro
- the destination path and filename for the %MM_GetModelFile macro

**Required Tables**

Whether you use SAS Model Manager or the access macros, SAS Model Manager must know the model input variables, the output variables, and the target variables to register a model. SAS Model Manager uses an XML file to describe each of these types of files. Before you can register a SAS code model, you must create a SAS data set that represents the input, output, and target variables:

- The model input table contains the variables that are used as input by the model. During model registration, SAS Model Manager uses this table to create the inputvar.xml file.
- The model output table is a table whose variables contain the model output values. During model registration, SAS Model Manager uses this table to create the outputvar.xml file.
- The model target variable table is a table whose one variable is the target variable that is used in the training data. During model registration, SAS Model Manager uses this file to create the targetvar.xml file.

Each of these tables can be a one-row table. The tables' purpose is to define and describe the variables that are used by the model.

You can create each of these tables using the training data that you used to train your model. The following example SAS program uses the training data to create all three tables:

```sas
/* Set the location for the model tables */
libname hmeqtabl "c:\myModel\hmeq\tables";

data hmeqtabl.target;
   set hmeqtabl.training(obs=1);
   keep bad;
run;

/* DATA step to create the input variable table. */
/* Keep only the variables used for input by the model. */
```
data hmeqtabl.invars;
    set hmeqtabl.training (obs=1);
    keep debtinc delinq derog job loan mortdue ning reason value yoj;
run;

/***********************************************************/
/* DATA step to create the output variable table. */
/* Keep only the variables used for output by the model. */
/* Include the score code to get the output variables. */
/***********************************************************/
data hmeqtabl.outvars;
    set hmeqtabl.training;
    %include "c:\myModel\hmeq\score.sas";
    keep f_bad i_bad p_0 p_1;
run;

Dictionary

%MM_AddModelFile Macro
Add model component files to an existing SAS Model Manager model.

Syntax

%MM_AddModelFile (ModelId=path-to-model, SASDataFile=SAS-data-set-reference | SASCatalog=SAS-catalog-reference | TextFile=fileref-to-text-file | BinaryFile=fileref-to-binary-file <, Name=alternateFileName> <, Trace=OFF | ON > );

Arguments

ModelId=path-to-model
specifies an identifier of the model in the model repository. The identifier specifies the location in the model repository where the file is to be added. path-to-model can be either a SAS Model Manager UUID or a SAS Model Manager path. ModelId is a required argument. The default value is the value of the _MM_CId macro variable.

Examples

ModelId=8904daa1-0a29-0c76-011a-f7bb587be79f
ModelId=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HomeEquity/1.0/Models/HMEQ%20Loan%20Project
SASDataFile=\textit{SAS-data-set-reference}

specifies the location of the SAS data set to add to a model in the model repository. \textit{SAS-data-set-reference} must be a two-level SAS name in the form \texttt{libref.filename}.

Example: \texttt{SASDataFile=myaslib.hmeqloan}

SASCatalog=\textit{SAS-catalog-reference}

specifies the location of a SAS catalog that includes one or more SAS code model component files to add to a model in the SAS Model Manager repository. \textit{SAS-catalog-reference} must be a two-level SAS name in the form \texttt{libref.catalog}. Use the SASCatalog argument to add the catalog to a model.

Example: \texttt{SASCatalog=mylib.modelinput}

TextFile=\textit{fileref-to-text-file}

specifies the file reference to the location of a SAS code model component file that is an ASCII text file. \textit{fileref-to-text-file} is a one-level SAS reference to a model component file.

Example: \texttt{TextFile=inputxml}

BinaryFile=\textit{fileref-to-binary-file}

specifies the file reference to the location of a SAS code model component file that is a binary file. \textit{fileref-to-binary-file} is a one-level SAS reference to a model component file that is not a text file.

Example: \texttt{BinaryFile=gainscsv}

Name=\textit{alternateFileName}

specifies a name for the file that you are adding. Use the Name argument when your model component filename does not follow the SAS Model Manager model component file naming convention that is specified in the model's template file or your model requires a file to have a particular filename. If Name is not specified, the filename that is registered is the name of the file.

Example: \texttt{Name=score.sas}

Trace=\texttt{ON} | \texttt{OFF}

specifies whether to supply verbose trace messages to the SAS log.

Default: \texttt{OFF}

Example: \texttt{Trace=on}

**Details**

For models that require model component files other than the score code, you can use the \texttt{%MM_AddModelFile} macro to add model component files to a registered model, one file at a time. All files that are added using the \texttt{%MM_AddModelFile} macro are placed in the model repository. After files have been added, you can view the files in the Models page of a project.

The \texttt{%MM_AddModelFile} macro supports two types of files, text and binary. Text files are ASCII files that contain character data. Binary files are files created by an
application in a format specific to that application. If you are adding a text file, you must use the TextFile argument to specify the file. To avoid any unintentional character translations, all non-text files should be added using the BinaryFile argument.

SAS data sets and SAS catalogs are both binary files. Instead of using the BinaryFile argument to add SAS files, you can use the SASDataFile and SASCatalog arguments respectively to add files using the SAS two-level references `libref.filename` or `libref.catalog`. The TextFile and BinaryFile arguments require a single SAS filename that can be a fileref.

The ModelId argument defaults to the value of the global variable `_MM_CIld`. For example, after a call to the `%MM_Register` macro, the `_MM_CIld` variable is set to the identifier for the registered model. In this case, you can use the `%MM_AddModelFile` macro to add additional component files to your model without having to explicitly specify the ModelId argument.

When you use the `%MM_AddModelFile` macro to add a component file to your SAS Model Manager model, the name of the added component file remains unchanged by default. If you need to change the name of the component file when you save it to a SAS Model Manager model, you can use the Name argument to specify the new component filename. Whenever possible, you should try to follow the component file naming conventions that are specified in the model's template file. When you use the model template file naming conventions, you are less likely to be confused about filenames.

**Example**

```sas
/* Adding a file to a registered model. */
Options NOmlogic NOmprint NOspool;

FILENAME MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;

LIBNAME mtbls 'c:\mysascode';

/* A LIBNAME for a table. */
LIBNAME mtbls 'c:\mysascode';

/* Set to detect failure in case macro load fails */
/* and add the input data source. */
%let _MM_RC= -1;

%MM_AddModelFile(
ModelId= //ModelManagerRepo/MMRoot/HomeEquity/HMEQ/1.0/hmeqDecTree1,
Name=modelinput.sas7bdat,
SASDataFile=mtbls.myInputVariables,
)```
FILENAME tcode 'c:\myModel\inputvar.xml';

/** A FILENAME for a text file. */
FILENAME tcode 'c:\myModel\inputvar.xml';

/***************************/
/* Set to detect failure in case macro load fails */
/* and add the xml file for the input data source */
/***************************/

%let _MM_RC= -1;

%MM_AddModelFile{
   ModelId= //ModelManagerRepo/MMRoot/HomeEquity/HMEQ/1.0/hmeqDecTree1,
   TextFile=tcode,
   Trace=on};

%MM_GetModelFile Macro
Access files in the model repository. This macro copies the specified model file to the specified location on a local or network computer.

Syntax

%MM_GetModelFile ( 
   ModelId= path-to-model | VersionId= path-to-version | ProjectId= path-to-project, 
   TextFile= fileref-to-text-file | BinaryFile= fileref-to-binary-file 
   <, Name= alternateFilename> 
   <, Trace= ON | OFF>
);

Arguments

ModelId= path-to-model
specifies an identifier to the model in the model repository. path-to-model can be either a SAS Model Manager UUID or a SAS Model Manager path that describes the location of the specific model. ModelId is a required argument. The default value is the value of the _MM_CId macro variable.

Examples

ModelId=b2341a42-0a29-0c76-011a-f7bb7bc4f1e9
ModelId=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HomeEquity/1.0/Models/HMEQ%20Loan%20Project
**VersionId**

specifies an identifier of the version to where a champion model resides in the model repository. *path-to-version* can be either a SAS Model Manager UUID or a SAS Model Manager path that describes the location of the version.

**Examples**

```
VersionId=b23327cb-0a29-0c76-011a-f7bb3d790340
VersionId=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HomeEquity/1.0
```

**ProjectId**

specifies an identifier of the project object. The identifier specifies the location where the champion model under the default version resides in the model repository. *path-to-project* can be either a SAS Model Manager UUID or a SAS Model Manager path that describes the location of the project.

**Examples**

```
VersionId=b232d766-0a29-0c76-011a-f7bb50921b42
VersionId=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HomeEquity
```

**SASDataFile**=`SAS-data-set-reference`


**Example**

```
SASDataFile=mylib.modelinput
```

**SASCatalog**=`SAS-catalog-reference`

specifies the library reference for the location of the SAS catalog to store a SAS catalog file. *SAS-catalog-reference* must be a two-level SAS name in the form `libref.catalog`.

**Example**

```
SASCatalog=mylib.format
```

**TextFile**=`fileref-to-text-file`

specifies the file reference for the location of a component file that is an ASCII text file. *fileref-to-text-file* is a one-level SAS reference to a model component file.

**Example**

```
TextFile=myfileref
```

**BinaryFile**=`fileref-to-binary-file`

specifies the file reference for the location of a model component file that is a binary file. *fileref-to-binary-file* is a one-level SAS reference to a model component file that is not a text file.

**Example**

```
BinaryFile=myfileref
```

**Name**=`alternateFileName`

specifies a name for the model component file that you are retrieving. Use the Name argument when the name of the destination file does not match the name of the file in the model repository. The Name argument is the filename within the model repository. If Name is not specified, the filename that is registered in the model repository is the name of the file.

**Example**

```
Name=score.sas
```

**Trace**=`ON | OFF`

specifies whether to supply verbose trace messages to the SAS log.
Details

Use the `%MM_GetModelFile` macro to retrieve a component file for a model that has been registered in the model repository. You can retrieve a component file for any model by specifying the repository location of the model, or you can retrieve a component file for a champion model by specifying the version or project location in the model repository.

The `%MM_GetModelFile` macro supports two types of files, text and binary files. Text files are ASCII files that contain character data. Binary files are files that are created by an application in a format that is specific to that application. If you are retrieving a text file, you must use the `TextFile` argument to specify the file. To avoid any unintentional character translations, all non-text files should be retrieved by using the `BinaryFile` argument.

SAS data files and SAS catalogs are binary files. Instead of using the `BinaryFile` argument to retrieve model component files to store as a SAS file or in a SAS catalog, you can use the `SASDataFile` and `SASCatalog` arguments respectively to specify the SAS location to store the file. The `TextFile` and `BinaryFile` arguments require a single SAS filename.

You can use the optional `Name` argument if you want to save the model component file with a different name from the name within the model repository.

After you use the `%MM_GetModelFile` macro to copy a model component file to its new location, you can use the model component file for any purpose. For example, a simple application might use the `%MM_GetModelFile` macro to copy a registered model's score code file to the SAS WORK library. After the score code is copied to WORK, you can write SAS code that includes the score code in a SAS DATA step and is executed for experimental purposes.

If the destination file argument or the two-level SAS library reference name that is invoked in the macro uses the original filename, you do not need to specify the `Name` argument. In other words, the macro can use the SAS logical names to determine the name of the file in the model hierarchy. If the name of the destination file needs to be different from the name of the original file that was copied, use the `Name` argument to specify the new name for the model component file.

Example

```sas
/* Get the score code from a registered model and run it. */
Options NOmlogic NOmprint NOspool;
FILENAME MMAccess catalog 'sashelp.modelmgr.accessmacros.source';```
%include MMAccess;

/***************************************************************************/
/* Specify the model component file name and */
/* destination. */
/***************************************************************************/

%let WorkPath = c:\myProject\1.0;
FILENAME dest '&WorkPath.\score.sas';

/***************************************************************************/
/* Set to detect failure in case macro load fails. */
/***************************************************************************/

%let _MM_RC = -1;

/***************************************************************************/
/* Get score code. */
/***************************************************************************/

%MM_GetModelFile(ModelId=//ModelManagerRepo/MMRoot/HomeEquity/HMEQ/1.0/
DecisionTree, TextFile=dest);

/***************************************************************************/
/* Display Model Management set macro variables. */
/***************************************************************************/

Options nosource;
%PUT _MM_RC = &_MM_RC;
%PUT _MM_CId = &_MM_CId;
Options source;

/***************************************************************************/
/* Run score code. Specify the LIBNAME input path. */
/***************************************************************************/

LIBNAME input 'c:\mysascode\1.0\DTree';
DATA score;
  set input.dTreeInp;
%include dest;
run;

**%MM_GetURL Macro**

Translates a specified SAS Model Manager UUID to a URL-style path address and sets the URL as the value of the _MM_URL and _MM_ResourcesURL macro variables.

**Syntax**

```
%MM_GetURL(ID=UUID, <Trace=ON | OFF>);
```
Arguments

ID=UUID

specifies the UUID of the object for which an URL is desired. A SAS Model Manager UUID is a 36-character string that identifies a single object in the model repository. The ID argument is required.

Example

ID=cca1ab08-0a28-0e97-0051-0e3991080867

Trace=ON | OFF

specifies whether to supply verbose trace messages to the SAS log.

Default OFF

Example Trace=on

Details

The %MM_GetURL macro sets the value of the global macro variable _MM_URL to the URL of the specified object’s UUID.

If the ID argument specifies a SAS Model Manager project, version, or model, then the macro sets the global macro variable _MM_ResourcesURL to the URL of that object’s associated Resources folder.

The %MM_GetURL macro does not set a value for the global macro variable, _MM_CID.

Example

/***********************************************************/
/* Get the URL for the location of a model. */
/***********************************************************/

Options nomlogic nomprint nospool;

/***********************************************************/
/* Get the Model Management macro code. */
/***********************************************************/

FILENAME MMAccess catalog 'sashelp.modelmgr.accessmacros.source';
%include MMAccess;

/***********************************************************/
/* Set to detect failure in case macro load fails */
/* and get the URL. */
/***********************************************************/

%let _MM_RC= -1;

%let target=aef7a78e-0a28-0e97-01c0-b8a0e5ba15c7;
%MM_GetURL(ID=&target,Trace=on);
%put _MM_URL=&_MM_URL;
%put _MM_ResourcesURL=&_MM_ResourcesURL;
%MM_Register Macro

Registers a model to an existing version in the SAS Model Manager model hierarchy.

Syntax

%MM_Register(
    VersionId=destination-version-UUID,
    ModelTemplate=model-template-name,
    EMMModelPackage=fileref-for-EM-package-file,
    ScoreDataStepCode=fileref-to-data-step-fragment-score-code,
    ScoreProgram=fileref-to-SAS-program-score-code,
    InDataSamp=SAS-data-set-reference-to-input-data-sample-table,
    InDataInfo=SAS-data-set-reference-to-input-variable-metadata-table,
    OutDataSamp=SAS-data-set-reference-for-output-data-sample-table,
    OutDataInfo=SAS-data-set-reference-for-output-variable-metadata-table,
    TargetDataSamp=SAS-data-set-reference-for-target-data-sample-table,
    TargetDataInfo=SAS-data-set-reference-for-target-variable-metadata-table,
    TrainingDataSamp=SAS-data-set-reference-for-training-data-sample-table,
    LogisticOutModelTable=SAS-data-set-reference-for-PROC-LOGISTIC-outmodel-table,
    ReportDir=path-to-EMREPORT-directory,
    KeepInVars=keep-variable-list-for-InDataSamp,
    KeepOutVars=keep-variable-list-for-OutDataSamp,
    KeepTargetVars=keep-variable-list-for-TargetDataSamp,
    ModelName=model-name,
    Description=model-description,
    Label=model-label,
    Subject=model-subject,
    Algorithm=model-algorithm,
    Function=model-function,
    Modeler=modeler-property,
    Tool=model-tool-property,
    ToolVersion=model-tool-version,
    <Trace=ON | OFF>
);
Default: the value of the _MM_CId macro variable

Note: This argument is required.

ModelTemplate=MODEL-TEMPLATE-NAME
specifies the SAS Model Manager model template that was used to register and validate this model.

Defaults:
- For models that were registered using the EMMModelPackage parameter, the template is set according to the information that is contained within the named SAS Enterprise Miner model package file.
- Models that were registered using the LogisticOutModelTable parameter are registered with the Classification template.
- All other registrations default to the AnalyticalModel template.

EMModelPackage=SAS-fileref-for-EM-package-file
specifies a SAS file reference that points to the SAS Enterprise Miner model package file (SPK) that contains the model to be registered.

Note: The EMModelPackage argument is required unless you use the ReportDir argument, the ScoreDataStepCode argument, or the ScoreProgram argument to specify the model code filename. If two or more of these arguments are used, the order of priority is ReportDir, EMMModelPackage, ScoreProgram, and then ScoreDataStepCode.

ScoreDataStepCode=fileref-to-data-step-fragment-score-code
specifies a SAS file reference for the model score code that is a fragment of SAS code that can be included in a DATA step. A DATA step fragment contains no DATA, PROC, or RUN statements.

Note: The ScoreDataStepCode argument is required unless you use the EMModelPackage argument, the ReportDir argument, or the ScoreProgram argument to specify the model code filename. If two or more of these arguments are used the order of priority is ReportDir, EMMModelPackage, ScoreProgram, and then ScoreDataStepCode.

ScoreProgram=fileref-to-SAS-program-score-code
specifies a SAS file reference for a text file containing the SAS program, including all step code that is required for successful execution of the model score code.

Note: The ScoreProgram argument is required unless you use the EMModelPackage argument, the ReportDir argument, or the ScoreDataStepCode argument to specify the model code filename. If two or more of these arguments are used the order of priority is ReportDir, EMMModelPackage, ScoreProgram, and then ScoreDataStepCode.

InDataSamp=SAS-data-set-reference-to-input-data-sample-table
specifies a two-level SAS data set reference in the form libref.filename that points to a model input data sample table. The input data sample table is a table that contains all model input variables and is used to create the inputvar.xml file that is required for model registration. The input data sample table is not required for models that were imported as SAS Enterprise Miner package files.

Note: The InDataSamp argument is required unless you use the InDataInfo argument.
Tip When you use the `%MM_Register` macro to register a model, the `inputvar.xml` file should contain only input variables for the model that you are registering. If the input data sample table includes variables that are not used by the model, use the `KeepInVars` argument to remove these variables. If no variables are specified by the `KeepInVars` argument, SAS filters the target variables from the table specified by the `InDataSamp` argument.

See KeepInVars argument on page 28

**InDataSamp**=`SAS-data-set-reference-for-input-data-sample-table`
specifies a two-level SAS data set reference in the form `libref.filename` that points to a model output data sample table. The output data sample table should contain all variables that are created or modified by the model and is used to create the `outputvar.xml` file that is required for model registration. The output data sample table is not required for models that were imported as SAS Enterprise Miner package files.

**Interaction** If the output data sample table includes variables that are created or modified by the model, use the `KeepOutVars` argument to remove these variables. If no variables are specified in the `KeepOutVars` argument, SAS filters the input variables and the target variables from the table that is specified by the `OutDataSamp` argument.

Note The `OutDataSamp` argument must be specified unless you use the `OutDataInfo` argument.

See KeepOutVars argument on page 28

**OutDataInfo**=`SAS-data-set-reference-for-output-variable-metadata-table`
specifies a two-level SAS data set reference in the form `libref.filename` that points to a model output variable metadata table. The output variable metadata table should contain all of the variables that are created or modified by the model. The SAS file should be in the form of the CONTENTS procedure output file, which has the columns `NAME`, `TYPE`, `LENGTH`, `LABEL`, `FORMAT`, `LEVEL`, and `ROLE`. Each row of the table contains a variable. The output variable metadata table is used to create the `outputvar.xml` file that is required for model registration.
Interaction

If no variables are specified by the KeepOutVars argument, SAS filters the input variables and target variables from the table that is specified by the OutDataInfo argument.

Note

The OutDataInfo argument must be specified unless you use the OutDataSamp argument.

TargetDataSamp=\texttt{SAS-data-set-reference-for-target-data-sample-table}

specifies a two-level SAS data set reference in the form \texttt{libref.filename}. The data set reference points to a SAS table that contains the model target variable. The SAS file should contain the variable that was used as the model target during training. The SAS file is used to create the target variable information in the targetvar.xml file that is used for SAS Model Manager model registration.

Tip

If the target data sample table includes other variables that are not model target variables, use the KeepTargetVars argument to remove these variables.

See

KeepTargetVars argument on page 27

TargetDataInfo=\texttt{SAS-data-set-reference-for-target-variable-metadata-table}

specifies a two-level SAS data set reference in the form \texttt{libref.filename}. The data set reference points to a SAS table that contains the model's target variable and its metadata. The SAS file should be in the form of the CONTENTS procedure output file, which has the columns \texttt{NAME}, \texttt{TYPE}, \texttt{LENGTH}, \texttt{LABEL}, \texttt{FORMAT}, \texttt{LEVEL}, and \texttt{ROLE}. Each row of the table contains a variable. The metadata in the SAS file is used to create the target variable information in the target.xml file that is used for SAS Model Manager model registration.

TrainingDataSamp=\texttt{SAS-data-set-reference-for-training-data-sample-table}

specifies a two-level SAS data set reference in the form \texttt{libref.filename}. The data set reference points to a SAS file that contains the training data that is used for a model created by the \texttt{LOGISTIC} procedure. The training data sample must be an exact sample of the training data that is submitted to the \texttt{LOGISTIC} procedure. When the TrainingDataSamp argument and the LogisticOutModelTable argument are specified, the \%MM_Register macro can derive the input, output, and target variables to create the inputvar.xml file, the outputvar.xml file, and the targetvar.xml file.

LogisticOutModelTable=\texttt{SAS-data-set-reference-for-PROC-LOGISTIC-outmodel-table}

specifies a two-level SAS data set reference in the form \texttt{libref.filename} that points to a \texttt{LOGISTIC} procedure fit table that was created by using the \texttt{PROC LOGISTIC OUTMODEL=} statement, and is suitable for use with the \texttt{PROC LOGISTIC INMODEL} statement. If the TrainingDataSamp argument is specified, then SAS generates the input, output, and target variable metadata from this table. In this case, the InDataSamp and OutDataSamp arguments do not need to be specified.

Note

This argument is required only if the model is created by the \texttt{LOGISTIC} procedure using the \texttt{OUTMODEL} statement.

ReportDir=\texttt{path-to-EMREPORT-directory}

specifies an absolute file path to the EMREPORT directory that was created by the SAS Enterprise Miner batch code. All SAS Enterprise Miner model packages that are named miningResult.spk and that reside in a subdirectory of the EMREPORT directory are registered to the target version. The ReportDir argument is valid only for use with SAS Enterprise Miner model package files.

Note

The ReportDir argument is required unless you use the EMModelPackage argument, the ScoreDataStepCode argument, or the ScoreProgram argument.
to specify the model code filename. If two or more of these arguments are used the order of priority is ReportDir, EMMModelPackage, ScoreProgram, and then ScoreDataStepCode.

KeepInVars=\texttt{keep-variable-list-for-InDataSamp}

specifies a list of input variables or columns that are retained in the model's inputvar.xml file. Only variables from the table that is specified by the InDataSamp argument can be specified in this list.

See InDataSamp argument on page 25

KeepOutVars=\texttt{keep-variable-list-for-OutDataSamp}

specifies a list of variables or columns that are retained in the model's outputvar.xml file. Only variables from the table that is specified by the OutDataSamp argument can be specified in this list.

See OutDataSamp argument on page 26

KeepTargetVars=\texttt{keep-variable-list-for-TargetDataSamp}

specifies a list of variables or columns that are retained in the model's targetvar.xml file. Only variables from the tables that are specified by the TargetDataSamp argument can be specified in this list.

See TargetDataSamp argument on page 27

ModelName=\texttt{model-name}

specifies the name of the model, which is used as the value of the model Name property on the General tab of the Models Properties page.

Note This argument is required.

Description=\texttt{model-description}

specifies a description of the model, which is used as the value of the model Description property on the General tab of the Models Properties page.

Label=\texttt{model-label}

specifies a model's label, which is used as the value for the model Model label property on the Specific tab of the Models Properties page. model-label is a text string that is used as the label for the selected model in the model assessment charts that SAS Model Manager creates. If model-label is not specified, SAS Model Manager uses the text string that is specified for the ModelName argument.

Subject=\texttt{model-subject}

specifies the model's subject, which is used as the value for the model Subject property on the Specific tab of the Models Properties page. model-subject provide an additional description for a model, such as a promotional or campaign code. This property is not tied to any computational action by SAS Model Manager.

Algorithm=\texttt{model-algorithm}

specifies the model's computation algorithm, which is used as the value of the model Algorithm property on the Specific tab of the Models Properties page.

Example Algorithm=Decison Tree

Function=\texttt{model-function}

specifies the model's function class, which is used as the value for the model Function on the Specific tab of the Models Properties page. Valid values are Classification, Prediction, Association, Clustering, Sequence, Forecasting, TextMining, Transformation, and EMCreditScoring.
Modeler=model-creator
specifies the SAS Model Manager user ID for the person who created the model, which is used as the value of the model Modeler property on the Specific tab of the Models Properties page.

Tool=model-tool
specifies the modeling tool that was used to create the model, and that is used as the value of the model Tool property on the Specific tab of the Models Properties page.

ToolVersion=model-tool-version
specifies the version of the tool that was used to create the model, and that is used as the value of the model Tool version property on the Specific tab of the Models Properties page.

Trace=ON | OFF
specifies whether to supply verbose trace messages to the SAS log.

Default OFF
Example trace=on

Details

Overview of Using the %MM_Register Macro

The %MM_Register macro registers the following types of models to an existing version in the model repository:

- a model as a SAS Enterprise Miner package
- a SAS DATA step fragment
- a SAS program

In order to register a model using the %MM_Register macro, the macro must know the model name, the version in which the model is registered, the model source code, the model template, and the model input and output variables. If you register a SAS Enterprise Miner model, this information is included in a SAS Enterprise Miner package file (SPK file). When you register SAS code models, you must specify the model name, version, and model score code, as well as the model input and output variables in the respective macro arguments. Several %MM_Register macro arguments enable you to provide values for model property values that appear on the Models Properties page.

Registering SAS Enterprise Miner Models

Models that were created in SAS Enterprise Miner and saved as a SAS Enterprise Miner SPK file contain all of the information that is needed to register a model in SAS Model Manager. Registering SAS Enterprise Miner SPK files requires you to specify the following arguments:

- ModelName
- VersionId
- EMModelPackage or ReportDir arguments

To register one SAS Enterprise Miner model, you can specify the EMModelPackage argument. To register multiple SAS Enterprise Miner models, you use the ReportDir argument to name a directory whose subdirectories each contain a miningResult.spk file. You can register multiple models simultaneously in SAS Model Manager.

SAS Enterprise Miner generates a program, EMBatch, to create multiple models in a batch program. You can modify the EMBatch program to include the %MM_Register
macro, using the macro variable &EMREPORT as the value of the ReportDir argument. By making this change to the EMBatch program, you can create and register SAS Enterprise Miner models in a batch program for use in SAS Model Manager.

**Registering SAS Code Models**

When you register SAS code models, the information that is required is not contained in an SPK file and you must specify the required information using the %MM_Register arguments. Each model that you register must specify the model name, the model version, the model template, the model code, and the SAS data sets that describe the input, output, and target variables.

Use the following table for usage information about using the %MM_Register arguments:

<table>
<thead>
<tr>
<th>Required Information</th>
<th>Argument</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>model name</td>
<td>ModelName</td>
<td>Specify the name of the model, which is used to identify the model in the model repository.</td>
</tr>
<tr>
<td>version</td>
<td>VersionId</td>
<td>Specify the name of the version in which the model is registered.</td>
</tr>
<tr>
<td>model score code</td>
<td>ScoreDataStepCode</td>
<td>Specify a fileref that points to a file that contains score code that is a DATA step fragment. A DATA step fragment contains no DATA, PROC, or RUN statements. When you specify the ScoreDataStepCode argument, your model input and output variables can be defined using one of the following pairs of arguments:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InDataSamp and OutDataSamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InDataInfo and OutDataInfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InDataSamp and OutDataInfo</td>
</tr>
<tr>
<td>Required Information</td>
<td>Argument</td>
<td>Usage</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
|                            | ScoreProgram           | Specify a LOGISTIC procedure FIT table in the form `libref.filename` that was created by the PROC LOGISTIC OUTMODEL= statement. The FIT table can be used as the value in a PROC LOGISTIC INMODEL= statement. When you specify the ScoreProgram argument, your model input and output variables can be defined using one of the following pairs of arguments:  
• InDataSamp and OutDataSamp  
• InDataInfo and OutDataInfo |
|                            | LogisticOutModelTable  | Specify a `libref.filename` that points to a LOGISTIC procedure FIT table that was created by the PROC LOGISTIC OUTMODEL= statement, which can be used as the value to a PROC LOGISTIC INMODEL= statement. If the model does not contain data transmission and you specify a value for the TrainingDataSamp argument, SAS Model Manager uses the training sample data set and the FIT table to create the model inputvar.xml file, the outputvar.xml file, and the targetvar.xml file.  
If you do not specify a value for the TrainingDataSamp argument or if your program transforms the model input before running the LOGISTICS procedure, you must provide the model input and output variables using the InDataSamp or InDataInfo argument, and the OutDataSamp or OutDataInfo argument. |
### Required Information | Argument | Usage
--- | --- | ---
input variables | InDataSamp | Specify a fileref to a SAS data set whose variables contain the input variables that are used by the SAS code model. An example would be a data set that was used for training the model.

SAS Model Manager reads one observation in the data set that is specified by the InDataSamp argument to create the inputvar.xml file for the model. The inputvar.xml file defines the model input variables and their metadata.

Based on the arguments that were specified, the %MM_Register macro uses arguments to filter variables from the data set to create the inputvar.xml file.

- You can use the KeepInVars argument to specify the variables in the InDataSamp data set that are used to create the inputvar.xml file.
- If you do not specify the KeepInVars argument, you can specify a value for the TargetDataSamp argument or the TargetDataInfo argument to filter variables based on this target data sample data set.

For more information, see KeepInVars argument on page 28.
<table>
<thead>
<tr>
<th>Required Information</th>
<th>Argument</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>InDataInfo</strong></td>
<td>Specify a fileref that points to a SAS data set whose variables are NAME, TYPE, LENGTH, LABEL, FORMAT, LEVEL, and ROLE. These variables define metadata for the model input variables. Each row in this data set contains the metadata for model input variables. Such a table can be created by the CONTENTS procedure. SAS Model Manager reads the data set that is specified by the InDataInfo argument to create the inputvar.xml file for the model. The inputvar.xml file defines the model input variables and their metadata. The variables in the data set that are specified by the TargetDataSamp argument or the TargetDataInfo argument are used as a filter to create the inputvar.xml file.</td>
</tr>
<tr>
<td>Required Information</td>
<td>Argument</td>
<td>Usage</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>output variables</td>
<td>OutDataSamp</td>
<td>Specify a fileref that points to a SAS data set whose variables contain the output variables that are created or modified by the SAS code model. An example is a data set that was the scored output of the model. SAS Model Manager reads the data set that is specified by the OutDataSamp argument to create the outputvar.xml file for the model. The outputvar.xml file defines the model output variables and their metadata. Based on the arguments that were specified, the %MM_Register macro uses arguments to filter variables from the data set to create the outputvar.xml file.</td>
</tr>
</tbody>
</table>

• You can use the KeepOutVars argument to specify the variables in the OutDataSamp data set that are used to create the outputvar.xml file. |

• If you do not specify the KeepOutVars argument, input variables and target variables are filtered from the output table. |

For more information, see KeepOutVars argument on page 28.
<table>
<thead>
<tr>
<th>Required Information</th>
<th>Argument</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutDataInfo</td>
<td></td>
<td>Specify a fileref that points to a SAS data set whose variables are NAME, TYPE, LENGTH, LABEL, FORMAT, LEVEL, and ROLE. These variables define metadata for the model output variables. Each row in this data set contains the metadata for model output variables. Such a table can be created by the CONTENTS procedure. SAS Model Manager reads the data set that is specified by the OutDataInfo argument to create the outputvar.xml file for the model. The outputvar.xml file defines the model output variables and their metadata. If you do not specify the KeepOutVars argument, input variables and target variables are filtered from the output table.</td>
</tr>
</tbody>
</table>

| target variable      | TargetDataSamp | Specify a fileref that points to a SAS data set whose variables contain the target variable that is created or modified by the SAS code model. An example is a data set that was the scored output of the model. SAS Model Manager reads the data set that is specified by the TargetDataSamp argument to create the targetvar.xml file for the model. The targetvar.xml file defines the target output variable and its metadata. You can use the KeepTargetVars argument to specify the variable in the TargetDataSamp data set that is used to create the targetvar.xml file. |
### Required Information

<table>
<thead>
<tr>
<th>Required Information</th>
<th>Argument</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TargetDataInfo</td>
<td>Specify a fileref that points to a SAS data set whose variables are NAME, TYPE, LENGTH, LABEL, FORMAT, LEVEL, and ROLE. These variables define metadata for the model target variable. A row in this data set contains the metadata for the model target variable. Such a table can be created by the CONTENTS procedure.</td>
</tr>
</tbody>
</table>

SAS Model Manager reads the data set that is specified by the TargetDataInfo argument to create the targetvar.xml file for the model. The targetvar.xml file defines the model target variable and its metadata.

Use the `%MM_AddModelMfile` macro to register other model component files that are not registered by the `%MM_Register` macro. For more information, see “Model Templates” in *SAS Model Manager: User’s Guide* and “%MM_AddModelFile Macro” on page 16.

### Examples

**Example 1: Registering a SAS Enterprise Miner Model Package**

```sas
Options NOmlogic NOmprint NOspool;

Filename MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;

FILENAME EMPak 'c:\myscorecode\EM\miningResult.spk';
```

/* Set to detect failure in case macro load fails */
/* and register the Enterprise Miner model. */
%let _MM_RC= -1;

%MM_Register(
  VersionId=  
    //ModelManagerModelRepos/MMRoot/HomeEquity/HMEQ/1.0,
  EMModelPackage=EMPak,
  ModelName=HMEQ,
  Description=Home Equity Score Code,
  Trace=ON);

Example 2: Registering a Generic Model

Options nomlogic nomprint nospool;

FILENAME MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;

LIBNAME modelTbl 'c:\myModel\tables';
FILENAME Code 'c:\myModel\scoreCode';

%let _MM_RC= -1;

%MM_Register(
  VersionId=  
    //ModelManagerModelRepos/MMRoot/HomeEquity/HMEQ/1.0,
  ScoreDataStepCode=CODE,
  InDataSamp=modelTbl.HMEQInput,
Example 3: Registering a PROC LOGISTIC OUTMODEL-Style Model

Options nomlogic nomprint nospool;

Filename MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;

LIBNAME modelTbl 'c:\myModel\Tables';
LIBNAME trainTbl 'c:\HomeEquity\Tables';
FILENAME ProgCode 'c:\myModel\scoreCode';

%let _MM_RC= -1;
%MM_Register(
    VersionId= //ModelManagerModelRepos/MMRoot/HomeEquity/HMEQ/1.0,
    ScoreProgram=ProgCODE,
    LogisticOutModelTable=modelTbl.HMEQProcLogisticOutput,
    InDataSamp=trainTbl.HMEQTraining,
    outDataSamp=modelTbl.HMEQOutput,  ModelName=HMEQLogisticOutmodel,
    Description=HMEQ Logistic OUTMODEL model added by macro,
    Trace=off);

OURCES nosource;
%PUT _MM_RC = &_MM_RC;
%PUT _MM_CId = &_MM_CId;
OPTIONS source;

Example 3: Registering a PROC LOGISTIC OUTMODEL-Style Model

Options nosource;
%PUT _MM_RC = &_MM_RC;
%PUT _MM_CId = &_MM_CId;
OPTIONS source;

Example 3: Registering a PROC LOGISTIC OUTMODEL-Style Model

Options nosource;
%PUT _MM_RC = &_MM_RC;
%PUT _MM_CId = &_MM_CId;
OPTIONS source;

Example 3: Registering a PROC LOGISTIC OUTMODEL-Style Model

Options nosource;
%PUT _MM_RC = &_MM_RC;
%PUT _MM_CId = &_MM_CId;
OPTIONS source;
%MM_RegisterByFolder Macro

Register one model or multiple models simultaneously to the model repository from a single directory. Each model is located in a subdirectory under the specified directory.

Syntax

%MM_RegisterByFolder (VersionId=\textit{path-to-version}, ReportDir=\textit{path-to-folder},
<Trace=ON | OFF>);

Arguments

\textit{VersionId=\textit{path-to-version}}

specifies the SAS Model Manager UUID for an existing version in the model repository where the models are registered. \textit{path-to-version} can be either a SAS Model Manager UUID or a version path.

Default the value of the \_MM\_CId macro variable

Note This argument is required.

Examples

\begin{itemize}
\item VersionId=b23327cb-0a29–0c76–011a-f7bb3d790340
\item VersionId=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/ \\
\hspace{1cm} HomeEquity/1.0
\end{itemize}

\textit{ReportDir=\textit{path-to-folder}}

specifies the directory that contains the models to be registered.

Note This argument is required.

\textit{Trace=ON | OFF}

specifies whether to supply verbose trace messages to the SAS log.

Default OFF

Example Trace=on

Details

You can register SAS Enterprise Miner models and SAS code models using the %MM_RegisterByFolder macro. The directory that you specify in the ReportDir argument is the parent folder. Each model has its own subfolder under the parent folder. Each type of model has requirements for the subfolder name and the contents of the subfolder:
Table 2.1  Requirements for Registering Models in a Directory

<table>
<thead>
<tr>
<th>Requirement Type</th>
<th>Enterprise Miner Models</th>
<th>SAS Code Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of ReportDir</td>
<td>a valid directory name</td>
<td>a valid directory name</td>
</tr>
<tr>
<td>Model subdirectory name</td>
<td>the subdirectory name must be the name of the model</td>
<td>the subdirectory name must be the name of the model</td>
</tr>
<tr>
<td>Contents of the subdirectory</td>
<td>one file named miningResult.spk</td>
<td>Required files:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modelmeta.xml</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modelinput.sas7bdat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• score.sas</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional files:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modeloutput.sas7bdat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• target.sas7bdat</td>
</tr>
</tbody>
</table>

Here is a description of the files that reside in the model subfolders:

miningResult.spk
The miningResult.spk file contains the model component files for a model that was created in SAS Enterprise Miner.

modelmeta.xml
The modelmeta.xml file uses XML to define the model component files and values for model properties.

Note: See modelinput.sas7bdat
modelinput.sas7bdat is a table that contains the model input variables. This file is used to create the model inputvar.xml file.

score.sas
score.sas contains the SAS score code, which can be a DATA step fragment or a SAS program.

modeloutput.sas7bdat
modeloutput.sas7bdat is a SAS data set that contains one or more model output variables.

target.sas7bdat
target.sas7bdat is a SAS data set that contains only the target variable.

The modelmeta.xml file is an XML file that is a mapping of SAS Model Manager component filenames to user-defined component filenames. The <Model> element has two main sections:

• <ModelMetadata> to define model properties
• <FileList> to list the model component files. This list is comparable to the Files section of the Local Files window, which you use to import SAS code models.

For a list of files for each model type, see “Model Template Component Files” in SAS Model Manager: User’s Guide.
Within the `<File>` element, put the name of the file that is defined in the model template, in the `<name>` element. The contents of the `<value>` element is the filename under the model directory.

Here is an example `modelmeta.xml` file for a classification model named HMEQ:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Model>
  <ModelMetadata>
    <name>hmeq</name>
    <description>Home Equity Model</description>
    <label>HMEQ</label>
    <algorithm></algorithm>
    <function>classification</function>
    <modeler></modeler>
    <tool>SASProc</tool>
    <toolversion></toolversion>
    <subject></subject>
    <modelTemplate>Classification</modelTemplate>
    <scoreCodeType>SAS Program</scoreCodeType>
  </ModelMetadata>
  <FileList>
    <File>
      <name>score.sas</name>
      <value>myScoreFile.sas</value>
    </File>
    <File>
      <name>modelinput.sas7bdat</name>
      <value>hmeqIn</value>
    </File>
    <File>
      <name>modeloutput.sas7bdat</name>
      <value>hmeqOut</value>
    </File>
    <File>
      <name>target.sas7bdat</name>
      <value>hmeqTar</value>
    </File>
    <File>
      <name>inputvar.xml</name>
      <value></value>
    </File>
    <File>
      <name>outputvar.xml</name>
      <value></value>
    </File>
    <File>
      <name>targetvar.xml</name>
      <value></value>
    </File>
    <File>
      <name>train.sas7bdat</name>
      <value></value>
    </File>
    <File>
      <name>Training.sas</name>
      <value></value>
  </FileList>
</Model>
```
Example Code 1  Registering a Generic Model

/* Register a SAS Code Model By Folder */
/*******************************************************************************/

Options nomlogic nomprint nospool;

/*******************************************************************************/
/* Load and access the Model Management macro code. */
/*******************************************************************************/

Filename MMAccess catalog 'SASHELP.Modelmgr.AccessMacros.source';
%include MMAccess;

/*******************************************************************************/
/* Specify the location of the folder. */
/*******************************************************************************/
%let modelFolder = c:\myModel;
%let hmeq1 = //ModelManagerModelRepos/MMRoot/HomeEquity/HMEQ/1.0;

/***********************************************************/
/* Set to detect failure in case macro load fails */
/* and register the models in the model repository. */
/***********************************************************/

%let _MM_RC= -1;
%MM_RegisterByFolder(VersionId=&hmeq1, ReportDir=&modelFolder, Trace=ON);

/***********************************************************/
/* Display the defined variables. */
/***********************************************************/

Options nosource;
%PUT _MM_RC = &_MM_RC;
Options source;

%MM_CreateModelDataset Macro

Creates a data set that contains information about models. SAS Model Manager can provide information for the champion model or for all models that are in the specified model repository path. The repository path that you specify can be MMRoot, an organizational folder, a project, a version, or a model. The data set contains the information for models that exist under the specified path.

Syntax

%MM_CreateModelDataset (mDatasetName = name-of-data-set,
   smmPath=folder-project-version-or-model-path, <isChampion=Y | N>, Trace=ON | OFF>);

Arguments

mDatasetName = name-of-data-set
   specifies the name of the data set that the macro creates. The macro can be created in a data set that you specify by using a two-level name in the form libref.filename.
   Default mDatasetName=work.models

smmPath=folder-project-version-or-model-path
   specifies the path from which to obtain the model data. If the path is a folder, the data set contains model information for all models under that folder unless isChampion=Y. If isChampion=Y, the information that is returned is for only the champion model. If the path is a project, the data set contains model information for models under that project. If the path is a version, the data set contains model information for models under that version. If the path is a model, the data set contains model information for only that model.
   Default //ModelManager/MMRoot
isChampion=Y | N
specifies whether the information that is returned contains information for only the champion model or for all models.

Y specifies that the information that is returned is for only the champion model.
N specifies that the information that is returned is for all models.

Default Y

Trace=ON | OFF
specifies whether to supply verbose trace messages to the SAS log.

Default OFF

Example Trace=on

Details
By default, the %MM_CreateModelDataset returns data only about the champion model. If you want information about models other than the champion model, specify isChampion=N. The data set that is created contains these variables:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Name</th>
<th>ScoreCodeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreationDate</td>
<td>Owner</td>
<td>Template</td>
</tr>
<tr>
<td>Description</td>
<td>ProductionDate</td>
<td>TemplateFileName</td>
</tr>
<tr>
<td>ExpirationDate</td>
<td>ProjectName</td>
<td>Tool</td>
</tr>
<tr>
<td>FolderName</td>
<td>ProjectPath</td>
<td>UserProperties</td>
</tr>
<tr>
<td>Function</td>
<td>ProjectState</td>
<td>VersionName</td>
</tr>
<tr>
<td>ModelLabel</td>
<td>ProjectURL</td>
<td>VersionState</td>
</tr>
<tr>
<td>ModelUUID</td>
<td>ProjectUUID</td>
<td>isChampion</td>
</tr>
<tr>
<td>Modeler</td>
<td>PublishedDate</td>
<td>isDefaultVersion</td>
</tr>
<tr>
<td>ModificationDate</td>
<td>RetiredDate</td>
<td>isPublished</td>
</tr>
<tr>
<td>ModelUUID</td>
<td>ProjectUUID</td>
<td>isChampion</td>
</tr>
<tr>
<td>Modeler</td>
<td>PublishedDate</td>
<td>isDefaultVersion</td>
</tr>
<tr>
<td>ModificationDate</td>
<td>RetiredDate</td>
<td>isPublished</td>
</tr>
</tbody>
</table>

Example

Example Code 2  Extracting Model Information

```sas
/* Create a data set to contain model information */
Options nomlogic nomprint nospool;

/* Load and access the Model Management macro code. */
Filename MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;

/* Specify the location of the data set and model path. */
```
/**libname modelDS 'c:\myModel\ModelInfo';*/
/* Set to detect failure in case macro load fails */
/* and create the model data set. */
/*libname modelDS 'c:\myModel\ModelInfo';*/
/* Set to detect failure in case macro load fails */
/* and create the model data set. */
%let hmeq1 = //ModelManagerModelRepos/MMRoot/HomeEquity/HMEQ/1.0;
%let _MM_RC= -1;
%MM_CreateModelDataset(mDatasetName=modelDS.models,
              smmpath=//ModelManagerDefaultRepo/MMRoot/DDHMEQ/HMEQ/1.0/Models/
                      Regression,
                      Trace=ON);
/* Display the defined variables. */
/* Display the defined variables. */
Options nosource;
%PUT _MM_RC = &_MM_RC;
Options source;
%MM_CreateModelDataset Macro 45
Chapter 3
Macros for Registering Models to the SAS Metadata Repository

Using Macros to Register Models Not Created by SAS Enterprise Miner

About the %AA_Model_Register Macro

Register a Model in the SAS Metadata Repository Using a SAS/STAT Item Store

Create a SAS Package File Using a SAS/STAT Item Store

Register a Model in the SAS Metadata Repository Using Model Component Files

Dictionary

%AAModel Autocall Macro

%AA_Model_Register Autocall Macro

Using Macros to Register Models Not Created by SAS Enterprise Miner

About the %AA_Model_Register Macro

You can use the %AAModel macro and the %AA_Model_Register macro to register the SAS Metadata Repository models that are not created by SAS Enterprise Miner. These models are created by SAS procedures and are supported by SAS Model Manager:

- SAS/STAT item store models
- High-performance models
- PROC COUNTREG models
- PROC SEVERITY models

If you do not want to register the model, you can create SAS package files (SPK) without registering the model. After the model is registered to the SAS Metadata Repository, you can import the model to SAS Model Manager using the import from SAS Metadata Repository method. If you create an SPK file, you would import the model using the import from SAS Model Package File method.

The %AAModel macro is an autocall macro that loads the %AA_Model_Register macro. This macro must be submitted before you submit the %AA_Model_Register macro.

You specify these types of arguments in the %AA_Model_Register macro:
• The model identification argument’s name. You must also describe a model and identify a SAS/STAT item store.

• Action arguments specify whether to create an SPK file and whether to register the model in the SAS Metadata Repository.

• You specify model component arguments when a SAS/STAT procedure does not create an item store, if a model is created using high-performance analytic procedures, or if you are registering PROC COUNTREG or PROC SEVERITY models. The model component arguments identify the train data set, the model level, and the score code file. The arguments also identify whether the score code is only DATA step code or a SAS program that includes DATA step code, macros, procedures.

• The Lookup=Select option if a SAS/STAT model’s input variable includes non-latin1 characters. This option ensures the generation of correct score code.

• Other options are available to add information to the model or to specify whether to keep or delete the data sets that the macro produces.

For more information, see “%AA_Model_Register Autocall Macro” on page 52.

If you are running macros in a SAS session that requires a connection to the SAS Metadata Server, you must specify the metadata connection system options before you run the macros. If these options are not specified, you will be prompted for the information. If you are running the macros in batch mode, you must include the metadata connection system options. Otherwise, you will receive a failure message. For more information, see “Setting Metadata Connection System Options” on page 2.

These SAS/STAT procedures can create an item store using the STORE statement:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Item Store Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENMOD</td>
<td>Training code is not included</td>
</tr>
<tr>
<td>GLIMMIX</td>
<td>Training code is not included</td>
</tr>
<tr>
<td>GLM</td>
<td>Training code is not included</td>
</tr>
<tr>
<td>GLMSELECT</td>
<td>Fit statistics are not included</td>
</tr>
<tr>
<td>LOGISTIC</td>
<td>None</td>
</tr>
<tr>
<td>MIXED</td>
<td>Training code is not included</td>
</tr>
<tr>
<td>PHREG</td>
<td>Training code is not included</td>
</tr>
<tr>
<td>REG</td>
<td>Training code or fit statistics are not included</td>
</tr>
</tbody>
</table>

Note: Item store restrictions have not been evaluated for other SAS/STAT procedures that have a STORE statement. Using the %AA_Model_Register macro might cause undesirable results.
Register a Model in the SAS Metadata Repository Using a SAS/STAT Item Store

After you run a SAS/STAT procedure using the STORE statement, you use the %AA_Model_Register macro to register the model to the SAS Metadata Repository.

In the following example program, the PROC LOGISTICS STORE statement creates an item store in work.logisticStore. The %AA_Model_Register macro uses the item store in work.logisticStore to create the register file.

```
/* PROC LOGISTIC specifies the STORE statement to create an item store. */
proc logistic data=sampsio.hmeq;
  class job;
  model bad = loan value job;
  store work.logisticStore;
run;

/* Set up the meta data connection system options. */
options metaPort=8561
  metaServer=server-address
  metaRepository=Foundation
  metaUser=user-ID
  metaPass=password;

/* Load the macros. */
%aamodel;

/* Register the model in the SAS Metadata Repository. */
%aa_model_register(modelname=LogisticTest,
  modeldesc=%nrbquote(Logistic Test),
  itemstore=work.logisticstore,
  register=Y,
  mrPath=%NRBQUOTE(/User Folders/user-ID/My Folder/),
  spk=N,
  spkfolder=c:\temp\,
  data=sampsio.hmeq);
```

The model can now be imported to SAS Model Manager using the import from SAS Metadata Repository method.

**Note:** When you are using the SAS Metadata Repository method to import a SAS/STAT item store model into SAS Model Manager, the batch.sas file is empty. Therefore, you cannot retrain the model. To retrain a SAS/STAT item store model, make sure that the model has all of the required files and contents. To avoid missing files or data that is required for retraining a model, create a SAS model package file (SPK) on your local or network drive, and then import it directly into the model repository using SAS Model Manager.
Create a SAS Package File Using a SAS/STAT Item Store

To create a SAS package (SPK) file without registering it to the SAS Metadata Repository, you specify the Register=Y, SPK=Y, and the SPKFOLDER= arguments. This example shows these modifications using the previous example:

```sas
/* PROC LOGISTIC specifies the STORE statement to create an item store. */
proc logistic data=sampsio.hmeq;
   class job;
   model bad = loan value job;
   store work.logisticStore;
run;

/* Set up the meta data connection system options. */
options metaPort=8561
   metaServer=server-address
   metaRepository=Foundation
   metaUser=user-ID
   metaPass=password;

/* Load the macros. */
%aamodel;

/* Create an SPK file; do not register the model in the SAS Metadata Repository. */
%aa_model_register(modelname=LogisticTest,
   modeldesc=&nrbquote(Logistic Test),
   itemstore=work.logisticstore,
   register=N,
   spk=Y,
   spkfolder=c:\temp,
   data=sampsio.hmeq);
```

The macro creates a folder for the model in the `c:\temp` folder. The folder name is the UUID of the model. The name of the SPK file is `miningResults.spk`. The SPK file can be imported to SAS Model Manager using the import from SAS Model Package File method.

Register a Model in the SAS Metadata Repository Using Model Component Files

If you do not have an item store, or if you have the information and files that you need for a model, you can use the `%AA_Model_Register` macro to register the model in the SAS Metadata Repository. In addition to the macro’s model identification arguments and the action arguments, you can use these arguments to register the model:

- Data=`training-data-set-name`
- Level=Binary | Ordinal | Nominal | Interval
- ScoreCodeFile=`filename`
- ScoreCodeFormat=Datastep | Program
Target\text{-}target\text{-}variable

The following SAS program uses model component arguments to register the model to the SAS Metadata Repository. Other arguments identify the mining function and mining algorithm.

/* Train high-performance model */

proc hplogistic data=gplib.hmeqid; class job reason;
    id value;
    class bad ;
    model bad = clage clno debtinc delinq derog mortdue job reason;
    output out=gplib.hpregid_score pred;
    code file='c:\temp\score.sas';
run;

/* Set up metadata connections */

options metaPort=8561
    metaServer=\textit{server-address}
    metaRepository=Foundation
    metaUser=\textit{user-ID}
    metaPass=password;

/* Load the macros. */

%aamodel;

/* Register the model in the SAS Metadata Repository */

%aa_model_register
    (modelname=Model1,
     modeldesc=`\textbackslash nrbquote(First Model for registration),
     register=Y,
     mrPath=`\textbackslash nrbquote(/User Folders/\textit{user-ID}/My Folder/),
     spk=N,
     spkfolder=c:\temp\,
     data=sampsio.hmeq,
     target=bad,
     level=BINARY,
     miningfunction=Classification,
     miningalgorithm=Regression,
     scorecodefile=c:\temp\score.sas)
    ;

The model can now be imported to SAS Model Manager using the import from SAS Metadata Repository method.

---

**Dictionary**

\textbf{\%AAModel Autocall Macro}

Loads the \%AA\_Model\_Register macro.
Syntax

%AAAModel

Details

The %AAAModel macro loads the %AA_Model_Register macro. You must specify %aamodel; before you use the %AA_Model_Register macro. The %AAAModel macro produces these messages in the SAS log:

```
NOTE: Loading the aa_model_eval macro
NOTE: Loading the aa_model_register macro
```

Note: The %AA_Model_Eval macro is used internally by SAS Model Manager.

%AA_Model_Register Autocall Macro

Creates an SPK package file and registers models to the SAS Metadata Repository.

Syntax

%AA_Model_Register(
   ModelName model-name,
   ModelDesc=description,
   Register=Y | N,
   MRPath=SAS-Metadata-Repository- folder,
   SPK=Y | N,
   SPKFolder=SPK-folder-path,
   ItemStore=item-store-name,
   Data=training-data-set-name,
   Target=target-variable,
   Level=Binary | Ordinal | Nominal | Interval,
   ScoreCodeFile=filename,
   ScoreCodeFormat=Datastep | Program,
   <Score=scored-data-set-name>,
   <PMMLFile=filename>,
   <TrainFile=train-program-filename>,
   <MiningAlgorithm=algorithm>,
   <MiningFunction=mining-function>,
   <Segment=segment-variable-name>,
   <Lookup=lookup-method>,
   <Debug=Y | N>)
**Model Identification Arguments**

ModelName=`model-name`

specifies the name of the model.

Default `aa_model_&sysuserid`, where &sysuserid contains the user ID or login of the current SAS process.

ModelDescr=`description`

is a description of the model.

ItemStore=`item-store-name`

specifies the name of the item store that is created by some SAS/STAT procedures. The item store is used to retrieve input and target variable metadata, data set names, score code, training code, the mining algorithm, and the mining function.

Note Item store data is not available from these SAS/STAT procedures: REG, GLM, GENMOD, GLIMMIX, PHREG, and SURVEYPHREG.

Tip If you do not specify the ITEMSTORE= option, you must specify these options: DATA=, TARGET=, SCORECODEFILE=, SCORECODEFORMAT=. If you specify the ITEMSTORE= option, you do not need to specify these options.

**Action Arguments**

Register=`Y | N`

specifies whether to register the model in the SAS Metadata Repository.

Y indicates to register the model in the SAS Metadata Repository.

N indicates not to register the model in the SAS Metadata Repository.

Default `Y`

MRPath=`SAS-Metadata-Repository-Folder`

specifies a folder, using SAS Folders as the root node in the SAS Metadata Repository, where the model is registered.

Default `/Shared Data/

Note The forward slash ( / ) after the last folder in the path is not required.

Example `/Shared Data/Model Manager/Models/`

SPK=`Y | N`

specifies whether to create a SAS package file:

Y indicates to create a SAS package file.

N indicates not to create a SAS package file.

Requirement If SPK=Y, you must use the SPKFOLDER= option to specify a location to store the SPK file.

SPKFOLDER=`SPK-folder-path`

specifies the location to store the SPK file.
Requirement  The option is required when you specify SPK=Y.

Model Component Arguments
These arguments must be specified if you do not specify the ITEMSTORE= option:

Data=training-data-set-name
specifies the name of the training data set for the model.

Level=Binary | Ordinal | Nominal | Interval
specifies the class target level of the model.

  Binary     the variable can contain two discrete values (for example, Yes and No).
  Ordinal    the variable can contain discrete values that have a logical order (for example, 1, 2, 3, 4).
  Nominal    the variable contains discrete values that do not have a logical order (for example, car, truck, bus, and train).
  Interval   the variable contains values across a range. For example, temperature ranges could be between 0–100.

ScoreCodeFile=filename
specifies the name of the file, including the full path to the location where the score code file is stored.

Tip  If you specify the ITEMSTORE= option, you do not need to specify this option.

ScoreCodeFormat=Datastep | Program
specifies the format of the score code.

  DATASTEP   the score code contains only DATA step statements
  PROGRAM    the score code contains DATA step statements, procedures, or macros.

Target=target-variable
specifies the name of the target variable for model.

Optional Arguments

Debug=Y | N
specifies whether to prevent deletion of the generated data sets:

  Y    indicates to keep the generated data sets.
  N    indicates not to keep the generated data sets.

Lookup=lookup-method
specifies the algorithm for looking up CLASS levels in SAS/STAT models. Here are the valid lookup methods:

  Auto
  selects the LINEAR algorithm if a CLASS variable has fewer than five categories. Otherwise, the Binary algorithm is used. This is the default.

  Binary
  specifies to use a binary search. This method is fast, but it might produce incorrect results. The normalized category values might contain characters that
collate in different orders in ASCII and EBCDIC, if you generate the code on an ASCII machine and execute the code on an EBCDIC machine, or vice versa.

Linear
uses a linear search with IF statements that have categories in the order of the class levels. This method is slow if there are many categories.

Select
uses a SELECT statement.

Requirement
Use Lookup=Select when a SAS/STAT model contains non-latin1 characters to ensure the generation of the correct score code. If a model with non-latin1 characters is published to a database and Lookup=Select is not specified, the scoring results might be incorrect.

MiningAlgorithm=algorithm
specifies the type of algorithm that is used to create the model (for example, DecisionTree or logistic).

MiningFunction=mining-function
specifies one of the following mining functions:
- classification
- prediction
- segmentation

PMMLFile=filename
specifies the name of the file that contains the PMML score code, including the full path to the location of the file. This option is optional.

Score=scored-data-set-name
specifies the name of the scored training data set. This data set is used when there is no score code available to determine the output variables or the score code type is not SAS DATA step.

Segment=variable
specifies the name of the segment variable.

TrainFile=train-program-filename
specifies the name of the training program file, including the full path to the location of the file.
Chapter 4
Macros for Adding Folders, Projects, Versions, and Setting Properties

Adding Folders, Projects, Versions, and Properties Using Macros

Overview of Using a SAS Program to Add Folders, Projects, Versions, and Properties
Writing Your SAS Program
Creating the Properties Table

Dictionary
%mdlmgr_AddFolder Macro
%mdlmgr_AddProject Macro
%mdlmgr_AddVersion Macro
%mdlmgr_SetProperty Macro Function

Example: Add a Folder, Project, and Version; Set Properties

Adding Folders, Projects, Versions, and Properties Using Macros

Overview of Using a SAS Program to Add Folders, Projects, Versions, and Properties

Note: You must make sure that you can access the SAS Model Manager web application before using these macros. If you cannot access the web application, contact your SAS application administrator for assistance. For instructions about how to access the web application, see “Sign In” in SAS Model Manager: User’s Guide.

SAS Model Manager provides four macros that you can use in a SAS program to add folders, projects, and versions, and to set properties:

%mdlmgr_AddFolder()
Adds a folder under MMRoot or adds a subfolder.

%mdlmgr_AddProject()
Adds a project under a folder or a subfolder.

%mdlmgr_AddVersion()
Adds a version to a project.

Note: When you use the %mdlmgr_AddProject( ) to create a new project, an initial version is automatically created within the new project. The default version is 1.0. In order to add a new version to an existing project, use the %mdlmgr_AddVersion( ) macro.
%mdlmgr_SetProperty( )

Sets project properties that appear in the **Specific** section of the project **Properties** tab in SAS Model Manager.

After you have added the project objects or set properties, you refresh the folder or project object to see the new objects and property settings in the SAS Model Manager. You can then use these objects in SAS Model Manager to further define your projects and versions.

To delete a folder, project, or version, you use SAS Model Manager.

**Writing Your SAS Program**

Include these language elements in your SAS program:

If you are setting properties, use a DATA step to create a table that contains property and value pairs.

One of the %mdlmgr_SetProperty( ) arguments is the name of a table that contains property-value pairs. “Creating the Properties Table” on page 59 lists the properties that you can include in the table. When you create the table, the first column must be Name and the second column must be Value. Both columns must be character. See “Example: Creating a Properties Table” on page 61.

Access the macros by using the FILENAME and %INCLUDE statements.

```sas
filename file1 catalog 'sashelp.modelmgr.accessmacros.source';
%include file1;
filename file1;

filename file2 catalog 'sashelp.modelmgr.mdlmgr_addfolder.source';
%include file2;
filename file2;

filename file3 catalog 'sashelp.modelmgr.mdlmgr_addproject.source';
%include file3;
filename file3;

filename file4 catalog 'sashelp.modelmgr.logtrace.source';
%include file4;
filename file4;

filename file5 catalog 'sashelp.modelmgr.mdlmgr_addversion.source';
%include file5;
filename file5;

filename file6 catalog 'sashelp.modelmgr.mdlmgr_setproperty.source';
%include file6;
filename file6;

You can change the fileref name.

Call the macros:

```sas
%mdlmgr_AddFolder(ParentId=, Name=, Desc=, NewFolderId=, Trace=);

%mdlmgr_AddProject(ParentId=, Name=, Desc=, ModelFunction=,
                   InputVarTable=, OutputVarTable=, NewProjectId=, Trace=);

%mdlmgr_AddVersion(ParentId=, Desc=, NewVersionId=, Trace=);
```
There is no requirement to call all of the macros in the same SAS program.

When SAS returns from a macro call that adds a node, the value of NewFolderId=, NewProjectId=, and NewVersionId= is used to create a global macro variable that can be referenced by other macros in the same SAS session. The value of the macro variable is the UUID or the model repository path for the node that is added. You can then use that macro reference as a value for the ParentId= argument of another macro or for the %mdlmgr_SetProperty( ) macro FolderId= argument. For example, in the %mdlmgr_AddProject( ) macro, if you set NewProject=projectId, the variable name projectId is used to create the global macro variable &projectId. The &projectId macro reference can now be used as the value of the ParentId= argument in the %mdlmgr_AddVersion( ) macro, ParentId=&projectId. The same macro reference can be used as a value for the FolderId= argument in the %mdlmgr_SetProperty( ) macro, FolderId=&projectId.

If you are running macros in a SAS session that requires a connection to the SAS Metadata Server, you must specify the metadata connection system options before you run the macros. If these options are not specified, you will be prompted for the information. If you are running the macros in batch mode, you must include the metadata connection system options. Otherwise, you will receive a failure message. For more information, see “Setting Metadata Connection System Options” on page 2.

Creating the Properties Table

Property Table Requirements
To set project properties, you use a DATA step to create a data set that contains property-value pairs. The data set variables must be Name and Value, and they must be character variables.

In the data set, property names can be mixed case. The required appended text, :sas-libraries, must be lowercase. For more information, see “Specifying Data Sets” on page 59.

Specifying Data Sets
Some property values specify the name of a default table, such as the default train table or the default performance table. You specify tables using the form SMRLibrary:table for libraries in the SAS Metadata Repository and libref:table for SAS libraries. See the Data Sources category view for valid library and table names. In the SAS Metadata Repository tab, SMRLibrary is the folder-name where the data set is stored. In the SAS Libraries tab, libref can be one of the librefs under the SAS Libraries node.

Properties That You Can Set
Use a property in the following Property Name column as a value for the Name variable in the property table.
### Table 4.1 Project and Version Properties That Can Be Set by `%mdlmgr_SetProperty( )` Macro

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Name As It Appears in the SAS Model Manager</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassificationRole</td>
<td>Output Event Probability Variable</td>
<td>A text string that specifies the output event probability variable. Set for a project with a model function of classification.</td>
</tr>
<tr>
<td>ClassTargetEvent</td>
<td>Class Event Value</td>
<td>A number that represents the target event value. Set for a project.</td>
</tr>
<tr>
<td>ClassTargetEventValues</td>
<td>Class Target Values</td>
<td>A text string that represents the class target values. Set for a project.</td>
</tr>
<tr>
<td>ClassTargetLevel</td>
<td>Class Target Level</td>
<td>One of the following text strings: &quot;BINARY&quot;, &quot;NOMINAL&quot;, &quot;ORDINAL&quot;, or &quot;INTERVAL&quot;. Set for a project.</td>
</tr>
<tr>
<td>ClassTargetVar</td>
<td>Training Target Variable</td>
<td>A text string that indicates the training target variable. Set for a project.</td>
</tr>
<tr>
<td>EventProbabilityRole</td>
<td>Output Event Probability Variable</td>
<td>A text string that specifies the output event probability variable. Specify this property only if you specify the outputVarTable= argument in the <code>%mdlmgr_AddProject( )</code> macro. The value of EventProbabilityRole must be a variable in the project output table. Set for a project.</td>
</tr>
<tr>
<td>Function</td>
<td>Model Function</td>
<td>A text string that specifies the model function. Valid values are &quot;CLASSIFICATION&quot;, &quot;PREDICTION&quot;, &quot;SEGMENTATION&quot;, and &quot;ANALYTICAL&quot;. Set for project,</td>
</tr>
<tr>
<td>InterestedParty</td>
<td>Interested Party</td>
<td>A text string that specifies a person or group that has an interest in the project. Set for a project.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Property Name As It Appears in the SAS Model Manager</td>
<td>Valid Values</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MetadataLock</td>
<td>Lock Project Metadata</td>
<td>Specify &quot;YES&quot; or &quot;NO&quot; to indicate whether the project metadata is locked. Set for a project.</td>
</tr>
<tr>
<td>PredictionRole</td>
<td>Output Prediction Variable</td>
<td>A text string that specifies the output prediction variable. Set for a project with a model function of prediction.</td>
</tr>
<tr>
<td>ScoreInputDS</td>
<td>Default Scoring Input Table</td>
<td>The default scoring test input table in the form <code>libref.table</code>. Set for projects and versions.</td>
</tr>
<tr>
<td>ScoreOutputDS</td>
<td>Default Scoring Output Table</td>
<td>The default scoring test output table in the form <code>libref.table</code>. Set for projects and versions.</td>
</tr>
<tr>
<td>SegmentRole</td>
<td>Output Segmentation Variable</td>
<td>A text string that specifies the output segmentation variable. Set for a project with a model function type of segmentation.</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
<td>Select one:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0  Under Development</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2  Inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3  Retired</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set for a project.</td>
</tr>
<tr>
<td>TestDS</td>
<td>Default Test Table</td>
<td>The default test table in the form <code>libref.table</code>. Set for projects and versions.</td>
</tr>
<tr>
<td>TrainDS</td>
<td>Default Train Table</td>
<td>The default train table in the form <code>libref.table</code>. Set for projects and versions.</td>
</tr>
</tbody>
</table>

**Example: Creating a Properties Table**

Here is a sample DATA step to create a properties table:

```sas
data HMEQProp;
  length name $20;
  length value $40;
  input name value;
  datalines;
```
Dictionary

%mdImgr_AddFolder Macro

Adds a folder to the model repository.

Syntax

%mdImgr_AddFolder(  
  ParentId= parent-UUID-or-path  
  Name= folder-name  
  <Desc= description>  
  NewFolderId= folder-Id-variable  
  <Trace= On | Off>  
);  

Required Arguments

ParentId= parent-UUID-or-path

specifies the UUID or the model repository path of the parent folder.

If the folder that you are creating is a subfolder, you can use the value of
NewFolderId= that was specified during the macro call of parent folder as the value
for parent-UUID-or-path. For example, if a parent folder exists and
NewFolderId=&folderId was set in the macro call for the parent folder, then
you can specify ParentId=&folderId in the subfolder macro call.

If you specify the repository path, use one of these forms:

//ModelManagerDefaultRepo/MMRoot/  
//ModelManagerDefaultRepo/MMRoot/folder-name/

Restriction A folder can be added only to the MMRoot node or a folder in the Projects category.

Name= folder-name

specified the name of the folder. The name can contain letters, spaces, the underscore
( _ ), the hyphen ( - ), and the period ( . ).

NewFolderId= folder-Id-variable

specifies a variable that is used to identify the new folder.
SAS Model Manager creates a global macro variable, &folder-Id-variable, whose value is the folder UUID or the path in the SAS Metadata Repository. You can use &folder-Id-variable as the value of a ParentId= argument in the %mdlmgr_AddFolder() or %mdlmgr_AddProject() macros. For example, if NewFolderId=folderId, then you can use ParentId=(&folderId) in the %mdlmgr_AddProject() macro.

Note: Use SAS macro quoting functions, such as %superq(), to mask double-byte and special characters. Here is an example: ParentId=%superq(folderId).

**Optional Arguments**

**Desc=** *description*

specifies a description of the folder.

**Trace=** On | Off

specifies whether to supply verbose trace messages to the SAS log.

Default Off

---

**%mdlmgr_AddProject Macro**

Adds a project to a folder.

**Note:** When you use the %mdlmgr_AddProject() macro to create a new project, an initial version is automatically created within the new project. The default version is 1.0. To add a new version to an existing project, use the %mdlmgr_AddVersion() macro.

---

**Syntax**

%mdlmgr_AddProject(
    ParentId=parent-UUID
    Name=folder-name
    <Desc=description>
    ModelFunction=model-function
    <InputVarTable=project-input-variable-table>
    <OutputVarTable=project-output-variable-table>
    NewProjectId=project-Id-variable
    <Trace=On | Off>
);

**Required Arguments**

**ParentId=** *parent-UUID-or-path*

specifies the UUID of the parent folder or the model repository path for the parent folder.

You can use &folder-Id-variable that is set for the NewFolderId= argument in the %mdlmgr_AddFolder() macro as the value of parent-UUID-or-path.

The model repository path is in this form:

```//ModelManagerDefaultRepo/MMRoot/folder-name/```
Name=project-name
specified the name of the project. The name can contain letters, spaces, the underscore (_), the hyphen (-), and the period (.).

ModelFunction=model-function
specifies the project model function type. These are the valid values:

- Classification
- Prediction
- Segmentation
- Analytical

Default: Classification

NewProjectId=project-Id-variable
specifies a variable or a macro variable that is used to identify the new project.

SAS Model Manager creates a global macro variable, &project-Id-variable, whose value is the project UUID or the path in the SAS Metadata Repository. You can use &project-Id-variable as the value of a ParentId= argument in the %mdlmgr_AddVersion( ) macro or the FolderId= argument in the %mdlmgr_SetProperty( ) macro. For example, if you set NewProjectId=projectId, you can use ParentId=(&projectId) in the %mdlmgr_AddVersion( ) macro.

Note: Use SAS macro quoting functions, such as %superq(), to mask double-byte and special characters. Here is an example: ParentId=%superq(projectId).

The SAS Model Manager path is in this form:

//ModelManagerDefaultRepo/MMRoot/folder-name/project-name

Optional Arguments

Desc=description
specifies a description of the project.

InputVarTable=project-input-variable-table
specifies a data set that must include the input variables that are used by the champion model. If you have several candidate models for your project, make sure that all candidate model input variables are included in the project input table. The data set does not need to contain data. If you use the train table as a project input table, be sure to exclude the target variable.

The input variable table is used to create the inputvar.xml file, which describes all of the model input variables.

Requirement: The data set must be a local or network file. This macro does not support project input tables in the SAS Metadata Repository.

Tip: The project input table can be defined after the project is created. It must be defined before the project champion model is set.

See: “Create a Project Input Table” in SAS Model Manager: User’s Guide

OutputVarTable=project-output-variable-table
specifies a data set that includes only output variables that are created or modified by the champion model. If you have several candidate models for your project, you
must make sure that all project output variables are mapped to the champion model output variables. If you use the train table as the project output table, use the SET statement to specify the training table, and use the KEEP statement to specify the variables from the training table that you want in the project output table.

The output variable table is used to create the outputvar.xml file, which describes all of the model output variables.

**Requirement**  
The data set must be a local or network file. This macro does not support project output tables in the SAS Metadata Repository.

**Tip**  
The project output table can be defined after the project is created. It must be defined before the project champion model is set.

**See**  
“Create a Project Output Table” in SAS Model Manager: User’s Guide

**Trace=On | Off**  
specifies whether to supply verbose trace messages to the SAS log.

**Default**  
Off

---

**%mdlmgr_AddVersion Macro**

Adds a version to a project.

**Note:**  
The **Name=version-name** argument was deprecated with the release of SAS Model Manager 13.1. It is no longer supported in SAS Model Manager 13.1 and later. If you have migrated from a previous release, the value of the version name appears in the **Description** field. You must remove this argument from your existing macro code to avoid errors. When you use the %mdlmgr_AddProject( ) macro to create a new project, an initial version is automatically created within the new project. In order to add a new version to an existing project, use the %mdlmgr_AddVersion( ) macro.

**Syntax**

```sas
%mdlmgr_AddVersion(
   ParentId=parent-UUID-or-path
   <Desc=description>
   NewVersionId=version-Id-variable
   <Trace=On | Off>
);   
```

**Required Arguments**

**ParentId=parent-UUID-or-path**  
specifies the UUID of the project for which the version is to be created.

You can use **&project-Id-variable** that is set for the NewProjectId= argument in the %mdlmgr_AddProject( ) macro as the value of **parent-UUID-or-path**. For example, if **NewProjectId=projectId**, you can specify **ParentId=(&projectId)**.
Note: Use SAS macro quoting functions, such as %superq(), to mask double-byte and special characters. Here is an example:

```
ParentId=%superq(projectId)
```

The SAS Model Manager path is in the form

```
//ModelManagerDefaultRepo/MMRoot/folder-name/project-name
```

**NewVersionId=version-Id-variable**

specifies a variable name that is used to identify the new version.

SAS Model Manager creates a global macro variable, &version-Id-variable, whose value is the version UUID or the path in the SAS Metadata Repository. You can use &version-Id-variable as the value of the FolderId= argument in the %mdlmgr_SetProperty( ) macro. For example, if you set NewVersionId=versionId, then you can specify FolderId=(&versionId) in the %mdlmgr_SetProperty( ) macro.

Note: Use SAS macro quoting functions, such as %superq(), to mask double-byte and special characters. Here is an example:

```
FolderId=%superq(versionId)
```

The version path is in this form:

```
//ModelManagerDefaultRepo/MMRoot/folder-name/project-name/version-name
```

**Optional Arguments**

- **Desc=description**
  
specifies a description of the version.

- **Trace=Off | On**
  
specifies whether to supply verbose trace messages to the SAS log.

  Default: Off

---

**%mdlmgr_SetProperty Macro Function**

Sets project properties in the model repository

**Syntax**

```
%mdlmgr_SetProperty(
  FolderId=folder-UUID-or-path
  Table=property-value-table-name
  PropertyType=System | User
  FolderType=UUID-or-folder-type
  <Trace=On | Off>
)
```

**Required Arguments**

- **FolderId=folder-UUID-or-path**
  
specifies the project folder UUID or path.

  To add a project property, you can use &project-Id-variable that is set for the NewProjectId= argument in the %mdlmgr_AddProject( ) macro as the value of
To add a version property, you can use &version-Id-variable that is set for the NewVersionId= argument in the %mdlMgr_AddVersion( ) macro as the value of project-folder-UUID-or-path. For example, if NewVersionId=versionId, then you can specify FolderId=&versionId.

Note: Use SAS macro quoting functions, such as %superq(), to mask double-byte and special characters. Here is an example: FolderId=%superq(versionId).

Table=property-value-data-set
specifies the data set that contain the properties to set. property-value-table-name must be in the form libref.data-set.

See “Creating the Properties Table” on page 59

PropertyType=System | User
specifies whether the property is a SAS Model Manager property or if the property is user-defined. Specify system for all SAS Model Manager properties.

Default System

FolderType=folder-type
specifies the folder type for the properties that are being set. If FolderId is a UUID, this argument is optional. The valid value for folder-type is Project.

Optional Argument

Trace=On | Off
specifies whether to supply verbose trace messages to the SAS log.

Default Off

Example: Add a Folder, Project, and Version; Set Properties

libname temp 'your-path';
data temp.property;
  length name $ 30 value $ 40;
  input name value;
  infile datalines;
datalines;
ScoreInputDS MMLIB.HMEQ_SCORE_INPUT
ScoreOutputDS MMLIB.HMEQ_SCORE_OUTPUT
TrainDS MMLIB.HMEQ_TRAIN
TestDS MMLIB.HMEQ_TEST
ClassTargetEvent 1
ClassTargetLevel BINARY
ClassTargetVar BAD
EventProbabilityRole SCORE
;
run;

/* Access the macros */

filename file1 catalog 'sashelp.modelmgr.accessmacros.source';
%include file1;
filename file1;

filename file2 catalog 'sashelp.modelmgr.mdlmgr_addfolder.source';
%include file2;
filename file2;

filename file3 catalog 'sashelp.modelmgr.mdlmgr_addproject.source';
%include file3;
filename file3;

filename file4 catalog 'sashelp.modelmgr.logtrace.source';
%include file4;
filename file4;

filename file5 catalog 'sashelp.modelmgr.mdlmgr_addversion.source';
%include file5;
filename file5;

filename file6 catalog 'sashelp.modelmgr.mdlmgr_setproperty.source';
%include file6;
filename file6

/*add folder*/
%mdlmgr_AddFolder( parentId=//ModelManagerDefaultRepo/MMRoot,
                   name=Bank3,
                   desc=,
                   newFolderId=newFolderIdVar,
                   Trace=on);

/*add project*/
%mdlmgr_AddProject( parentId=%superq(newFolderIdVar),
                    name=HMEQ,
                    desc=Home Equity,
                    modelFunction=classification,
                    inputVarTable=,
                    outputVarTable=,
                    newProjectId=newProjectIdVar1,
                    Trace=on); 

/*set properties*/
%mdlmgr_SetProperty( folderId=%superq(newProjectIdVar1),
                   table=temp.property,
                   propertyType=system,
                   folderType=Project,
                   Trace=on);

/*add version*/
%mdlmgr_AddVersion( parentId=%superq(newProjectIdVar1),
                    desc=,
                    newVersionId=newVersionIdVar1,
                    Trace=off);
Chapter 5

Macros for Generating Score Code

Generating Score Code for COUNTREG Procedure Models .......................... 69
Generating Score Code for SEVERITY Procedure Models .......................... 70
Dictionary ................................................................. 70
%MM_Countreg_Create_Scorecode Autocall Macro .................................. 70
%MM_Severity_Create_Scorecode Autocall Macro .................................. 84

Generating Score Code for COUNTREG Procedure Models

The %MM_Countreg_Create_Scorecode macro creates DATA step statements to compute the predicted values of a model that you create using the COUNTREG procedure. Input to the macro is the ODS output data set ParameterEstimates that is created by the COUNTREG procedure. You can also specify the location to save the score code and other macro output files. You can specify a location for prefix values for the dependent variable and the variable for the probability of having a zero-generating process.

Note: SAS Model Manager does not support PROC COUNTREG models when VALIDVARNAME="ANY".

The score code generation supports the following COUNTREG procedure features:

<table>
<thead>
<tr>
<th>PROC COUNTREG Feature</th>
<th>Supported Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorical predictor</td>
<td>Character and numeric class variables</td>
</tr>
<tr>
<td>Continuous predictor</td>
<td>Variable values are used as is.</td>
</tr>
<tr>
<td>MODEL specification</td>
<td>Effect specifications that are allowed by the MODEL statement, including main effects, interactions, and powers of continuous predictors. Only one MODEL statement can be specified.</td>
</tr>
<tr>
<td>ZEROMODEL specification</td>
<td>Effect specifications that are allowed in the MODEL statement, including the intercept, main effects, interactions, and powers of continuous predictors.</td>
</tr>
</tbody>
</table>


## PROC COUNTREG Feature

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET variables</td>
<td>The offset variables in the MODEL and ZEROMODEL statements are retrieved from the FitSummary table.</td>
</tr>
<tr>
<td>ZEROMODEL statement LINK function</td>
<td>The LOGISTIC and NORMAL link distribution functions that are allowed in the ZEROMODEL statement.</td>
</tr>
</tbody>
</table>

BY-group processing is not supported. For more information, see the “COUNTREG Procedure” in *SAS/ETS User’s Guide*.

After you have created the score code, you can register the score code and other COUNTREG procedure model component files by using the %AA_Model_Register macro or you can import the model using the local files method. For more information, see “Using Macros to Register Models Not Created by SAS Enterprise Miner” on page 47 and “Import Models from Local Files” in *SAS Model Manager: User’s Guide*.

### Generating Score Code for SEVERITY Procedure Models

The %MM_Severity_Create_Scorecode macro generates score code for PROC SEVERITY models. Inputs to the macro are the ODS output data sets ParameterEstimates and ModelInformation that are created by the SEVERITY procedure. You can also specify the location to save the score code and other macro output files, and the prefix value for the dependent variable.

Custom distributions and BY-group processing are not supported by the macro. For more information, see the “SEVERITY Procedure” in *SAS/ETS User’s Guide*.

After you have created the score code, you can register the score code and other SEVERITY procedure model component files by using the %AA_Model_Register macro or you can import the model using the local files method. For more information, see “Using Macros to Register Models Not Created by SAS Enterprise Miner” on page 47 and “Import Models from Local Files” in *SAS Model Manager: User’s Guide*.

### Dictionary

**%MM_Countreg_Create_Scorecode Autocall Macro**

Generates score code for a model that is created by the COUNTREG procedure.

**Syntax**

```
%MM_Countreg_Create_Scorecode (ParamEst=countreg-parameter-estimate-data-set <FileRef=output-fileref> <PredPrefix=dependent-variable-prefix> <PZPrefix=probability-zero-variable-prefix>)
```
Arguments

ParamEst=countreg-parameter-estimate-dataset
specifies the name of the parameter estimations ODS output data. This ParameterEstimates data set is created when PROC COUNTREG executes. To capture this data set, use the ODS OUTPUT statement before PROC COUNTREG executes.

Tip In the PROC COUNTREG code, include the PREDICTION= and PREOBZERO= options in the OUTPUT statement.

FileRef=output-fileref
specifies the fileref that defines the location of the macro output files.

Default The SAS log

PredPrefix=dependent-variable-prefix
specifies a prefix for the predicted dependent variable. The variable is named in the PRED option of the PROC COUNTREG OUTPUT= statement. When this prefix is applied to the dependent variable, this new name becomes the prediction variable.

Default P_

PZPrefix=probability-zero-variable-prefix
specifies a prefix for a particular variable. This variable indicates the probability that the response variable will take on the value of zero as a result of the zero-generating process. The variable is named in the PROBZERO option of the PROC COUNTREG OUTPUT= statement. When the prefix is applied to the probability zero variable, this new name becomes the probability zero variable.

Default PHI_

Details

To create score code for a model that you create with PROC COUNTREG, include the following SAS code:

1. Use a LIBNAME statement to identify the location of the output that you create using PROC COUNTREG.

2. Before PROC COUNTREG, use the ODS OUTPUT statement to capture the ParameterEstimates output data set. Here is an example:

   ods output ParameterEstimates=CntReg.ParameterEstimates;

3. Build your model using PROC COUNTREG and close the ODS OUTPUT destination.

4. Use the FILENAME statement to define a fileref for the macro output location.

5. Invoke the %mm_countreg_create_scorecode macro.

6. Execute the score code within a DATA step.
Example: Generate the PROC COUNTREG Score Code for Insurance Risk

Create the Sample Insurance Data

The following SAS program creates sample data that resembles an automobile policy history file for a property and casualty insurance program:

```sas
%let MyProj = C:\Users\sasdemo;
libname CntReg "&MyProj.\CountReg\Test";
options fmtsearch = (CntReg.formats);
proc format library = CntReg cntlout = phf_fmt;
  value $ Gender_fmt
    'Male' = 'Man'
    'Female' = 'Woman';
  value HO_fmt
    0 = 'No'
    1 = 'Yes';
run;

data CntReg.phf;
  length CarType $ 5;
  label CarType = 'Type of Car';
  length Gender $ 6;
  format Gender $ Gender_fmt.;
  label Gender = 'Gender Identification';
  /* This variable name will test how the macro will resolve name conflicts */
  length Estimate $ 6;
  label Estimate = 'Gender Identification (Copy)';
  label AgeDriver = 'Driver Age';
  format HomeOwner HO_fmt.;
  call streaminit(27513);

  do PolicyId = 00001 to 00099;
    StartYr = 2000 +
      rand('table', 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1);
    do ExpYr = StartYr to 2011;
      EExp = rand('uniform');
      MyOffset = 0;
      select (rand('table', 0.499, 0.299, 0.199, 0.003));
        when (1)
          do;
            CarType = 'SEDAN';
            fCarType = 0;
            end;
        when (2)
          do;
            CarType = 'TRUCK';
            fCarType = 0.5;
            end;
        when (3)
          do;
            CarType = 'OTHER';
            fCarType = 1;
            end;
    end;
  end;
run;
```

Chapter 5 • Macros for Generating Score Code
do;
   CarType = 'SPORT';
   fCarType = 1.0;
end;
otherwise CarType = ' ';
end;

AgeDriver = 18 + rand('binomial',0.375, 72);
fAgeDriver = 0.0123 * (AgeDriver - 17);

HomeOwner = rand('bernoulli', 0.25);
if (HomeOwner eq 0) then fHomeOwner = 0.7;
   else if (HomeOwner eq 1) then fHomeOwner = 0;
if (HomeOwner eq 1) then
do;
   IS = round(rand('uniform') * 5) - 2.5;
   fIS = -0.0456 * IS * IS;
end;
if (EExp lt 0.5) then
do;
   Gender = 'Male';
   fGender = 0;
end;
else if (EExp lt 0.9) then
do;
   Gender = 'Female';
   fGender = -1.5;
end;
   else Gender = ' '; 
Estimate = Gender;
if (missing(HomeOwner) eq 0 and missing(IS) eq 0)
   then mu_zero = 0.987 + fHomeOwner + fIS;
   else mu_zero = 0.987;
phi = cdf('normal', mu_zero, 0, 1);
if (rand('bernoulli', phi) eq 0) then
do;
   if (missing(CarType) eq 0 and missing(AgeDriver) eq 0 and
      missing(Gender) eq 0)
      then mu = 2 + fCarType + fAgeDriver + fGender;
      else mu = 2;
   nClaim = rand('poisson', exp(mu));
end;
else nClaim = 0;
output;
end;
end;
drop fCarType fAgeDriver fHomeOwner fGender;
drop mu_zero mu;
run;

Run the Sample Program

Here is the sample program:

%let MyProj = C:\Users\sasdemo;
libname CntReg "&MyProj.\CountReg\Test";
options fmtsearch = (CntReg.formats);

/* Original Model */
%let model = 1;

/* Build the model and deliver the required ODS datasets */
ods output ParameterEstimates = CntReg.ParameterEstimates_&model.;
ods output close;

proc countreg data = CntReg.phf;
class CarType Gender HomeOwner;
model nClaim = CarType AgeDriver Gender / dist = poisson;
zeromodel nClaim ~ HomeOwner IS * IS / link = normal;
output out = CntReg.phf_pred_&model.
predicted = Pred_nClaim probzero = Phi_nClaim;
run;

/* Define the fileref for the output syntax */
filename ThisFile "&MyProj.CountReg.Test\ScoreCode_&Model..sas";

/* Invoke the macro */
%mm_countreg_create_scorecode(
   ParamEst = CntReg.ParameterEstimates_&Model.,
   FileRef = ThisFile,
   PredPrefix = MyPred_,
   PZPrefix = MyPhi_
);

/* Execute the score codes within a DATA STEP */
data CntReg.phf_pred_compare;
set CntReg.phf_pred_&Model.;
%include ThisFile;
IsMiss_Pred_nClaim = missing(Pred_nClaim);
IsMiss_Phi_nClaim = missing(Phi_nClaim);
IsMiss_MyPred_nClaim = missing(MyPred_nClaim);
IsMiss_MyPhi_nClaim = missing(MyPhi_nClaim);
if (IsMiss_Pred_nClaim eq 0 and IsMiss_MyPred_nClaim eq 0)
   then MyDiffPred = MyPred_nClaim - Pred_nClaim;
if (IsMiss_Phi_nClaim eq 0 and IsMiss_MyPhi_nClaim eq 0)
   then MyDiffPhi = MyPhi_nClaim - Phi_nClaim;
run;

proc contents data = CntReg.phf_pred_compare;
run;

/* If the score codes work correctly, then the MyDifference variable should be a constant variable of all zero values */
proc freq data = CntReg.phf_pred_compare;
tables _WARN_;
run;

proc tabulate data = CntReg.phf_pred_compare;
class IsMiss_Pred_nClaim IsMiss_MyPred_nClaim IsMiss_Phi_nClaim IsMiss_MyPhi_nClaim;
Example: Generate the PROC COUNTREG Score Code for Insurance Risk

```plaintext
var Pred_nClaim MyPred_nClaim MyDiffPred Phi_nClaim
    MyPhi_nClaim MyDiffPhi;
table IsMiss_Pred_nClaim * IsMiss_MyPred_nClaim *
    (n nmiss mean*format=e22. stddev*format=e22. min*format=e22. max*format=e22.),
    (Pred_nClaim MyPred_nClaim MyDiffPred);
table IsMiss_Phi_nClaim * IsMiss_MyPhi_nClaim *
    (n nmiss mean*format=e22. stddev*format=e22. min*format=e22. max*format=e22.),
    (Phi_nClaim MyPhi_nClaim MyDiffPhi);
run;
quit;
```
The Generated Score Code and Output Tables

Output 5.1 Generated Score Code

```sas
/* Begin scoring code for COUNTREG */
/* Model: ZIP */
/* Created By: sasdemo */
/* Date: April 26, 2013 */
/* Time: 09:27:39 */

LENGTH _WARN_ $ 4;
_WARN_ = '    '; LABEL _WARN_ = "Warnings";
_nInputMiss = 0;

LENGTH _UFormat_1 $ 5 ;
LABEL _UFormat_1 = "Formatted Value of CarType";
IF ( MISSING( CarType ) EQ 0 ) THEN DO;
   _UFormat_1 = STRIP( PUT( CarType , $5. ) );
   IF ( _UFormat_1 NOTIN ( "SEDAN", "SPORT", "TRUCK") ) THEN _nInputOutRange = _nInputOutRange + 1;
END;
ELSE _nInputMiss = _nInputMiss + 1;

LENGTH _UFormat_2 $ 5 ;
LABEL _UFormat_2 = "Formatted Value of Gender";
IF ( MISSING( Gender ) EQ 0 ) THEN DO;
   _UFormat_2 = STRIP( PUT( Gender , $GENDER_FMT5. ) );
   IF ( _UFormat_2 NOTIN ( "Man", "Woman") ) THEN _nInputOutRange = _nInputOutRange + 1;
END;
ELSE _nInputMiss = _nInputMiss + 1;
```
Example: Generate the PROC COUNTREG Score Code for Insurance Risk

```plaintext
LENGTH _UFormat_3 $ 3 ;
LABEL _UFormat_3 = "Formatted Value of HomeOwner" ;
IF ( MISSING( HomeOwner ) EQ 0 ) THEN DO;
   _UFormat_3 = STRIP( PUT( HomeOwner , HO_FMT3. ) ) ;
   IF ( _UFormat_3
      NOTIN ( "No" , "Yes" )
   ) THEN _nInputOutOfRange = _nInputOutOfRange + 1;
END;
ELSE _nInputMiss = _nInputMiss + 1;

/******************************************************************************
/* Set _WARN_ value                                                          */
/******************************************************************************

_VALID2SCORE = 1;
LABEL _VALID2SCORE = "Is this record valid to be scored? 1=Yes, 0=No" ;
IF ( _nInputMiss GT 0 ) THEN DO;
   SUBSTR(_WARN_,1,1) = 'M';
   _VALID2SCORE = 0;
END;
IF ( _nInputOutOfRange GT 0 ) THEN DO;
   SUBSTR(_WARN_,2,1) = 'U';
   _VALID2SCORE = 0;
END;

/******************************************************************************
/* Calculate scores only if current record contains valid values            */
/******************************************************************************

IF ( _VALID2SCORE EQ 1 ) THEN DO;
   _NU_MODEL = 0 ;
   _NU_ZEROMODEL = 0 ;
   _NU_MODEL = _NU_MODEL + 7.889048183464800E-01
   ;
   IF ( _UFormat_1 EQ "SEDAN" ) THEN DO;
      _NU_MODEL = _NU_MODEL - 4.983426513164500E-01
   END;
   IF ( _UFormat_1 EQ "SPORT" ) THEN DO;
      _NU_MODEL = _NU_MODEL + 4.985885591940500E-01
   END;
   _NU_MODEL = _NU_MODEL + 1.227923016048900E-02 * AgeDriver
   ;
   IF ( _UFormat_2 EQ "Man" ) THEN DO;
      _NU_MODEL = _NU_MODEL + 1.503894036936300E+00
   END;
   _NU_ZEROMODEL = _NU_ZEROMODEL + 9.925866013120000E-01
   ;
```

IF ( _UFormat_3 EQ "No" ) THEN DO;
    _NU_ZEROMODEL = _NU_ZEROMODEL + 6.905739218180000E-01
    ;
END;

_NU_ZEROMODEL = _NU_ZEROMODEL - 4.346588113784800E-02
* IS
* IS
;
_LOG_TAIL_P_ = LOGSDF( 'NORMAL', _NU_ZEROMODEL );

IF ( (_NU_MODEL + _LOG_TAIL_P_) LE 709.780 )
THEN MyPred_nClaim = EXP( _NU_MODEL + _LOG_TAIL_P_ );
ELSE MyPred_nClaim = .;

MyPhi_nClaim = 1 - EXP( _LOG_TAIL_P_ );

END; /* END (_VALID2SCORE EQ 1) IF BLOCK */

LABEL MyPred_nClaim = &quot;Predicted value of nClaim&quot; ;
LABEL MyPhi_nClaim = &quot;Probability of nClaim being zero as a result of the zero-generating process&quot; ;

DROP _nInputMiss _VALID2SCORE _NU_MODEL;
DROP _NU_ZEROMODEL _LOG_TAIL_P_ ;
DROP _nInputOutRange
    _UFormat_1
    _UFormat_2
    _UFormat_3
    ;

/******************************************************************************/
/* End scoring code for COUNTREG                                          */
/******************************************************************************/

Output 5.2  The Tables Created by the Sample Program

The SAS System

The COUNTREG Procedure

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CarType</td>
<td>3</td>
<td>SEDAN SPORT TRUCK</td>
</tr>
<tr>
<td>Gender</td>
<td>2</td>
<td>Man Woman</td>
</tr>
<tr>
<td>HomeOwner</td>
<td>2</td>
<td>No Yes</td>
</tr>
</tbody>
</table>
### Model Fit Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dependent Variable</strong></td>
<td>nClaim</td>
</tr>
<tr>
<td><strong>Number of Observations</strong></td>
<td>582162</td>
</tr>
<tr>
<td><strong>Missing Values</strong></td>
<td>67258</td>
</tr>
<tr>
<td><strong>Data Set</strong></td>
<td>CNTREG.PHF</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>ZIP</td>
</tr>
<tr>
<td><strong>ZI Link Function</strong></td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Log Likelihood</strong></td>
<td>-283522</td>
</tr>
<tr>
<td><strong>Maximum Absolute Gradient</strong></td>
<td>0.00229</td>
</tr>
<tr>
<td><strong>Number of Iterations</strong></td>
<td>9</td>
</tr>
<tr>
<td><strong>Optimization Method</strong></td>
<td>Newton-Raphson</td>
</tr>
<tr>
<td><strong>AIC</strong></td>
<td>567066</td>
</tr>
<tr>
<td><strong>SBC</strong></td>
<td>567190</td>
</tr>
</tbody>
</table>

### The SAS System

**The COUNTREG Procedure**

#### Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CarType</td>
<td>3</td>
<td>SEDAN SPORT TRUCK</td>
</tr>
<tr>
<td>Gender</td>
<td>2</td>
<td>Man Woman</td>
</tr>
<tr>
<td>HomeOwner</td>
<td>2</td>
<td>No Yes</td>
</tr>
<tr>
<td>Parameter</td>
<td>DF</td>
<td>Estimate</td>
</tr>
<tr>
<td>-----------------</td>
<td>----</td>
<td>----------</td>
</tr>
<tr>
<td>Intercept</td>
<td>1</td>
<td>0.788905</td>
</tr>
<tr>
<td>CarType SEDAN</td>
<td>1</td>
<td>-0.498343</td>
</tr>
<tr>
<td>CarType SPORT</td>
<td>1</td>
<td>0.498589</td>
</tr>
<tr>
<td>CarType TRUCK</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AgeDriver</td>
<td>1</td>
<td>0.012279</td>
</tr>
<tr>
<td>Gender Man</td>
<td>1</td>
<td>1.503894</td>
</tr>
<tr>
<td>Gender Woman</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Infl_Intercept</td>
<td>1</td>
<td>0.992587</td>
</tr>
<tr>
<td>Infl_HomeOwner No</td>
<td>1</td>
<td>0.690574</td>
</tr>
<tr>
<td>Infl_HomeOwner Yes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Infl_IS*IS</td>
<td>1</td>
<td>-0.043466</td>
</tr>
</tbody>
</table>
**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th><em>LABEL</em></th>
<th>_VALUE_1</th>
<th>_VALUE_2</th>
<th>_VALUE_3</th>
<th>MacVar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CARTYPE</td>
<td>Type of Car</td>
<td>SEDAN</td>
<td>SPORT</td>
<td>TRUCK</td>
<td>CarType</td>
</tr>
</tbody>
</table>

**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th><em>LABEL</em></th>
<th>_VALUE_1</th>
<th>_VALUE_2</th>
<th>MacVar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GENDER</td>
<td>Gender Identification</td>
<td>Man</td>
<td>Woman</td>
<td>Gender</td>
</tr>
</tbody>
</table>

**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>_VALUE_1</th>
<th>_VALUE_2</th>
<th>MacVar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HOMEOWNER</td>
<td>No</td>
<td>Yes</td>
<td>HomeOwner</td>
</tr>
</tbody>
</table>

**The SAS System**

**The CONTENTS Procedure**

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>CNTREG.PHF_PRED_COMPARE</th>
<th>Observations</th>
<th>649420</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>25</td>
</tr>
<tr>
<td>Engine</td>
<td>V9</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>04/25/2013 16:49:55</td>
<td>Observation Length</td>
<td>192</td>
</tr>
<tr>
<td>Last Modified</td>
<td>04/25/2013 16:49:55</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representaion</td>
<td>WINDOWS_64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>wlatin1 Western (Windows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Engine/Host Dependent information

<table>
<thead>
<tr>
<th>Data Set Page Size</th>
<th>65536</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Data Set Pages</td>
<td>1911</td>
</tr>
<tr>
<td>First Data Page</td>
<td>1</td>
</tr>
<tr>
<td>Max Obs per Page</td>
<td>340</td>
</tr>
<tr>
<td>Obs in First Data Page</td>
<td>321</td>
</tr>
<tr>
<td>Number of Data Set Repairs</td>
<td>0</td>
</tr>
<tr>
<td>ExtendObsCounter</td>
<td>YES</td>
</tr>
<tr>
<td>Filename</td>
<td>C:\Users\emdev\CountReg\Test\phf_pred_compare.sas7bdat</td>
</tr>
<tr>
<td>Release Created</td>
<td>9.040 B0</td>
</tr>
<tr>
<td>Host Created</td>
<td>X64_S08R2</td>
</tr>
</tbody>
</table>

### Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>AgeDriver</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Driver Age</td>
</tr>
<tr>
<td>3</td>
<td>CarType</td>
<td>Char</td>
<td>5</td>
<td></td>
<td>Type of Car</td>
</tr>
<tr>
<td>11</td>
<td>EEExp</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Estimate</td>
<td>Char</td>
<td>6</td>
<td></td>
<td>Gender Identification (Copy)</td>
</tr>
<tr>
<td>10</td>
<td>ExpYr</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Gender</td>
<td>Char</td>
<td>6</td>
<td>$GENDER_FMT</td>
<td>Gender Identification</td>
</tr>
<tr>
<td>7</td>
<td>HomeOwner</td>
<td>Num</td>
<td>8</td>
<td>HO_FMT.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>IS</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>isMiss_MyPhi_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>isMiss_MyPred_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>isMiss_Phi_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>isMiss_Pred_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>MyOffPhl</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>MyOffPred</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MyOffset</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>MyPhi_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Probability of nClaim being zero as a result of the zero-generating process</td>
</tr>
<tr>
<td>18</td>
<td>MyPred_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted value of nClaim</td>
</tr>
<tr>
<td>2</td>
<td>Phi_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Probability of nClaim being zero</td>
</tr>
<tr>
<td>8</td>
<td>Polidy</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pred_nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted value of nClaim</td>
</tr>
<tr>
<td>9</td>
<td>StartYr</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td><em>WARN</em></td>
<td>Char</td>
<td>4</td>
<td></td>
<td>Warnings</td>
</tr>
<tr>
<td>14</td>
<td>IS</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>nClaim</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>phi</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### SAS System

**The FREQ Procedure**

#### Warnings

<table>
<thead>
<tr>
<th><em>WARN</em></th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>67258</td>
<td>100.00</td>
<td>67258</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Frequency Missing = 582162

#### Example: Generate the PROC COUNTREG Score Code for Insurance Risk

<table>
<thead>
<tr>
<th>isMiss_Pred_nClaim</th>
<th>isMiss_MyPred_nClaim</th>
<th>Predicted value of nClaim</th>
<th>Predicted value of nClaim</th>
<th>MyDiffPred</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>N 582162</td>
<td>582162</td>
<td>582162</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NMISS 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mean 9.4293059680972000E-01</td>
<td>9.4293059680971000E-01</td>
<td>-2.49508170328000E-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StdDev 1.05495527856579700E-01</td>
<td>1.06495527855579700E-00</td>
<td>4.181909199995600E-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min 9.1238483847418000E-02</td>
<td>9.1289483847418000E-02</td>
<td>-3.437259484239400E-13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max 7.925944704397700E+00</td>
<td>7.925944704397300E+00</td>
<td>3.063113317171900E-16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>N 67258</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NMISS 0</td>
<td>67258</td>
<td>67258</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mean 3.8624649068790000E-01</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StdDev 3.1935249858749000E-01</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min 9.1238483847418000E-02</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max 4.5254937353080000E-00</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

#### Example: Generate the PROC COUNTREG Score Code for Insurance Risk

<table>
<thead>
<tr>
<th>isMiss_Phi_nClaim</th>
<th>isMiss_MyPhi_nClaim</th>
<th>Probability of nClaim being zero</th>
<th>Probability of nClaim being zero as a result of the zero-generating process</th>
<th>MyDiffPhi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>N 582162</td>
<td>582162</td>
<td>582162</td>
</tr>
<tr>
<td></td>
<td>NMISS 0</td>
<td>Mean 9.1044998735180000E-01</td>
<td>9.1044998735180000E-01</td>
<td>-8.4367925621642000E-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StdDev 5.8386749053129000E-02</td>
<td>5.8386749053129000E-02</td>
<td>3.8958040902956700E-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min 7.6452212186512000E-01</td>
<td>7.6452212186512000E-01</td>
<td>-1.4432899320127000E-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max 9.5276824099900000E-01</td>
<td>9.5276824099900000E-01</td>
<td>-2.2244404925300000E-16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>N 67258</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>NMISS 0</td>
<td>Mean 9.1059340418704000E-01</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StdDev 5.8116724461392000E-02</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min 7.6452212186512000E-01</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max 9.5276824099900000E-01</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
%MM_Severity_Create_Scorecode Autocall Macro

Creates DATA step statements to compute the predicted values of a model that you create using the SEVERITY procedure.

Syntax

%MM_Severity_Create_Scorecode (  
   ParamEst=severity-parameter-estimate-data-set  
   ModelInfo=model-info-data-set<FileRef=output-fileref>  
   <PredPrefix=dependent-variable-prefix>  
) / store secure;

Arguments

ParamEst=severity-parameter-estimate-dataset  
 specifies the name of the parameter estimations output data. This data set is created when you specify the OUTTEST= option in the PROC SEVERITY statement.

ModelInfo=model-info-data-set  
 specifies the name of the model information output data set. This data set is created when you specify the OUTMODELINFO= option in the PROC SEVERITY statement.

FileRef=output-fileref  
 specifies the fileref that defines the location of the macro output files.

   Default The SAS log

PredPrefix=dependent-variable-prefix  
 specifies a prefix for the predicted dependent variable. The variable is named in the PROC SEVERITY LOSS= statement. When this prefix is applied to the dependent variable, this new name becomes the prediction variable.

   Default P_

Details

To create score code for a model that you create with PROC SEVERITY, include the following SAS code:

1. Use a LIBNAME statement to identify the location of the output that you create using PROC SEVERITY.
2. Build your model using PROC SEVERITY. Specify the OUTTEST= option to create the ParameterEstimates data. Specify OUTMODELINFO= option to create the ModelInformation data set. Close the ODS OUTPUT destination.
3. Use the FILENAME statement to define a fileref for the macro output location.
4. Invoke the %MM_Severity_Create_Scorecode Macro.
Create the Sample Insurance Data

%let MyProj = C:\Users\sasdemo;
libname Severity "&MyProj.\Severity\Test";

data Severity.phf;

   /* Regression Coefficient for the Intercept Term */
   retain fIntercept  6.8024;

   /* Regression Coefficient for continuous AgeDriver */
   retain fAgeDriver  0.01234;

   /* Regression Coefficient for the three dummy indicators for nominal CarType */
   retain fCarType_SEDAN  0;
   retain fCarType_SPORT  1.0;
   retain fCarType_TRUCK  0.5;

   /* Regression Coefficient for the two dummy indicators for nominal Gender */
   retain fGender_Female -1.5;
   retain fGender_Male  0;

   /* Regression Coefficient for the two dummy indicators for nominal HomeOwner */
   retain fHomeOwner_NO  0;
   retain fHomeOwner_YES  0.7;

   /* Regression Coefficient for continuous IS */
   retain fIS  -0.00456;

   /* Regression Coefficient for continuous MileageDriven */
   retain fMileageDriven  0.013579;

   /* Variable Labels */
   label AgeDriver = 'Age of Driver';

   label AmountLoss = 'Amount of Loss in Dollars';
   format AmountLoss dollar.;

   label CarType_SEDAN = 'Indicator of Car Type is Sedan';
   label CarType_SPORT = 'Indicator of Car Type is Sport';
   label CarType_TRUCK = 'Indicator of Car Type is Truck';

   label EExp = 'Earned Exposure in Units of One Year';

   label ExpYr = 'Exposure Year';

   label Gender_Female = 'Indicator of Gender is Female';
   label Gender_Male = 'Indicator of Gender is Male';
label HomeOwner_NO = 'Indicator of Home Ownership is No';
label HomeOwner_YES = 'Indicator of Home Ownership is Yes';

label IS = 'Insurance Score of Driver';
label MileageDriven = 'Mileage Driven in Units of 1,000 Miles';
label PolicyId = 'Policy Identifier';

call streaminit(27513);
do PolicyId = 00001 to 00099;
    StartYr = 2000 +
        rand('table', 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1);
do ExpYr = StartYr to 2011;
    EExp = rand('uniform');
    AgeDriver = 18 + rand('binomial', 0.375, 72);

    CarType_SEDAN = 0;
    CarType_SPORT = 0;
    CarType_TRUCK = 0;
    select (rand('table', 0.4999, 0.2999, 0.1999, 0.0003));
        when (1) CarType_SEDAN = 1;
        when (2) CarType_SPORT = 1;
        when (3) CarType_TRUCK = 1;
        otherwise
do;
        CarType_SEDAN = .;
        CarType_SPORT = .;
        CarType_TRUCK = .;
end;
end;

    Gender_Female = 0;
    Gender_Male = 0;
    if (EExp lt 0.4999) then Gender_Female = 1;
    else if (EExp lt 0.9999) then Gender_Male = 1;
    else
do;
        Gender_Female = .;
        Gender_Male = .;
end;
end;

    HomeOwner_NO = 0;
    HomeOwner_YES = 0;
    if (rand('bernoulli', 0.25) eq 1) then HomeOwner_YES = 1;
    else HomeOwner_NO = 1;

    IS = round(rand('gamma', 600));
    if (IS gt 800) then IS = 800;
    else if (IS lt 1) then IS = 1;

    MileageDriven = rand('gamma', 12);
    /* Annual Mileage Driven in unit of 1000 miles */
    if (nmiss(MileageDriven, AgeDriver, CarType_SEDAN, CarType_TRUCK,
CarType_SPORT,
Gender_Male, Gender_Female,
HomeOwner.YES, HomeOwner.NO, IS eq 0)
then
do;
mu = fIntercept
+ fAgeDriver * (28 - AgeDriver)
+ fCarType_SEDAN * CarType_SEDAN + fCarType_SPORT
+ CarType_SPORT
+ fCarType_TRUCK * CarType_TRUCK
+ fGender_Female * Gender_Female + fGender_Male
+ Gender_Male
+ fHomeOwner_NO * HomeOwner_NO + fHomeOwner.YES
+ HomeOwner.YES
+ fIS * IS
+ fMileageDriven * (MileageDriven - 12);
AmountLoss = exp(mu) * rand('gamma', 25);
end;
else AmountLoss = .;
output;
end;
drop fAgeDriver fCarType_SEDAN fCarType_TRUCK fCarType_SPORT fGender_Male
fGender_Female fHomeOwner.YES fHomeOwner.NO fIntercept fIS fMileageDriven;
run;

Run the Sample Program

%let MyProj = C:\\Users\\sasdemo;
libname Severity "&MyProj.\\Severity\\Test";
title "SCALEMODEL and all applicable distributions";
%let model = 1;
%let predlist = AgeDriver CarType_SEDAN CarType_TRUCK CarType_SPORT
   Gender_Male Gender_Female HomeOwner.YES HomeOwner.NO IS MileageDriven;
/* Build the model and obtain the required datasets */
proc severity data = Severity.phf
   outest = Severity.ParamEst_&Model.
   outmodelinfo = Severity.ModelInfo_&model.
   loss AmountLoss;
dist _predefined_ stweedie;
scalemodel &predlist.;
nloptions maxiter = 1000;
run;
/* Define the fileref for the output syntax */
filename ThisFile "&MyProj.\\Severity\\Test\\ScoreCode_&Model..sas";
/* Invoke the macro */
%mm_severity_create_scorecode
   ParamEst = Severity.ParamEst_&Model.,
   ModelInfo = Severity.ModelInfo_&model.,
The Generated Score Code and Output Tables

Output 5.3 Generated Score Code

```sas
LENGTH _WARN_ $ 4;
_WARN_ = '    '; LABEL _WARN_ = "Warnings" ;

/* Check the SCALEMODEL regression variables */
IF ( MISSING( MileageDriven ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
IF ( MISSING( IS ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
/* NOTE: HomeOwner_NO is not checked for missing values because it is a redundant regressor. */
IF ( MISSING( HomeOwner_YES ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
/* NOTE: Gender_Female is not checked for missing values because it is a redundant regressor. */
IF ( MISSING( Gender_Male ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
IF ( MISSING( CarType_SPORT ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
IF ( MISSING( CarType_TRUCK ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
/* NOTE: CarType_SEDAN is not checked for missing values because it is a redundant regressor. */
IF ( MISSING( AgeDriver ) EQ 1 ) THEN _nInputMiss = _nInputMiss + 1;
```
Example: Generate the PROC SEVERITY Score Code for Insurance Risk

```plaintext
/**********************************************
/* Calculate scores only if current record contains valid values */
***********************************************/

IF (_nInputMiss EQ 0 ) THEN DO;

/**********************************************
/* Distribution: BURR */
***********************************************/

  _XBETA_ = 0;
  _XBETA_ = _XBETA_ + 1.401229993940100E-02 * MileageDriven ;  
  _XBETA_ = _XBETA_ - 4.744055757460300E-03 * IS ;  
  /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 7.381815343018200E-01 * HomeOwner_YES ;
  /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 1.495958484730900E+00 * Gender_Male ;
  _XBETA_ = _XBETA_ + 7.317848636754000E-01 * CarType_TRUCK ;
  /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ - 1.066440552950800E-02 * AgeDriver ;

  MyPred_AmountLoss_BURR = 6.236121394142300E+03 * EXP(_XBETA_);

/**********************************************
/* Distribution: EXP */
***********************************************/

  _XBETA_ = 0;
  _XBETA_ = _XBETA_ + 1.364239152005100E-02 * MileageDriven ;
  _XBETA_ = _XBETA_ - 4.739639251429400E-03 * IS ;
  /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 7.307690977221500E-01 * HomeOwner_YES ;
  /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 1.496025385765200E+00 * Gender_Male ;
  _XBETA_ = _XBETA_ + 4.644249484904200E-01 * CarType_TRUCK ;
  /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ - 1.151946645256000E-02 * AgeDriver ;

  MyPred_AmountLoss_EXP = 6.506747031895200E+03 * EXP(_XBETA_);

/**********************************************
/* Distribution: GAMMA */
***********************************************/

  _XBETA_ = 0;
  _XBETA_ = _XBETA_ + 1.364239152005100E-02 * MileageDriven ;
  _XBETA_ = _XBETA_ - 4.739639251429400E-03 * IS ;
  /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 7.307690978471100E-01 * HomeOwner_YES ;
  /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ + 1.496025385874300E+00 * Gender_Male ;
  _XBETA_ = _XBETA_ + 4.644249491258800E-01 * CarType_TRUCK ;
  /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
  _XBETA_ = _XBETA_ - 1.151946645256000E-02 * AgeDriver ;

  MyPred_AmountLoss_GAMMA = 6.506745978302000E+03 * EXP(_XBETA_);
```

_XBETA_ = 0;
 XBETA = _XBETA_ + 1.364260675458400E-02 * MileageDriven;
 XBETA = _XBETA_ - 4.739190610741700E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ + 7.307694223592400E-01 * HomeOwner_YES;
 /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ + 1.496025631070000E+00 * Gender_Male;
 XBETA = _XBETA_ + 9.166441449252000E-01 * CarType_SPORT;
 XBETA = _XBETA_ + 4.644265557308700E-01 * CarType_TRUCK;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ - 1.151818427565500E-02 * AgeDriver;

MyPred_AmountLoss_GPD = 6.504569551883800E+03 * EXP(_XBETA_);

_XBETA_ = 0;
 XBETA = _XBETA_ + 1.375427253970500E-02 * MileageDriven;
 XBETA = _XBETA_ - 4.808982977719000E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ + 1.499838197531700E+00 * Gender_Male;
 XBETA = _XBETA_ + 9.940256355577800E-01 * CarType_SPORT;
 XBETA = _XBETA_ + 4.644584176107000E-01 * CarType_TRUCK;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ - 1.161363856490700E-02 * AgeDriver;

MyPred_AmountLoss_IGAUSS = 6.790140708730100E+03 * EXP(_XBETA_);

_XBETA_ = 0;
 XBETA = _XBETA_ + 1.379535910756300E-02 * MileageDriven;
 XBETA = _XBETA_ - 4.808040604328300E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ + 7.286190149538200E-01 * HomeOwner_YES;
 /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ + 9.356227862952000E-01 * CarType_SPORT;
 XBETA = _XBETA_ + 4.647870916071300E-01 * CarType_TRUCK;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA = _XBETA_ - 1.152580789026700E-02 * AgeDriver;

MyPred_AmountLoss_LOGN = 6.759869565247100E+03 * EXP(_XBETA_);
Example: Generate the PROC SEVERITY Score Code for Insurance Risk

```plaintext
/* Distribution: PARETO */

 XBETA_ = 0;
 XBETA_ = XBETA_ + 1.365643731959700E-02 * MileageDriven;
 XBETA_ = XBETA_ - 4.710463064928600E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 7.307900535342100E-01 * HomeOwner_YES;
 /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 9.918130012553400E-01 * CarType_SPORT;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ - 1.143601522019100E-02 * AgeDriver;

 MyPred_AmountLoss_PARETO = 6.367949496138600E+03 * EXP(XBETA_);

/* Distribution: STWEEDIE */

 XBETA_ = 0;
 XBETA_ = XBETA_ + 1.346890420267100E-02 * MileageDriven;
 XBETA_ = XBETA_ - 4.675720482528600E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 7.318380293889600E-01 * HomeOwner_YES;
 /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 1.493775242549600E+00 * Gender_Male;
 XBETA_ = XBETA_ + 9.908288084354000E-01 * CarType_SPORT;
 XBETA_ = XBETA_ + 4.641286573822800E-01 * CarType_TRUCK;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ - 1.156384352057300E-02 * AgeDriver;

 MyPred_AmountLoss_STWEEDIE = 6.293220261934300E+03 * EXP(XBETA_);

/* Distribution: WEIBULL */

 XBETA_ = 0;
 XBETA_ = XBETA_ + 1.282907583324000E-02 * MileageDriven;
 XBETA_ = XBETA_ - 4.480140052950500E-03 * IS;
 /* NOTE: HomeOwner_NO is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 7.297752425496000E-01 * HomeOwner_YES;
 /* NOTE: Gender_Female is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ + 9.931730126153400E-01 * CarType_SPORT;
 XBETA_ = XBETA_ + 4.546576968650700E-01 * CarType_TRUCK;
 /* NOTE: CarType_SEDAN is skipped because it is a redundant regressor. */
 XBETA_ = XBETA_ - 1.211172101144000E-02 * AgeDriver;

 MyPred_AmountLoss_WEIBULL = 5.778646806752700E+03 * EXP(XBETA_);

END;
```
ELSE DO;
   /*******************************************************************************/
   /* Set _WARN_ value */
   /*******************************************************************************/
   SUBSTR(_WARN_,1,1) = 'N';
END;

LABEL MyPred_AmountLoss_BURR = "Predicted mean for the Burr Distribution" ;
LABEL MyPred_AmountLoss_EXP = "Predicted mean for the Exponential Distribution" ;
LABEL MyPred_AmountLoss_GAMMA = "Predicted mean for the Gamma Distribution" ;
LABEL MyPred_AmountLoss_GPD = "Predicted mean for the Generalized Pareto Distribution" ;
LABEL MyPred_AmountLoss_IGAUSS = "Predicted mean for the Inverse Gaussian Distribution" ;
LABEL MyPred_AmountLoss_LOGN = "Predicted mean for the Lognormal Distribution" ;
LABEL MyPred_AmountLoss_PARETO = "Predicted mean for the Pareto Distribution" ;
LABEL MyPred_AmountLoss_STWEEDIE = "Predicted mean for the Tweedie Distribution with Scale Parameter" ;
LABEL MyPred_AmountLoss_WEIBULL = "Predicted mean for the Weibull Distribution" ;
DROP _nInputMiss _XBETA_;
/*******************************************************************************/
/* Rnd scoring code for SEVERITY */
/*******************************************************************************/

The following tables are a sampling of the output tables that are created by the example. For each distribution type, PROC SEVERITY creates these tables: Distribution
Information, Convergence Status, Optimization Summary, Fit Statistics, and Parameterization Estimation. The output displays the tables for the stweedie distribution.

**SCALEMODEL and all applicable distributions**

The SEVERITY Procedure

<table>
<thead>
<tr>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>SEVERITY.PHF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Converged</td>
</tr>
<tr>
<td>-2 Log Likelihood</td>
</tr>
<tr>
<td>Selected</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>stweedie</td>
</tr>
<tr>
<td>Burr</td>
</tr>
<tr>
<td>Exp</td>
</tr>
<tr>
<td>Gamma</td>
</tr>
<tr>
<td>Igauss</td>
</tr>
<tr>
<td>Logn</td>
</tr>
<tr>
<td>Pareto</td>
</tr>
<tr>
<td>Gpd</td>
</tr>
<tr>
<td>Weibull</td>
</tr>
</tbody>
</table>
**SCALEMODEL and all applicable distributions**

**The SEVERITY Procedure**  
stweedie Distribution

<table>
<thead>
<tr>
<th>Distribution Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>stweedie</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Tweedie Distribution with Scale Parameter</td>
</tr>
<tr>
<td><strong>Distribution Parameters</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Regression Parameters</strong></td>
<td>7</td>
</tr>
</tbody>
</table>

**Convergence Status**  
Convergence criterion (GCONV=1E-8) satisfied.

**Optimization Summary**  
<table>
<thead>
<tr>
<th>Optimization Technique</th>
<th>Trust Region</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iterations</strong></td>
<td>9</td>
</tr>
<tr>
<td><strong>Function Calls</strong></td>
<td>37</td>
</tr>
<tr>
<td><strong>Log Likelihood</strong></td>
<td>-4197.9</td>
</tr>
</tbody>
</table>

**Fit Statistics**  
| -2 Log Likelihood | 8396 |
| AIC               | 8416 |
| AICC              | 8416 |
| BIC               | 8460 |
| Kolmogorov-Smirnov| 0.33978 |
| Anderson-Darling  | 6448 |
| Cramer-von Mises  | 26.70330 |
### Parameter Estimates

| Parameter        | Estimate | Standard Error | t Value | Approx Pr > |t| |
|------------------|----------|----------------|---------|--------------|---|
| Theta            | 9.18984  | 18.75328       | 0.49    | 0.6243       |   |
| Lambda           | 24.39485 |                |         |              |   |
| P                | 1.03440  | 0.06739        | 15.35   | <.0001       |   |
| AgeDriver        | -0.01166 | 0.00203        | -5.71   | <.0001       |   |
| CarType_TRUCK    | 0.46413  | 0.02295        | 20.23   | <.0001       |   |
| CarType_SPORT    | 0.99083  | 0.01879        | 52.74   | <.0001       |   |
| Gender_Male      | 1.49378  | 0.01658        | 90.10   | <.0001       |   |
| HomeOwner_YES    | 0.73184  | 0.01911        | 38.30   | <.0001       |   |
| IS               | -0.00468 | 0.0003363      | -13.90  | <.0001       |   |
| MileageDriven    | 0.01347  | 0.00239        | 5.62    | <.0001       |   |

### SCALEMODEL and all applicable distributions

The `CONTENTS` Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>SEVERITY.PHF_WPREDICTION</th>
<th>Observations</th>
<th>625</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>24</td>
</tr>
<tr>
<td>Engine</td>
<td>V9</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>05/30/2015 23:33:10</td>
<td>Observation Length</td>
<td>192</td>
</tr>
<tr>
<td>Last Modified</td>
<td>05/30/2015 23:33:10</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>WINDOWS_64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>utf8 Unicode (UTF-8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engine/Host Dependent Information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Set Page Size</strong></td>
<td>65536</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Number of Data Set Pages</strong></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>First Data Page</strong></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max Obs per Page</strong></td>
<td>340</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Obs in First Data Page</strong></td>
<td>310</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Number of Data Set Repairs</strong></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ExtendObsCounter</strong></td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Filename</strong></td>
<td>C:\sasdemo\Severity\Test\phf_wprediction.sas?bdat</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Release Created</strong></td>
<td>9 0401M3</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Host Created</strong></td>
<td>X64_S08R2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Generate the PROC SEVERITY Score Code for Insurance Risk

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AgeDriver</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Age of Driver</td>
</tr>
<tr>
<td>2</td>
<td>AmountLoss</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR.</td>
<td>Amount of Loss in Dollars</td>
</tr>
<tr>
<td>3</td>
<td>CarType_SEDAN</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Car Type is Sedan</td>
</tr>
<tr>
<td>4</td>
<td>CarType_SPORT</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Car Type is Sport</td>
</tr>
<tr>
<td>5</td>
<td>CarType_TRUCK</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Car Type is Truck</td>
</tr>
<tr>
<td>6</td>
<td>EExp</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Earned Exposure in Units of One Year</td>
</tr>
<tr>
<td>7</td>
<td>ExpYr</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Exposure Year</td>
</tr>
<tr>
<td>8</td>
<td>Gender_Female</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Gender is Female</td>
</tr>
<tr>
<td>9</td>
<td>Gender_Male</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Gender Is Male</td>
</tr>
<tr>
<td>10</td>
<td>HomeOwner_NO</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Home Ownership is No</td>
</tr>
<tr>
<td>11</td>
<td>HomeOwner.YES</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Indicator of Home Ownership is Yes</td>
</tr>
<tr>
<td>12</td>
<td>IS</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Insurance Score of Driver</td>
</tr>
<tr>
<td>13</td>
<td>MileageDriven</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Mileage Driven In Units of 1,000 Miles</td>
</tr>
<tr>
<td>16</td>
<td>MyPred_AmountLoss_BURR</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Burr Distribution</td>
</tr>
<tr>
<td>17</td>
<td>MyPred_AmountLoss_EXP</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Exponential Distribution</td>
</tr>
<tr>
<td>18</td>
<td>MyPred_AmountLoss_GAMMA</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Gamma Distribution</td>
</tr>
<tr>
<td>19</td>
<td>MyPred_AmountLoss_GPD</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Generalized Pareto Distribution</td>
</tr>
<tr>
<td>20</td>
<td>MyPred_AmountLoss_IGAUSS</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Inverse Gaussian Distribution</td>
</tr>
<tr>
<td>21</td>
<td>MyPred_AmountLoss_LOGN</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Lognormal Distribution</td>
</tr>
<tr>
<td>22</td>
<td>MyPred_AmountLoss_PARETO</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Pareto Distribution</td>
</tr>
<tr>
<td>23</td>
<td>MyPred_AmountLoss_TWEEDIE</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Tweedie Distribution with Scale Parameter</td>
</tr>
<tr>
<td>24</td>
<td>MyPred_AmountLoss_WEIBULL</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Predicted mean for the Weibull Distribution</td>
</tr>
<tr>
<td>14</td>
<td>PolicyId</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Policy Identifier</td>
</tr>
<tr>
<td>15</td>
<td><em>WARN</em></td>
<td>Char</td>
<td>4</td>
<td></td>
<td>Warnings</td>
</tr>
</tbody>
</table>
Chapter 6
Macros for Creating Performance Reports in Batch

Overview of SAS Programs to Monitor Model Performance .................................. 100
Prerequisites for Running Batch Performance Reports ................................. 101
  Overview of Prerequisites for Running Batch Performance Reports ............ 101
  Publish the Champion Model from the Project Folder ............................... 101
  Create a Folder Structure ................................................................. 101
  Obtain Performance Data ................................................................. 103
  Determine the Publish Channel ......................................................... 103
  Copy Example Batch Programs ......................................................... 104
  Determine SAS Model Manager User ID and Password ........................... 104
Report Output in Test and Production Modes .............................................. 104
  Report Output in Test Mode .............................................................. 104
  Report Output in Production Mode ..................................................... 105
Define the Report Specifications ............................................................... 106
  Overview of Code to Define Report Specifications .................................. 106
  Required Libref .................................................................................. 106
  Project Specifications ................................................................. 106
  Email Recipient Specifications .......................................................... 109
  Report Specifications ................................................................. 110
  Job Scheduling Specifications .......................................................... 113
  Example Code to Create the Report Specifications .................................. 114
Extracting the Champion Model from a Channel .......................................... 117
  Using the %MM_GetModels() Macro .................................................. 117
  Accessing Model Management Report Macros ....................................... 118
  %MM_GetModels() Macro Syntax ...................................................... 118
  Example Program to Extract a Model from a Channel ......................... 118
  The current.sas7bdat Data Set ......................................................... 119
SAS Code to Run Performance Reports ....................................................... 120
  Overview of the SAS Code to Run the Performance Reports ................. 120
  Accessing Model Management Report Macros ....................................... 121
  Required Librefs .............................................................................. 121
  Macro Variables to Define Report Local Folders and Data Sets .......... 121
  Macro Variables That Are Used by the %MM_RunReports() Macro ....... 122
  The DATA Step to Access the Performance Data Set ............................ 123
  The %MM_RunReports() Macro ....................................................... 123
  Example Code to Run the Reports ..................................................... 125
Overview of SAS Programs to Monitor Model Performance

A SAS program that creates performance monitoring reports consists of three conceptual sections:

- The first section defines the report specifications that identify the project, the types of reports that you want to create, alert and warning conditions, and the date and time to run the batch jobs.
- The second section extracts the champion model from a publishing channel. Any batch job that creates performance monitoring reports must extract models from a publishing channel. The champion model must have been published to the channel from the project folder.
- The third section defines the operating environment and the performance data set. This section calls a SAS macro that creates the reports.

*Note:* SAS programs for performance monitoring reports can be run only for champion models. Performance monitoring reports for challenger models can be run only by creating a performance definition using SAS Model Manager.

You define the report specifications by writing four DATA steps:

- `mm_jobs.project` defines the project specifications.
- `mm_jobs.emailaddr` defines the email address where you send job, alert, and warning notifications.
- `mm_jobs.reportdef` defines which type of reports you want to create, and the alert and warning conditions for those reports.
- `mm_jobs.jobtime` defines the date and time to run the batch jobs.

After the report specification data sets have been created, you extract the champion model from the publishing channel to the SAS Model Manager server using the `%MM_GetModels()` macro. You set macro variables to define the operating environment, specify the performance data set, and run the `%MM_RunReports()` macro to create the reports.

You view the reports by selecting the project’s **Performance** page in SAS Model Manager. The reports are saved at the version level.

SAS Model Manager provides the following performance monitoring macros:

- `%MM_GetModels()` extracts models from a publishing channel.
- `%MM_UpdateCharacteristicTable` creates a Characteristic report.
- `%MM_UpdateStabilityTable` creates a Stability report.
- `%MM_UpdateAssessmentTable` creates model monitoring reports.
- `%MM_RunReports()` sets the operating environment and runs the macros to create the reports.

*Note:* The macros are in the sashelp.modelmgr catalog. The location of this catalog for Windows is `\sasinter\SASFoundation\9.4\mmcommon\sashelp`. The default value for `sasinstalldir` in Windows is `C:\Program Files \SASHome`. The location of this file for UNIX is `/sasinstalldir/`.
SAS provides example SAS programs in the sashelp.modelmgr catalog that you can modify for your environment.

Prerequisites for Running Batch Performance Reports

Overview of Prerequisites for Running Batch Performance Reports

Batch performance reporting requires you to complete several tasks before you can modify the example programs. After the following tasks have been completed, you are ready to modify the example programs:

• Determine the channel that is used to publish the project champion model.

• Ensure that the champion model has been published from the project folder to a channel.

• Create a folder structure on the SAS Model Manager server.

  Note: The folder on the SAS Model Manager server that is used in creating batch performance reports must be accessible to the batch performance program.

• Store performance data sets on the SAS Model Manager server.

• If you are using SAS example programs, copy the example programs to the SAS Model Manager server.

• Determine a SAS Model Manager user ID and password to authorize the batch processing.

Publish the Champion Model from the Project Folder

In order to run performance reports in batch, you must publish the champion model from the project folder. The SAS Model Manager performance macros use project metadata when running performance reports.

Whenever you have a new champion model, you must publish the new champion model again.

Create a Folder Structure

Create a folder structure on your SAS Model Manager server to contain the report monitoring files. First, create a root folder to contain performance reporting files for one or more SAS Model Manager projects. You might further organize your file structure by project. The examples in the following table use a classification of HMEQ for the files that are used to create home equity performance monitoring reports. Create folders to contain the following types of files:
<table>
<thead>
<tr>
<th>Folder Contents</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>job local path</td>
<td>Specifies the folder that contains the reporting specification data sets that are used by the %MM_RunReports() macro.</td>
<td>c:\mmReports\HMEQ\reportJobs</td>
</tr>
<tr>
<td>report output</td>
<td>Specifies the folder that contains data sets and auxiliary files that are created during the creation of the performance reports when the %MM_RunReports() macro is run in test mode.</td>
<td>c:\mmReports\HMEQ\testReportOutput</td>
</tr>
<tr>
<td>performance data</td>
<td>Specifies the folder that contains the performance data sets for each time period. Performance data sets can be stored in a DBMS as well. If your performance data set is in a DBMS, then this folder is not necessary.</td>
<td>c:\mmReports\HMEQ\scoreIn</td>
</tr>
</tbody>
</table>
| channel         | Specifies the folder on the SAS Model Manager server to save the SPK file that is created during the processing of the %MM_GetModels() macro. The SPK file contains the model. When you publish a model to a channel, the published model package file (SPK) is placed in this folder.  

*Note:* The channel that you publish a model to must use archive persistence because the %MM_GetModels() macro requires a directory path. The directory path must be the same path that was used for the channel’s persistence.  

A channel can be shared by multiple model projects.  
If you do not use the SAS channel that was predefined during the installation of SAS Model Manager, see the Help for the Publishing Framework in SAS Management Console. | c:\mmReports\HMEQ\channel2 |
Folder Contents | Description | Example
--- | --- | ---
model | Specifies the folder to where the SPK model is extracted to by the `%MM_GetModels()` macro. The macro creates a `scorecode` folder that contains the model score code and saves the data set `current.sas7bdat`, `logs.sas7bdat`, and `processingspk.sas7bdat` in the model folder. The `current.sas7bdat` data set contains project and model information that is used to create the performance monitoring reports. | `c:\mmReports\HMEQ\model\scorecode`

To ensure that your report data is not lost, regularly back up these report folders.

**See Also**
“Report Output in Test and Production Modes” on page 104

**Obtain Performance Data**

The performance data set is a snapshot of a data set that includes scoring input variables and one or more target variables. After the snapshot is available, store the data set in a performance data folder on the SAS Model Manager server.

**See Also**
“Creating a Performance Table” in *SAS Model Manager: User’s Guide*

**Determine the Publish Channel**

You can determine the channel that was used to publish the model by using one of these methods:

- Open a model and select **History ⇒ Log** on the **Model Properties** page. Look for a publish model entry. In this example, the channel name is **MMChannel**: Jul 5, 2015 9:47:18 PM[sasdemo] "Reg 1" was published to "MMChannel(sas-oma://rdesx16083.race.sas.com:8561/reposid=A5BXGH17/ITChannel;id=A5BXGH17.BI000001)" successfully.

- In SAS Management Console, click the **Plug-ins** tab and expand the following nodes under **SAS Management Console** to find the publishing channels that are used by SAS Model Manager: **Environment Management ⇒ Publishing Framework ⇒ Foundation ⇒ Channels ⇒ Model Manager Channels**. Right-click the channel and select **Properties**. The channel path is located on the **Persistent Store** tab.

  **Note:** If the **Plug-ins** tab does not appear in your view of SAS Management Console, contact your SAS administrator.

  **Note:** A publish channel can be shared by multiple projects.
Copy Example Batch Programs

SAS provides several example programs that you can use to create a batch program that monitors the performance of the champion model. You can find the example programs in the sashelp.modelmgr catalog. The catalog includes these example programs:

reportExample1 contains example SAS code to extract a project or model from the channel using the %MM_GetModels() macro.

reportExample2 contains DATA steps to create performance data that can be used to test the batch programs that create performance monitoring reports.

reportExample3 contains example DATA steps to create the SAS data sets that contain report specifications, such as the project UUID and path, various input variables, the location of the performance data source, alert and warning conditions, and email addresses for report notifications.

reportExample4 contains an example program that are used to define the operating environment using macro variables. This program also contains the DATA steps that are used to create the reports.

You can copy these example programs to the job local path folder and you can modify them for your operating environment.

Determine SAS Model Manager User ID and Password

Performance monitoring reports must specify a valid SAS Model Manager user ID and password. The user ID must be in one of the following groups:

- Model Manager Users
- Model Manager Advanced Users
- Model Manager Administrator Users

See Also

“Model Management User Tasks” in SAS Model Manager: Administrator’s Guide

Report Output in Test and Production Modes

Report Output in Test Mode

When you run the %MM_RunReports() macro, you can either run the report in Test mode or Production mode, by using the _MM_ReportMode macro variable.

To run in Test mode, ensure that you make the following assignments:

- In the DATA step mm_jobs.project, set the variable testDestination=reportOutputPath, where reportOutputPath is the report output folder on the SAS Model Manager server or network. This is the location that you defined when you completed the prerequisites for running batch performance jobs.
In the %MM_RunReports() macro, set the macro variable 
_MM_ReportMode=TEST.

Test report output is then written to the SAS Model Manager server or network location. You can test your %MM_RunReports() macro any number of times without corrupting the integrity of your model repository. You can delete the contents of the report output folder and resubmit your macro as necessary.

To view the report output:
1. Sign in to SAS Model Manager.
2. Open the project. For example, open the HMEQ project in the Tutorial3 folder.
3. Select the Versions page and click +. Click OK.
   
   Note: A best practice is to create a new version and add the files to it. If you do not create a new version, ensure that you delete the performance data sets from the Results tab of the Performance page when you no longer need these files.
4. Double-click the new version to select it as the displayed version.
5. Select the Results tab on the Performance page.
6. Click + to add the performance data set from the report output folder that is defined in testDestination=reportOutputPath. For more information, see “Manage Performance Data Sets” in SAS Model Manager: User's Guide.
7. Click ⏰ to update the performance results. You should then be able to select the different charts and view the performance data.

Report Output in Production Mode

When you run the %MM_RunReports() macro in Production mode, ensure that you complete the following code changes:

- In the DATA step mm_jobs.project, remove the assignment of the variable testDestination=reportOutputPath.
- In the %MM_RunReports() macro, set the macro variable 
_MM_ReportMode=Production.

Production report output is written to the champion version of a project.

To view the report output:
1. Sign in to SAS Model Manager.
2. Open the project. For example, open the HMEQ project in the Tutorial3 folder.
3. Select the Versions page and double-click the champion version to select it as the displayed version.
4. Select the Results tab on the Performance page.
5. Click ⏰ to update the performance results. You can now select the different charts and view the performance data.

See Also

“Prerequisites for Running Batch Performance Reports” on page 101
Define the Report Specifications

Overview of Code to Define Report Specifications

Before you can create a monitoring report for a project, you must create several data sets that define the report specifications:

- mm_jobs.project defines the project information, such as the project UUID, project variables, and the model repository URL for the project. It is recommended that you create only one observation in this data set.
- mm_jobs.emailaddr defines the email addresses for the recipients of job status and the notification flags for alert and warning notifications.
- mm_jobs.reportdef defines the types of reports to create and the warning and alert conditions that are associated with those reports.
- mm_jobs.jobtime defines the date and time to run the reports and a label that describes the time performance data set period.

The code that you write to create the report specifications might need to be run only after it is created and only whenever it is modified. These data sets might not need to be created every time you want to create reports.

Required Libref

To create the report specifications, you need to define the following libref:

- mm_jobs defines the local path to the folder that contains the report job files.

Example: `libname mm_jobs "c:\mmReports\HMEQ";`

Project Specifications

DATA Step mm_jobs.project

This DATA step defines the project specifications.

```
/**********************************************/
/* DATA step mm_jobs.project              */
/*                                            */
/* Create a data set to initialize the       */
/* performance monitoring batch program      */
/* project specifications                    */
/**********************************************/

DATA mm_jobs.project;
  length testDestination $150
  projectuuid $36
  projectpath $2000
  projectAlias $50
```
Define the Report Specifications

Variable Descriptions for mm_jobs.project

The following variables are used in the mm_jobs.project DATA step:

isActive='Y | N'

specifies whether to enable the project definitions. Valid values are Y (yes) and N (no). You specify N to indicate that project files do not need to be removed from the SAS Model Manager server in order to deactivate a project entry. Enclose the value of isActive in quotation marks.

Interaction: Always set isActive='Y' when the data set mm_jobs.project has only one observation.
testDestination='reportOutputPath';
    specifies the local path that contains the output files that are created when the
    %MM_RunReports() macro report mode macro variable _MM_ReportMode is set to
    TEST. Enclose the value of testDestination in quotation marks.

    Example: testDestination='c:\mmReports\HMEQ\testOutput';

    See: “Report Output in Test and Production Modes” on page 104

projectuuid
    specifies the universally unique identifier for a SAS Model Manager modeling
    project. To obtain the project UUID, in the SAS Model Manager, open a project and
    select System on the Properties page. You can copy the UUID from the UUID
    property. projectuuid is used to redirect reporting job output data sets to the
    appropriate project folders in the model repository when the %MM_RunReports() macro is run in Production mode.

    Note: If you copy the UUID from SAS Model Manager, you might need to remove
    leading text and spaces that are copied with the UUID.

precode='macroVariableDefinitions'
    specifies the macro variables that are used by the %MM_RunReports() macro. 
    Enclose the value of the precode variable in quotation marks.

    %let _MM_EventProbVar=outputEventProbabilityVariable;
    specifies the name of the output event probability variable. To obtain the name,
    open a project and select Specific on the Properties page. Use one of the values
    for the Output event probability variable property list box.

    %let _MM_TargetVar=targetVariable;
    specifies the target variable name. To obtain the name, open a project and select
    Specific on the Properties page. The target event variable is found in the
    property Training target variable. If a target variable is not specified, see your
    performance data set or the model for the name of the target variable.

    %let _TargetLevel=targetLevel;
    specifies the class target level of binary, nominal, ordinal, or interval. To obtain
    the name, open a project and select Specific on the Properties page. Use one of
    the values from the Class target level property list box.

    %let _MM_TargetEvent=targetEventValue;
    specifies the target event value. To obtain the name, open a project and select
    Specific on the Properties page. The value is found in the property Target event
    value. If a target event value is not specified, see your performance data set or
    the model to determine the value.

    Requirement: The value of _MM_TargetEvent must be an unformatted, raw
    value even if the original target variable has a SAS format applied to it.

    %let _MM_ScoreFlag=scoreFlagValue;
    specifies whether the macro should run the score code in the performance
    monitoring job. Valid values are 1 (yes) and 0 (no). If a value for the score flag
    macro variable is not specified, the default value is set to 1 and the score code is
    run. Using this macro variable has the same effect as using the Run model score
    code option in the Edit Performance Definition window when you are running
    the performance monitoring reports for a project.

    %let _MM_ReportDatasrc=scoreIn.dataSetName;
    specifies the libref and the data set name for the performance data set that is
    being analyzed.
If you process multiple data sets at one time, you can specify a generic data set name in this macro definition. The generic data set name is used to process all performance data sets. Before you run the %MM_RunReports() macro, you should create a DATA step with the name scoreIn.genericDataSetName, where the only statement in the DATA step is the SET statement that specifies the performance data set to process.

%let _MM_KeepVars=variablesToKeep;
specifies one or more output variables, separated by a space, that are kept in the performance data source to create the Stability report data set.

%let _MM_DropVars=variablesToDrop;
specifies one or more input variables, separated by a space, that are dropped from the performance data source to create the Characteristic report data set.

projectPath='projectURL'
specifies the project URL. To obtain the project URL, open a project and select System on the Properties page. You can copy the URL from the URL property. The project URL is used for information purposes only; it is not used to access project resources. projectURL is dynamically retrieved when the %MM_RunReports() macro runs. Enclose the value of projectPath in quotation marks.

projectAlias='alternateProjectName'
specifies an alternate project name. The alternate project name can be used to help identify the project when the projectPath is long. If you do not have an alternate project name, you can leave this variable blank. Enclose the value of projectAlias in quotation marks.

notes='userNotes'
specifies any notes that the user might want to add to the project specifications. Enclose the value of notes in quotation marks.

**Email Recipient Specifications**

**DATA Step mm_jobs.emailaddr**

This DATA step defines the email recipient specifications:

```plaintext
DATA mm_jobs.emailaddr;
  length address $50 sendAlertWarning sendJobStatus $1;
  address='emailAddress';
  sendAlertWarning='Y';
  sendJobStatus='N';
  output;
  address='emailAddress';
  sendAlertWarning='Y';
  sendJobStatus='Y';
  output;
run;
```

---

**Define the Report Specifications** 109
Variable Descriptions for mm_jobs.emailaddr
The following variables are used in the mm_jobs.emailaddr DATA step:

address='emailAddress'
specifies the email address of the user to receive job, alert, and warning notices. Enclose the value of address in quotation marks.

sendAlertWarning='Y | N'
specifies whether alert warning notifications are sent to the email address specified in address. Valid values are Y (yes) and N (no). Enclose the value of sendAlertWarning in quotation marks.

sendJobStatus='Y | N'
specifies whether the job status report is sent to the email address specified in address. Valid values are Y (yes) and N (no). Enclose Y or N in quotation marks.

Report Specifications

DATA Step mm_jobs.reportdef
This DATA step defines the type of reports to create, provides the macro syntax for the report type, and defines alert and warning specifications. You can specify one, two, or three report types in the DATA step. The %MM_RunReports() macro runs the reports that are defined in the mm_jobs.reportdef data set. For each type of report, assign the reportName, the macro, and alert and warning conditions.

/**************************************************/
/* DATA set mm_jobs.reportdef                      */
/*                                                 */
/* Create a data set that defines the report       */
/* metadata and alarm thresholds for the           */
/* Characteristic, Stability, and Model Assessment */
/* reporting jobs.                                */
/**************************************************/

DATA mm_jobs.reportdef;
  length reportName $20
  macro $1000
  alertCondition $200
  warningCondition $200
  isActive $1
  notes $500;

isActive='Y';

/***************************/
/* Characteristic Report */
/***************************/

reportName='Characteristic';
macro=
  %MM_UpdateCharacteristicTable(
    datasrc=&_MM_ReportDatasrc,
    dropVars=&_MM_DropVars;);
Variable Descriptions for mm_job.reportdef

The following variable definitions are used in the mm_jobs.reportdef DATA step:

isActive
  specifies whether to enable the report definitions. Valid values are Y (yes) and N (no). Specifying N means that a report definition file does not need to be removed from the SAS Model Manager server to deactivate a report definition entry.

Interaction: Always set isActive='Y' when the data set mm_jobs.project has only one observation.

reportName='reportName'
  specifies the name of the report. Valid report types are as follows:
  • Characteristic
  • Stability
  • Model Assessment

Enclose reportName in quotation marks. This argument is required.

macro='macroDefinition';
  specifies the report macro that is executed when the %MM_RunReports() macro is executed. This argument is required.
alertConditions="alertConditions";
specifies an alert condition for the type of report. Enclose alertConditions in quotation marks. Here are example alert conditions for each type of report:

<table>
<thead>
<tr>
<th>Report Type</th>
<th>Example Alert Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic</td>
<td>alertCondition='p1&gt;5 or p25&gt;0';</td>
</tr>
<tr>
<td>Stability</td>
<td>alertCondition='outputDeviation &gt; 0.03';</td>
</tr>
<tr>
<td>Model Assessment</td>
<td>alertCondition='lift5Decay&gt;0.15 and lift10Decay&gt;0.12) or giniDecay&gt;0.1 or ksDecay&gt;0.1'; alertCondition='msedecay &gt; 20';</td>
</tr>
</tbody>
</table>

See also: see “Performance Index Warnings and Alerts” in SAS Model Manager: User’s Guide.

warningConditions="warningConditions";
specifies a warning condition for the type of report. Enclose warningConditions in quotation marks.

<table>
<thead>
<tr>
<th>Report Type</th>
<th>Example Warning Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic</td>
<td>warningCondition='p1&gt;2';</td>
</tr>
<tr>
<td>Stability</td>
<td>alertCondition='outputDeviation &gt; 0.01';</td>
</tr>
<tr>
<td>Model Assessment</td>
<td>warningCondition='lift5Decay&gt;0.05'; warningCondition='msedecay &gt;10';</td>
</tr>
</tbody>
</table>

See also: see “Performance Index Warnings and Alerts” in SAS Model Manager: User’s Guide.

notes="userNotes";
specifies a note to add to the report definition data set. Enclose userNotes in quotation marks.

%MM_UpdateCharacteristicTable() Macro
Here is the syntax for the %MM_UpdateCharacteristicTable() macro:

\%
\%MM_UpdateCharacteristicTable(data.src=&_MM_ReportDatasrc, <dropvars=&_MM_DropVars>);

data.src=&_MM_ReportDatasrc
specifies the macro variable that defines the performance data set that is used to create the Characteristic report.

dropvars=&_MM_DropVars
specifies the macro variable that defines the input variables to drop from the performance data set. Consider dropping variables from the performance data set whose values do not need to be monitored.
%MM_UpdateStabilityTable() Macro

Here is the syntax for the %MM_UpdateStabilityTable() macro:

```%
MM_UpdateStabilityTable(datasrc=&_MM_ReportDatasrc,
<keepvars=&_MM_KeepVars>);
```

datasrc=&_MM_ReportDatasrc

specifies the macro variable that defines the performance data set that is used to create the Stability report.

keepvars=&_MM_KeepVars

specifies the macro variable that defines the output variables to keep in the performance data set. Consider keeping only the variables in the performance data set whose values are to be monitored.

%MM_UpdateAssessmentTable() Macro

Here is the syntax for the %MM_UpdateAssessmentTable() macro:

```%
MM_UpdateAssessmentTable(datasrc=&_MM_ReportDatasrc);
```

datasrc=&_MM_ReportDatasrc

specifies the macro variable that defines the performance data set that is used to create the Model Assessment reports.

Job Scheduling Specifications

DATA Step mm_jobs.jobtime

This DATA step defines the dates and times that the data sets that underlie the performance monitoring reports are to be created or updated.

```c
/*********************/
/* DATA step mm_jobs.jobtime       */
/* Define the report schedule by specifying the   */
/* dates and times for each incremental reporting */
/* interval. You can schedule as many jobs as you */
/* would like. The following jobs are scheduled to*/
/* run one second before midnight on the dates    */
/* listed below.                                  */
/*********************/

DATA mm_jobs.jobtime;
length scheduledTime $18 time $10;
scheduledTime='dateTime';time='timePeriodLabel'; output;
run;
```

Variable Descriptions for mm_jobs.jobtime

Here are the variables that are used in the DATA step mm_jobs.jobtime:

scheduledTime='dateTime'

specifies the date and time to run the report. The value of scheduledTime must be in the form `ddmmyyyy:hh:mm:ss` where `dd` is a two-digit year, `mmm` is the first three letters of the month, `yyyy` is a four-digit year, `hh` is a two-digit hour, `mm` is a two-digit minute, and `ss` is a two-digit second. Enclose `dateTime` in quotation marks.

The values of scheduledTime are used by the %MM_RunReports() macro, rather than by your job scheduler. Each time that the %MM_RunReports() macro runs, it
checks the values of the scheduleTime variable. If the scheduled time has passed, the report runs. If it has not passed, the performance data sets are not created.

Example: `scheduledTime='03Jun2012:23:59:00';`

time="timePeriodLabel"
specifies a label that represents the time period for which the performance data was collected. Enclose `time PeriodLabel` in quotation marks. Use short and clear labels to create charts that can be easily read.

Example: `time='2012Q4';`

Example Code to Create the Report Specifications

This example creates a single SAS program to create the report specification data sets. After you copy the example code from the sashelp.modelmgr library, you providing values for the required variables and macros. The variable and macro names are highlighted in the example code to identify the values that you would modify to create the report specifications.

```sas
/* Source file name: sashelp.modelmgr.reportExample3.source */

LIBNAME mm_jobs 'c:\mm.test\report.auto';

/*********************************************/
/* DATA step mm_jobs.project               */
/*                                           */
/* Create a data set to initialize the      */
/* performance monitoring report batch      */
/* job project specification metadata and   */
/* report precode metadata.                 */
/*********************************************/

DATA mm_jobs.project;
  length testDestination $50
  projectuuid $36
  projectpath $200
  projectAlias $50
  precode $32000
  isActive $1
  notes $500;

  isActive='Y';

/*********************************************/
/* Specify the destination path for the report */
/* the universal unique ID for the project */
/*********************************************/

testDestination=
  'c:\mm.test\report.test.output\project_123';
projectuuid=
  '8817ea06-0a28-0c10-0034-68f4ba396538';

/*********************************************/
/* The precode section uses macro variables to */
/* map individual model metadata components */
```
/* to their respective variables, target event */
/* values, and data used to create the report. */
*****************************************************************************
precode='
%let _MM_EventProbVar=p_bad1;
%let _MM_TargetVar=bad;
%let _TargetLevel=BINARY;
%let _MM_TargetEvent=1;
%let _MM_ReportDatasrc=scoreIn.hmeq0;
%let _MM_KeepVars=p_bad1;
%let _MM_DropVars=bad job;
';
*****************************************************************************
/* Specify the path to the project and provide */
/* an Alias name for the project reports. */
*****************************************************************************
projectPath='http://myserver:7980/SASContentServer/repository/default/ModelManager/MMRoot/demo/Creditcardpromotion';
projectAlias='credit risk for younger customers';
run;
*****************************************************************************
/* DATA set mm_jobs.emailaddr */
/* */
/* Create a data set that specifies the email */
/* recipient notification list, and whether to */
/* send the alert, warning, and job status */
/* notifications. */
*****************************************************************************
DATA mm_jobs.emailaddr;
length address $50 sendAlertWarning sendJobStatus $1;
address='recipient1@mail.com';
sendAlertWarning='Y';
sendJobStatus='N';
output;
address='recipient2@mail.com';
sendAlertWarning='Y';
sendJobStatus='Y';
output;
run;
*****************************************************************************
/* DATA set mm_jobs.reportdef */
/* */
/* Create a data set that defines the report */
/* metadata and alarm thresholds for the */
/* Characteristic, Stability, and Model Assessment */
/* reporting jobs. */
*****************************************************************************
DATA mm_jobs.reportdef;
length reportName $20
  macro $1000
  alertCondition $200
  warningCondition $200
  isActive $1
  notes $500;
 isActive='Y';

/*****************************************/
/* Characteristic Report */
/*****************************************/
reportName='Characteristic';
macro='
  %MM_UpdateCharacteristicTable(
    datasrc=&_MM_ReportDatasrc,
    dropVars=&_MM_DropVars;)
';
  alertCondition='p1>5 or p25>0';
  warningCondition='p1>2';
  output;

/*****************************************/
/* Stability Report */
/*****************************************/
reportName='Stability';
macro='
  %MM_UpdateStabilityTable(
    datasrc=&_MM_ReportDatasrc,
    keepVars=&_MM_KeepVars;)
';
  alertCondition='outputDeviation > 0.03';
  warningCondition='outputDeviation > 0.01';
  output;

/*****************************************/
/* Model Assessment Report */
/*****************************************/
reportName='Model Assessment';
macro='
  %MM_UpdateAssessmentTable(
    datasrc=&_MM_ReportDatasrc);
';
  alertCondition='(lift5Decay>0.15 and lift10Decay>0.12)
    or giniDecay>0.1
    or ksDecay>0.1';
  warningCondition='lift5Decay>0.05';
  output;
 run;

/*******************************************/
Extracting the Champion Model from a Channel

Using the %MM_GetModels() Macro

Before you run the %MM_RunReports() macro, you must extract the model from the publishing channel to the SAS Model Manager server. The model must have been published to the channel from the project folder. The %MM_GetModels() macro extracts models and auxiliary files from a SAS Publishing Framework SPK file to the SAS Model Manager server. All models that were published to the specified channel are included in the SPK file for a given modeling project. If a model has been published multiple times over the channel, the latest model is used in the extraction. The macro then extracts the files from the SPK file to their respective folders on the SAS Model Manager server. The auxiliary files are extracted to the model folder and the model score code is extracted to a folder named \scorecode, which the macro creates as a subfolder of the model folder.

Note: You can run the %MM_GetModels() macro when no new model has been published to the channel for a modeling project.

The auxiliary files include three SAS data sets:
- current.sas7bdat contains project and model metadata
- logs.sas7bdat contains the SAS logs that were created during the model extraction process
processingspk.sas7bdat contains information that is necessary to process the SPK file. The models in the `scorecode` folder are named using the project UUID as the model folder name. The %MM_RunReports() macro uses the mm_jobs.project data set to determine the project UUID. The project UUID is then used as the name of the model on the SAS Model Manager server for scoring when the performance monitoring reports are created.

The current data set contains project and model information and is used by the %MM_RunReports() macro. To ensure that the %MM_RunReports() macro is using the most current project and model metadata, always run the %MM_GetModels() macro before you run the %MM_RunReports() macro. For a list of the information that is contained in the current data set, see “The current.sas7bdat Data Set” on page 119.

Accessing Model Management Report Macros

The %MM_RunReports() macro, the %MM_GetModel() macro, and all other Model Management macros are available in the catalog sashelp.modelmgr.reportmacros.source. Use the following FILENAME statement to make these macros available to your program:

```sas
FILENAME repmacro catalog 'sashelp.modelmgr.reportmacros.source';
%inc repmacro;
```

%MM_GetModels() Macro Syntax

Here is the syntax for the %MM_GetModels() macro:

```sas
%MM_GetModels(channel=channelPath,localPath=localModelPath);
```

- `channel=channelPath` specifies the path of the channel to extract the models from. To obtain the channel path, see “Determine the Publish Channel” on page 103. Do not enclose the value of channel in quotation marks.

- `localPath=localModelPath` specifies a folder on the SAS Model Manager server to where the model and auxiliary files are extracted from the SPK file. Do not enclose `localModelPath` in quotation marks.

Example Program to Extract a Model from a Channel

The following SAS code uses the %MM_GetModel macro to extract a champion model from a channel.

```sas
/* Source file name: sashelp.modelmgr.reportExample1.source */
FILENAME mmmac
catalog 'sashelp.modelmgr.reportmacros.source';
%inc mmmac;

%MM_GetModels{
  channel=\\network1\MMChampion\channel1,
```
The current.sas7bdat Data Set

When models are extracted from a publishing channel, the current.sas7bdat data set contains the following information for each model:

<table>
<thead>
<tr>
<th>Variable Name for the Project or Model Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The algorithm that was used to create the model</td>
</tr>
<tr>
<td>fileName</td>
<td>Not used</td>
</tr>
<tr>
<td>isChampionModel</td>
<td>True or False to indicate whether the model is the champion model</td>
</tr>
<tr>
<td>keyWords</td>
<td>Keywords</td>
</tr>
<tr>
<td>miningFunction</td>
<td>The type of mining function, such as classification, prediction, segmentation</td>
</tr>
<tr>
<td>model</td>
<td>Not used</td>
</tr>
<tr>
<td>modeler</td>
<td>The name of the person who created the model</td>
</tr>
<tr>
<td>modelName</td>
<td>The name of the model</td>
</tr>
<tr>
<td>modelProductionTimestamp</td>
<td>The time at which the model was declared as a production model</td>
</tr>
<tr>
<td>modelTool</td>
<td>The name of the tool that was used to train the model</td>
</tr>
<tr>
<td>modelUUID</td>
<td>The UUID for the model</td>
</tr>
<tr>
<td>nodeDescription</td>
<td>Not used</td>
</tr>
<tr>
<td>projectPath</td>
<td>The project URL</td>
</tr>
<tr>
<td>project UUID</td>
<td>The UUID for the project</td>
</tr>
<tr>
<td>repository</td>
<td>The repository URL</td>
</tr>
<tr>
<td>scoreCodeType</td>
<td>DATA step or SAS program</td>
</tr>
<tr>
<td>subject</td>
<td>The subject name</td>
</tr>
<tr>
<td>targetName</td>
<td>The Training target variable name</td>
</tr>
<tr>
<td>userAttr</td>
<td>User-defined attributes, such as &quot;MODELER='sasguest' MODELPROJECTVARMAP='predictedProbability eq P_BAD1; predictedClass eq I_BAD;&quot;</td>
</tr>
</tbody>
</table>
### Variable Name for the Project or Model Information

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionName</td>
<td>The name of the version that contains the model</td>
</tr>
<tr>
<td>whenPublished</td>
<td>The date and time at which the project or model was published to the channel</td>
</tr>
<tr>
<td>whoPublished</td>
<td>The user who published the model</td>
</tr>
</tbody>
</table>

### See Also

- “Define the Report Specifications” on page 106
- “SAS Code to Run Performance Reports” on page 120

### SAS Code to Run Performance Reports

#### Overview of the SAS Code to Run the Performance Reports

After you have created the data sets that define the report specifications and have extracted the model from the publishing channel, you then run the `%MM_RunReports()` macro to create the reports for one or more time periods. Using the data sets that were created to define the report specifications, the `%MM_RunReports()` macro uses the report specifications to create the reports. The report specifications include the type of report to create, such as characteristic, stability, or model assessment. Other report specifications include the target variable, the libref, and the data set name that is used as the performance data source, variables to keep and drop from reports, email addresses to send report notifications, and performance index warnings and alerts.

To run the `%MM_RunReports()` macro, your code must accomplish the following tasks:

- access the reporting macros
- define the librefs and the macro variables that are required by the `%MM_RunReports()` macro
- specify the performance data set to process. To do this, execute a DATA step before each `%MM_RunReports()` macro

To ensure that you have the latest model, extract the model from the channel each time you create the performance reports. For this reason, you could combine into one SAS program the extraction process and the code to run the reports.

If you run a set of batch jobs every night, you could include this batch job with that set of batch jobs. The reports would be created only after the scheduled date and time that is specified in the `mm_jobs.jobtime` data set.

The following sections describe each of these components of your SAS program. The last section is an example of a program that is used to test the `%MM_RunReports()` macro.

If you are running macros in a SAS session that requires a connection to the SAS Metadata Server, you must specify the metadata connection system options before you run the macros. If these options are not specified, you will be prompted for the
information. If you are running the macros in batch mode, you must include the metadata connection system options. Otherwise, you will receive a failure message. For more information, see “Setting Metadata Connection System Options” on page 2.

Accessing Model Management Report Macros

The %MM_RunReports() macro, the %MM_GetModel() macro, and all other Model Management macros are available in the catalog sashelp.modelmgr.reportmacros.source. Use the following FILENAME statement to make these macros available to your program:

    filename repmacro catalog 'sashelp.modelmgr.reportmacros.source';
    %inc repmacro;

Required Librefs

The following librefs are required in your report monitoring program:

- **mm_jobs**
  defines the local path to the folder that contains the report job files.
  
  Example: `libname mm_jobs "c:\mmReports\HMEQ";`

- **mm_meta**
  defines the local path to the folder that stores the data sets that are created from running the %MM_GetModels() macro. The value of this libref must have the same value as the localPath argument for the %MM_GetModels() macro.
  
  Example: `libname mm_meta "c:\mmReports\HMEQ\model";`

- **scoreIn**
  specifies a user-defined libref that points to the local path that contains the performance data sources.
  
  Interaction: You can use this libref when you set the value of Model Management macro variables, such as _MM_ReportDatasrc, in the precode variable of the mm_jobs.project data set. Here is an example: `%let _MM_ReportDatasrc=scoreIn.foo;`

  Example: `libname scoreIn "c:\mmReports\project1\perfdatasets";`

Macro Variables to Define Report Local Folders and Data Sets

Define the following macro variables in your report monitoring program. Then define the location of the job and model on the SAS Model Manager server:

- **_MM_JobLocalPath**
  specifies the path on the SAS Model Manager server that contains the root folder for the reporting files of a given modeling project.
  
  Example: `%let _MM_JobLocalPath=c:\mmReports\HMEQ1;`

- **_MM_ModelLocalPath**
  specifies the path on the SAS Model Manager server that contains the model after it has been extracted from the SAS Metadata Repository.
  
  Example: `%let _MM_ModelLocalPath=c:\mmReports\HMEQ\model;`
mapTable
  specifies a libref and data set in the form `libref.dataSet` that contains the mapping of
  the project output variables to the model output variables. When the model is
  extracted from the channel, the data set `current.sas7bdat` is extracted to the folder that
  contains the model. Use this data set as the value of `mapTable`.

Example: `mapTable=mm_meta.current`. The data set name `current` is
  arbitrary. It is recommended that you use the name `current`.

For a description of the macro variables, see “Macro Variables” on page 4.

Macro Variables That Are Used by the `%MM_RunReports() Macro`

Required Macro Variables
The following macro variables are required to run the `%MM_RunReports()` macro:

  `_MM_User`
  specifies a valid user.

  `_MM_Password`
  specifies the password for the user who is identified in the `_MM_User` macro
  variable.

  See: “Encoding SAS Model Manager User Passwords” on page 122

For a description of the macro variables, see “Macro Variables” on page 4.

Optional Macro Variable
The example programs use the following global macro variable, which you might find
  useful in your report monitoring program:

  `_MM_ReportMode`
  specifies the mode to run the `%MM_RunReports()` macro. Valid values are TEST
  and PRODUCTION. The default value is PRODUCTION. You might want to use a
  value of TEST while you are testing your program. When the value is TEST, the
  report output files are written to the SAS Model Manager server. When the value is
  PRODUCTION, the report output files are written to the appropriate project folders
  in the model repository.

  Interaction: If `_MM_ReportMode` is set to TEST, you must supply a value for the
  testDestination variable in the `mm_jobs.project` data set.

  Example: `%let _MM_ReportMode=TEST;`

For a description of the macro variables, see “Macro Variables” on page 4.

Encoding SAS Model Manager User Passwords
Each time that your run a SAS program to be processed by SAS Model Manager, you
  specify a user ID and assign the user's password to the global macro variable
  `_MM_Password`. In order to not store passwords in clear text, you can use the
  PWENCODE procedure to encode a password and store it in a file, in a network-
  accessible directory. Then, in your SAS program, you create a fileref to the network file
  that contains the encoded password and you use a DATA step to assign the encoded
  password to the `_MM_Password` global macro variable.

In a separate SAS program, encode your password:

  `filename pwfile "my-network-path\pwfile";`
In your SAS program, use a DATA step to access the encoded password file:

```sas
proc pwencode in="12345" out=pwfile;
run;
```

```sas
filename pwfile "my-network-path\pwfile";
%let _MM_User=mmuser1;
data_null_;   infile pwfile obs=1 length=l;   input @;   input @1 line $varying1024. l;   call symput('_MM_Password',substr(line,1,l));
run;
```

**The DATA Step to Access the Performance Data Set**

You use a DATA step to access the performance data set before you run the %MM_RunReports() macro:

```sas
DATA libref.dataStepName;
   set libref.performanceDataSetName;
run;
```

Here is an example of a DATA step to access the performance data set:

```sas
DATA scoreIn.hmeq;
   set scoreIn.hmeq_perf_q1;
run;
```

**The %MM_RunReports() Macro**

**Description of the %MM_RunReports() Macro**

You use the %MM_RunReports() macro to create or update the data sets that underlie the performance monitoring reports. Before each %MM_RunReports() macro that you specify in your program, you might want to update the performance data set by including a DATA step that accesses the performance data set input file.

The %MM_RunReports() macro uses the data sets that are stored in the library that is specified by the mm_jobs libref. These data sets define the report specifications and are the data sets that are created in the report specification program. For more information about the report specification program, see “Define the Report Specifications” on page 106.

**Syntax**

Use the following syntax for the %MM_RunReports() macro:

```sas
%MM_RunReports(
   <dataSource=libname.dataset>, localPath=&_MM_JobLocalPath,
   mapTable=&mapTable,
   user=&_MM_User, password=&_MM_Password, <currentTime=&currentTime>);
```
**Syntax Description**

dataSource=libname.dataset

specifies the name of the performance data set. The default value is *scoreIn.hmeq0*. The default value is used if you do not specify this argument when calling this macro.

Example: `dataSource=mmlib.hmeq_perf_Q2`

localPath=&_MM_ModelLocalPath

specifies the path on the SAS Model Manager server to the location where the %MM_GetModels() macro stores the files extracted from the channel. The %MM_RunReports() macro retrieves the score code from the score code folder, which is a subfolder of &_MM_ModelLocalPath.

Example: `localPath=&_MM_ModelLocalPath`

mapTable=&mapTable

specifies the name of the data set that contains metadata about the extracted model. mapTable is the data set named current.sas7bdat that is created when the model is extracted using the %MM_GetModels() macro. No modification of this argument is necessary.

Example: `mapTable=&mapTable`

user=&_MM_User

specifies a valid user. Use the macro variable that defines the valid user.

Example: `user=&_MM_User`

password=&_MM_Password

specifies the password for _MM_User. Use the _MM_Password global macro variable that defines the password for the user. The value of _MM_Password is a text string.

Example: `password=&_MM_Password`

See: “Encoding SAS Model Manager User Passwords” on page 122

currentTime=currentTime

specifies a time to use for the current time. Use this argument for testing the %MM_RunReports() macro. You do not need to specify an argument for currentTime when you run the macro in a production environment, where the system timestamp is used as a value for currentTime.

The value of currentTime must be in the form **ddmmmyyyy:hh:mm:ss** where *dd* is a two-digit year, *mmm* is the first three letters of the month, *yyyy* is a four-digit year, *hh* is a two-digit hour, *mm* is a two-digit minute, and *ss* is a two-digit second.

Example: `currentTime=03Jul2013:12:15:30`

**Example %MM_RunReports() Macro**

The following code is an example of using the %MM_RunReports() macro:

```sas
%MM_RunReports(
    dataSource=scoreIn.hmeq0,
    localPath=&_MM_ModelLocalPath,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);
```
Example Code to Run the Reports

The following example program defines the librefs and macro variables to test the %MM_RunReports() macro's ability to assess home equity performance data for multiple time periods. Before this section of code can be run, the report specifications must be defined in SAS data sets and the model must be extracted from the publishing channel. For more information, see “Define the Report Specifications” on page 106 and “Extracting the Champion Model from a Channel” on page 117.

The example program sets the current time to a time that would trigger the creation of data sets or the updating of data sets that underlie the model monitoring reports. When you run your batch program in a production environment, you do not need a variable to set the current time. When no value is set for the current time, the %MM_RunReports() macro uses the system timestamp as the value of the current time variable.

The highlighted values are user-supplied values.

```sas
/* Source file name: sashelp.modelmgr.reportExample4.source */

FILENAME repmacro catalog 'sashelp.modelmgr.reportmacros.source';
%inc repmacro;

FILENAME pwfile "my-network-path\pwfile";

/***************************************************/
/* Specify the report execution metadata and */
/* configure the _MM_ macro variables to run the */
/* report job in TEST mode. */
/***************************************************/

%let _MM_ReportMode=TEST;
%let _MM_User=mmuser1;

/* DATA step scoreIn.hmeq0 */
SAS Code to Run Performance Reports 125
```
/* First, run the Q4 report. It is necessary to */
/* artificially declare a "currentTime" argument */
/* of 01Jan2013 in order to trigger the report */
/* execution scheduled for the Q4 interval. */
*******************************************************************************/

%let currentTime=01Jan2013:12:30:15;
%MM_RunReports(
    localpath=&_MM_JobLocalPath,
    currentTime=&currentTime,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);

*******************************************************************************/

/* Now, run the Q1 report. It is necessary to */
/* artificially declare a "currentTime" argument */
/* of 03Apr2012 in order to trigger the report */
/* execution scheduled for the Q1 interval. */
*******************************************************************************/

%let currentTime=03Apr2012:12:30:15;
%MM_RunReports(
    dataSource=scoreIn.hmeq_perf_q1, localpath=&_MM_JobLocalPath,
    currentTime=&currentTime,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);

*******************************************************************************/

/* Now, run the Q2 report. It is necessary to */
/* artificially declare a "currentTime" argument */
/* of 03Jul2012 in order to trigger the report */
/* execution scheduled for the Q2 interval. */
*******************************************************************************/

%let currentTime=03Jul2012:12:30:15;
%MM_RunReports(
    dataSource=scoreIn.hmeq_perf_q2, localpath=&_MM_JobLocalPath,
    currentTime=&currentTime,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);

*******************************************************************************/

/* Now, run the Q3 report. It is necessary to */
/* artificially declare a "currentTime" argument */
/* of 03Oct2012 in order to trigger the report */
/* execution scheduled for the Q3 interval. */
*******************************************************************************/

%let currentTime=03Oct2012:12:30:15;
%MM_RunReports(
    dataSource=scoreIn.hmeq_perf_q3, localpath=&_MM_JobLocalPath,
    currentTime=&currentTime,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);
/**************************************************/
/* Now, run the Q4 report. It is necessary to     */
/* artificially declare a "currentTime" argument */
/* of 03Jan2013 in order to trigger the report  */
/* execution scheduled for the 2012Q4 interval.  */
/*------------------------------------------------------------------------------*/
%let currentTime=03Jan2013:12:30:15;
%MM_RunReports(
    dataSource=scoreIn.hmeq_perf_q4,  localpath=&_MM_JobLocalPath,
    currentTime=&currentTime,
    mapTable=&mapTable,
    user=&_MM_User,
    password=&_MM_Password);

See Also

- “Define the Report Specifications” on page 106
- “Extracting the Champion Model from a Channel” on page 117
Overview of Portfolio Macros

SAS Model Manager portfolio macros enable you to publish project champion models from within a portfolio to a database or Hadoop. They also enable you to score project champion models that are within a portfolio. You can publish or score only those SAS Factory Miner project segment champion models that have been registered to the SAS model repository. The portfolios that contain the SAS Factory Miner project segments and models are located in the FactoryMiner folder of the model repository. They can be viewed in the Portfolios category of the SAS Model Manager web application. For more information, see “Overview of Portfolios” in SAS Model Manager: User’s Guide.

Here are the macros that are located in the SASHELP.modelmgr catalog:

%MM_CombineModelScoreCode
   This macro combines the score code of models that are stored within a SAS Model Manager portfolio into one file.

%MM_ScorePortfolioModels
   This macro is used for scoring models within a portfolio.

%MM_PublishPortfolioModelsDB
   This macro is used for publishing models from a portfolio to a database.

%MM_PublishPortfolioModelsHadoop
   This macro is used for publishing models from a portfolio to Hadoop.
Prerequisites

Before calling the portfolio publishing and scoring macros, you must use the public access macro %MM_CreateModelDataset to get the models’ properties from a portfolio. For more information, see “%MM_CreateModelDataset Macro” on page 43.

Only models that meet the following requirements can be combined and registered as one model, and then published or scored:

• The model is a project champion.
• The model score code type is DATA step.

Dictionary

%MM_CombineModelScoreCode Macro

The %MM_CombineModelScoreCode macro combines the score code of models that are stored in SAS Model Manager portfolios into one SAS code file.

Syntax

%MM_CombineModelScoreCode(
   workPath= input-directory-path,
   modelDS= model-property-table-name,
   portfolioScoreCode= scorecode-filename,
   codeFmt= DS | DS2,
   ds2Mode= SCTHR | MAC | EP | PKGARGS
);

Required Arguments

workPath= input-directory-path
   Specifies the directory path for the downloaded model files. The default value is the directory path of the SAS work library if you do not specify a value.

modelDS= SAS-dataset-name
   Specifies the name of the model property table that is generated from the %MM_CreateModelDataset macro.

portfolioScoreCode= score-code-filename
   Specifies the full directory path and filename of the generated portfolio score code.

Optional Arguments

codeFmt= DS | DS2
   Specifies the format for the generated score code. By default, the value is DS for DATA step score code.
ds2Mode=SCTHR | MAC | EP | PKGARGS
Specifies the parameter. ds2Mode is typically used for translating DATA step code for the dynamic in-database projects in SAS Embedded Process. The default value is MAC. This parameter should be set only when codeFmt=DS2.

SCTHR This flag tells DSTRANS to generate a DS2 thread to represent the score code.

MAC This flag is similar to SCTHR, except that DSTRANS generates a data statement instead of a thread, and uses macro names that are related to the SAS Embedded Process.

EP This flag is generic for all SAS Embedded Process projects. These are similar to the SCTHR flag, but instead of a thread, a DATA step is generated. For these flags, the input and output tables are hardcoded to the SAS Embedded Process virtual table names SASEP.OUT and SASEP.IN, rather than using the macros.

PKGARGS This flag tells DSTRANS to generate a real DS2 package instead of the C code PKG version. The PKGARGS package has standard DS2 parameters for the inputs and outputs instead of C structures.

Details
Here is the high-level process for the actions performed by this macro:

1. Get model files for each project champion model that is within a portfolio. The model files include score.sas, MiningResult.xml, and so on.
2. Combine all of the score code for the models into one SAS code file.
3. Transfer the combined score code to DS2 format if the user sets the macro parameter codeFmt=DS2.
4. Create a model SAS package file and register the model into the model repository.

Example: Combine Champion Model Score Code and a Register Model SAS Package File

******************************************************************
/* Step 1: Get all champion models from the specified portfolio */
******************************************************************
%let _MM_User=sasdemo;
%let _MM_Password=mypassword;
/* Portfolio UUID */
%let FMUUID=%nrstr(0ab98ee1-0a25-0f88-7dac-5ce40c19a999);

/* set up metadata connections */
options metaPort=8561
metaServer=myserver.sas.com
metaRepository=Foundation
metaUser=sasdemo
metaPass=xxx;

filename MMAccess catalog 'SASHELP.modelmgr.AccessMacros.source';
%include MMAccess;
filename MMAccess;

filename mmfm catalog 'SASHELP.modelmgr.MM_CombineModelScoreCode.source';
%include mmfm;
filename mmfm;

%let _MM_RC= -1;
%let propertyName = %str(ProjectName FolderName name projectuuid modeluuid isdefaultversion ischampion projectpath projecturl versionname tool scorecodetype);

%MM_CreateModelDataset(
    mDatasetName=_models,
    smmpath=%superq(FMUUID),
    propertyName=&propertyName,
    isChampion=Y,
    trace=ON);

/**************************************************************/
/* Step 2: Create a combined score code for the portfolio champion models */
/**************************************************************/
%let _MM_Portfolio_Scorecode=c:\temp\portfolio_score.sas;
%MM_CombineModelScoreCode(
    workPath=c:\temp,
    modelDS=_models,
    portfolioScoreCode=&_MM_Portfolio_Scorecode
);

/**************************************************************/
/* Step 3: Create an SPK file and register it in the SAS Metadata Repository */
/**************************************************************/

libname mmlib "C:\SMM143Samples\Data";
aamodel;
aamodel_register(
    modelname=FMSegCombined,
    modeldesc=%nrbquote(The combined DATA step model),
    data=mmlib.hmeq_train,
    target=bad,
    level=binary,
    scorecodefile=&_MM_Portfolio_Scorecode,
    scorecodeformat=DATASTEP,
    register=N,
    spk=Y,
    spkfolder=c:\temp,
    miningfunction=classification
);

%put &aa_spk_folderid;
%MM_ScorePortfolioModels Macro

The %MM_ScorePortfolioModels macro supports scoring models that are within a portfolio.

Syntax

%MM_ScorePortfolioModels (  
scoreInputDS=score-input-table,  
scoreOutputDS=score-output-table,  
workPath=input-directory-path,  
modelDS=model-property-table-name,  
portfolioScoreCode=score-code-filename,  
<codeFmt=DS | DS2>,  
<runScore=N | Y>  
);  

Required Arguments

scoreInputDS=score-input-table
   Specifies the name of scoring input table.

scoreOutputDS=score-output-table
   Specifies the name of scoring output table.

workPath=input-directory-path
   Specifies the input directory path for the downloaded model files. The default value is the directory path of the SAS work library if you do not specify a value.

modelDS=SAS-dataset-name
   Specifies the name of the model property table that is generated from the %MM_CreateModelDataset macro.

portfolioScoreCode=score-code-filename
   Specifies the full directory path and filename of the score code to be generated.
**Optional Arguments**

`codeFmt=DS | DS2`

Specifies the format for the generated score code. By default, the value is DS for DATA step score code.

`runScore=N | Y`

Specifies whether the generated score code should be executed. The default value is N, to not execute the score code.

**Details**

Here is the high-level process for the actions performed by this macro:

1. Get model files for each model from a portfolio. The model files include inputvar.xml, MiningResult.xml, and so on.
2. Generate the score code based on the model score code type of DATA step or analytic store.
3. Combine all of the score code for the models into one SAS code file.
4. Execute the score code.

**Example: Generate Score Code**

This example shows how to generate score code from a SAS Model Manager portfolio. In this example, there are five project segments inside the portfolio. There is one champion model for each project segment. The combined score code consists of the five models' score code.

```sas
/*****************************************
/* Generate score code for a SAS Model Manager portfolio */
*****************************************
filename score catalog 'SASHELP.modelmgr.MM_ScorePortfolioModels.source';
%include score;
filename score;
libname scorelib 'c:\temp';
%MM_ScorePortfolioModels(
    scoreInputDS=scorelib.hmeq_score_input,
    scoreOutputDS=scorelib.hmeq_score_output,
    workPath=c:\temp,
    modelDS=_models,
    portfolioScoreCode=c:\temp\portfolio_score.sas,
    codeFmt=DS,
    runScore=Y
);
```

134  Chapter 7 • Macros for Publishing and Scoring Models from within a Portfolio
%let _MM_InputDS=scorelib.hmeq_score_input;
%let _MM_OutputDS=scorelib.hmeq_score_output;
%let _MM_nthreads=2;

/* Score Code for the segment: JOB='ProfExe' */
proc astore;
   performance details nthreads=&_MM_nthreads;
   score data=&_MM_InputDS(where=(JOB='ProfExe')) out=_seg_2_out
      epcode="c:\temp\seg_2_score.sas" store="c:\temp\seg_2_score.sasast";
   run;
quit;

/* Score Code for the segment: JOB='Office' */
proc astore;
   performance details nthreads=&_MM_nthreads;
   score data=&_MM_InputDS(where=(JOB='Office')) out=_seg_3_out
      epcode="c:\temp\seg_3_score.sas" store="c:\temp\seg_3_score.sasast";
   run;
quit;

/* Score Code for the segment: JOB='Other' */
proc astore;
   performance details nthreads=&_MM_nthreads;
   score data=&_MM_InputDS(where=(JOB='Other')) out=_seg_5_out
      epcode="c:\temp\seg_5_score.sas" store="c:\temp\seg_5_score.sasast";
   run;
quit;

/* Score code for the segment: JOB='Self' */
data _seg_1_out;
   set &_MM_InputDS(where=(JOB='Self'));

Chapter 7 • Macros for Publishing and Scoring Models from within a Portfolio

*------------------------------------------------------------*;
* SAS Factory Miner Release: 15.1;
* SAS Release:           9.04.01M4P110916;
* Site Number:           00000001;
* Host:                  MYSERVER;
* Encoding:              utf-8;
* Java Encoding:         UTF8;
* Locale:                en_US;
* Project Path:          C:\SAS\Config\Lev1\AppData\FactoryMinerServer;
* Project Name:          FCMR_sasdemo_1434294488570;
* Segment Id:            8aa58f084df27e88014df29e0b430390;
* Segment Server Id:     7;
* Model Id:              8aa58f084df29e88014df29e0b430390;
* Generated by:          sasdemo;
* Date:                  14NOV2018:11:11:39
*------------------------------------------------------------*;
*------------------------------------------------------------*;
* Segment:          8aa58f084df27e88014df29e0b430390;
* SegmentServerId:  7;
*------------------------------------------------------------*;
if (trim(JOB) = 'Self')  then do;
*------------------------------------------------------------*;
* Node: Ids;
*------------------------------------------------------------*;
*------------------------------------------------------------*;
* Node: HPImp;
*------------------------------------------------------------*;
Label IMP_CLAGE = 'Imputed CLAGE';
Length IMP_CLAGE 8;
if missing("CLAGE"n) then IMP_CLAGE = 174.53622491;
else IMP_CLAGE = "CLAGE"n;
Label IMP_CLNO = 'Imputed CLNO';
Length IMP_CLNO 8;
if missing("CLNO"n) then IMP_CLNO = 23.026315789;
else IMP_CLNO = "CLNO"n;
Label IMP_DEBTINC = 'Imputed DEBTINC';
Length IMP_DEBTINC 8;
if missing("DEBTINC"n) then IMP_DEBTINC = 35.568005927;
else IMP_DEBTINC = "DEBTINC"n;
Label IMP_DELINQ = 'Imputed DELINQ';
Length IMP_DELINQ 8;
......
%------------------------------------------------------------*;
* Generating fixed output names;
*------------------------------------------------------------*;
Length EM_EVENTPROBABILITY 8;
LABEL EM_EVENTPROBABILITY = "Predicted: BAD=1";
EM_EVENTPROBABILITY = P_BAD1                          ;
LENGTH EM_CLASSIFICATION $32;
LABEL EM_CLASSIFICATION= "Prediction for BAD";
EM_CLASSIFICATION =I_BAD                           ;
Length EM_PROBABILITY 8;
LABEL EM_PROBABILITY = "Probability of Classification";
EM_PROBABILITY = max(P_BAD1,P_BAD0);  
end;
run;

/* Collect all segment outputs */
data &MM_OutputDS;  
  set _seg_1_out _seg_2_out _seg_3_out _seg_4_out _seg_5_out;  
run;

proc datasets nolist nodetails;  
delete _seg_1_out _seg_2_out _seg_3_out _seg_4_out _seg_5_out _segDS ;  
run;  
quit;

%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;
%------------------------------------------------------------*;

%MM_PublishPortfolioModelsDB

The %MM_PublishPortfolioModelsDBmacro is used for publishing models from a portfolio to a database.

**Requirement:** The macro is limited to publishing the project champion models whose score code type is DATA step. The publishing of SAS analytic store models is not currently supported.

**Syntax**

%MM_PublishPortfolioModelsDB (  
dbType=database-type,  
dbServer=database-server,  
dbUser=database-user,  
dbPassword=database-password,  
database=database-name,  
modelDS=sas-dataset-name,  
outputDir=output-directory-path  );

**Required Arguments**

dbType=database-type  
    Specifies the type of database (for example, Teradata).

dbServer=database-server  
    Specifies the name of the database server.

dbUser=database-user  
    Specifies the user name for the database.
\[ \text{dbPassword=database-password} \]

Specifies the password for the database user.

\[ \text{modelDS=SAS-dataset-name} \]

Specifies the name of the model property SAS data set that is generated from the \%MM_CreateModelDataset macro.

\[ \text{outputDir=output-directory-path} \]

Specifies the directory path where to store the publish results.

**Example: Publish Models to a Database**

This example shows how to publish the champion models of a SAS Model Manager portfolio to Teradata.

```sas
filename pubdb catalog 'SASHELP.modelmgr.MM_PublishPortfolioModelsDB.source';
%include pubdb ;
filename pubdb ;

%let Database=myDatabase;
%let DbType=Teradata;
%let Server=myTeradataServer;
%let User = %nrbquote(myDatabaseUserName);
%let Password = %nrbquote(myDatabasePassword);

%MM_PublishPortfolioModelsDB(
   dbType=&DbType,
   dbServer=&Server,
   dbUser =&User,
   dbPassword=&Password,
   database=&Database,
   modelDS=_models,
   outputDir=%nrstr(C:\temp\FMScore)
);
```

---

**%MM_PublishPortfolioModelsHadoop**

The %MM_PublishPortfolioModelsHadoop macro is used for publishing models from a portfolio to Hadoop.

**Syntax**

```sas
%MM_PublishPortfolioModelsHadoop(
   hadoopServer=Hadoop-server,
   hadoopUser=Hadoop-user,
   hadoopPassword=Hadoop-password,
   hadoopDir=model-directory-path,
   modelDS=model-property-dataset-name,
   outputDir=output-directory-path
);
```
**Required Arguments**

**hadoopServer**=*Hadoop-server*
- Specifies the name of the Hadoop Distributed File System server.

**hadoopUser**=*Hadoop-user*
- Specifies the user name for Hadoop.

**hadoopPassword**=*Hadoop-password*
- Specifies the password for the Hadoop user.

**hadoopDir**=*model-directory-path*
- Specifies the Hadoop directory path where to store the published model.

**modelDS**=*SAS-dataset-name*
- Specifies the name of the model property SAS data set that is generated from the `%MM_CreateModelDataset` macro.

**outputDir**=*output-directory-path*
- Specifies the directory path where to store the publish results.

**Example: Publish Models to Hadoop**

This example shows how to publish the champion models of a SAS Model Manager portfolio to Hadoop.

```sas
filename pubhd catalog 'SASHELP.modelmgr.MM_PublishPortfolioModelsHadoop.source';
%include pubhd;
filename pubhd;

%let hdServer=myHadoop.server.com;
%let hdPath=/tmp/mmng;
%let hdUser = %nrbquote(hadoop);
%let hdPassword=hadoop1;
%let HADOOP_Auth = ;

%MM_PublishPortfolioModelsHadoop(
   hadoopServer =&hdServer,
   hadoopUser =&hdUser,
   hadoopPassword=&hdPassword,
   hadoopDir =&hdpath,
   outputDir=&nstr(C:\temp\FMscore),
   modelDS=_models
);
```
Chapter 8
Macro for Converting and Exporting Model DS2 Code

Overview

The %MM_GetModelDS2Code macro enables you to convert and export a model’s DS2 score code from the model repository. This macro simplifies the integration of analytical models with SAS applications that support SAS DATA step, DS2, and analytic store score code execution. This document contains the macro’s syntax and argument descriptions, as well as examples.

Note: The models used in the code examples are in the SMM143Samples.zip file. You must import the models into SAS Model Manager before running the code examples. The ZIP file is available at http://support.sas.com/en/software/model-manager-support.html#documentation.

For SAS Model Manager 14.2M1 or later, the macro is delivered with SAS Model Manager software and is available in the sashelp.sasmacr catalog.

Note: The model’s score code type must be DATA step, DS2, or analytic store (ASTORE).

Using the %MM_GetModelDS2Code macro provides these benefits:

• Provides an out-of-the-box macro to convert and export a model’s DS2 score code and its attributes for use by external applications such as SAS Event Stream Processing Studio.

• Reduces errors that could result from manual steps that are required to obtain model DS2 score code and translate it to a format that is compatible with SAS Event Stream Processing.
Dictionary

%MM_GetModelDS2Code Macro
Converts and exports a model’s DS2 score code from the model repository.

Syntax

%mm_getmodelds2code(
  <data=validation-data-table>,
  <sampleSize=sample-size>,
  modelUUID=model-UUID,
  projectUUID=project-UUID,
  ds2pkgname=DS2-package-name,
  outZipFileName=output-ZIP-filename,
  outdir=output-directory-path,
  <trace=OFF | ON>
);

Required Arguments

data=validation-data-table
  The validation data table. A sample of observations is extracted from this table and used to generate a sampleData.csv file in the output ZIP file. For a SAS DATA step model, an additional CSV file that is named sampleDataESP.csv is also included in the output ZIP file.

sampleSize=sample-size
  The number of observations to be included in the sampleData.csv file.

  Default 100

modelUUID=model-UUID
  The UUID of the model. The model UUID is required if the project UUID is not set.

projectUUID=project-UUID
  The UUID of the project. The project UUID is required if the model UUID is not set.
The champion model must also be set for the project.

ds2pkgname=DS2-package-name
  The name of the DS2 package. This argument is used only for a model that has a DATA step score code type.

  Default mm_score_pkg

outZipFileName=output-ZIP-filename
  The filename of the generated output ZIP file. This is a required argument.

  Default ds2score
The output directory for the generated output ZIP file that contains the DS2 score code. This is a required argument.

**Optional Argument**

trace=OFF | ON

Specifies whether to supply verbose trace messages to the SAS log.

Default OFF

**Details**

The `%mm_getmodelds2code` macro enables you to convert and export a model’s DS2 score code from the model repository. The model’s score code type must be DATA step, analytic store (ASTORE), or DS2. The exported model information is placed in a ZIP file. The model information includes the model’s score code, input variables, output variables, and properties.

A different set of files is generated, depending on the score code type and the specified macro arguments. You can also export the DS2 score code for a project champion model. An optional macro argument enables you to include sample data in the output ZIP file.

The contents of the ZIP file can be used in a SAS program or by other SAS applications, such as SAS Event Stream Processing Studio. The SAS Event Stream Processing Studio web application can import the contents of the SAS Model Manager ZIP file or use individual files after they have been extracted.

SAS Event Stream Processing Studio can use the model files in three ways:

- Import a SAS Micro Analytic Service module as an input handler function.
- Import a SAS Embedded Process DS2 score code package as an input handler function.
- Score data using a model that is encoded in a SAS analytic store file.

For more information, see [SAS Event Stream Processing 4.3: Using SAS Event Stream Processing Studio](https://support.sas.com) on support.sas.com.

**Table 8.1 List of Files in the Output ZIP File**

<table>
<thead>
<tr>
<th>Output File</th>
<th>Model Score Code Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputVar.xml</td>
<td>DATA step, DS2, and analytic store</td>
<td>List of model input variables.</td>
</tr>
<tr>
<td>modelProperties.xml</td>
<td>DATA step, DS2, and analytic store</td>
<td>List of model properties.</td>
</tr>
<tr>
<td>outputVar.xml</td>
<td>DATA step, DS2, and analytic store</td>
<td>List of model output variables.</td>
</tr>
<tr>
<td>sampleData.csv</td>
<td>DATA step, DS2, and analytic store</td>
<td>Sample data set in CSV file format. This file can be used for testing the score code in any environment.</td>
</tr>
<tr>
<td>Output File</td>
<td>Model Score Code Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sampleDataESP.csv</td>
<td>DATA step</td>
<td>Sample data set in CSV file format. This file can be used for testing the SAS Micro Analytic Service DS2 score code package in SAS Event Stream Processing Studio.</td>
</tr>
<tr>
<td>score.ep.ds2</td>
<td>DATA step</td>
<td>The model score code that is converted to SAS Embedded Process DS2 score code for the SAS Event Stream Processing procedural window using the SAS Embedded Process.</td>
</tr>
<tr>
<td>score.mas.ds2</td>
<td>DATA step</td>
<td>The model score code that is converted to SAS Micro Analytic Service DS2 package code for the SAS Event Stream Processing procedural window using SAS Micro Analytic Service.</td>
</tr>
<tr>
<td>score.sas</td>
<td>DATA step</td>
<td>The original model score code for DATA step.</td>
</tr>
<tr>
<td>score.ds2</td>
<td>DS2 and analytic store</td>
<td>For a SAS analytic store model, this file is the generated DS2 code that references the SAS analytic store file. For a DS2 score code model, this file contains the custom code entered by the user.</td>
</tr>
<tr>
<td>score.sasast</td>
<td>Analytic store</td>
<td>SAS analytic store file. A SAS analytic store file is an alternative to a SAS DATA step score code file. The SAS analytic store file encodes the details of a complex machine learning model and SAS functions that can read the analytic store file and score new data records. SAS Event Stream Processing Studio can score data using a model encoded in an analytic store file. For more information, see the ASTORE Procedure in SAS Visual Data Mining and Machine Learning Procedures.</td>
</tr>
</tbody>
</table>
Examples

Example 1: Get the Score Code for a DATA Step Model

/* Set up the metadata connection system options. */
options metaPort=8561
metaServer=a123.us.company.com
metaRepository=Foundation
metaUser=myuserID
metaPass=sasuser1;

%mm_getmodelds2code(
data=sampsio.hmeq,
sampleSize=100,
modelUUID=e5e139c1-0a78-0cd4-19dd-82b24a4e3070,
ds2pkgname=mm_score_pkg,
outZipFileName=ds_score,
outdir=c:\sasdemo,
trace=OFF
);

Output 8.1 Contents of the ds_score ZIP File for a DATA Step Model

inputVar.xml
modelProperties.xml
outputVar.xml
sampleData.csv
sampleDataESP.csv
score.ep.ds2
score.mas.ds2
score.sas

Example 2: Get the Score Code for an Analytic Store Model

/* Set up the metadata connection system options. */
options metaPort=8561
metaServer=a123.us.company.com
metaRepository=Foundation
metaUser=myuserID
metaPass=sasuser1;

%mm_getmodelds2code(
data=sampsio.hmeq,
sampleSize=100,
modelUUID=e5d9fcd0-0a78-0cd4-19dd-82b26333886e,
outZipFileName=astore_score,
outdir=c:\sasdemo,
trace=OFF
);
Output 8.2  Contents of the astore_score ZIP File for an Analytic Store Model

```
inputVar.xml
modelProperties.xml
outputVar.xml
sampleData.csv
score.ds2
score.sasast
```

Example 3: Get the Score Code for a DS2 Model
/* Set up the metadata connection system options. */
options metaPort=8561
metaServer=a123.us.company.com
metaRepository=Foundation
metaUser=myuserID
metaPass=sasuser1;

%mm_getmodelds2code(
  data=sampsio.hmeq,
  sampleSize=100,
  modelUUID=e5d0eaf7-0a78-0cd4-19dd-82b212d17315,
  outZipFileName=ds2_score,
  outdir=c:\sasdemo,
  trace=OFF
);

Output 8.3  Contents of the ds2_score ZIP File for a DS2 Model

```
inputVar.xml
modelProperties.xml
outputVar.xml
sampleData.csv
score.ds2
```

Example 4: Get the Score Code for a Project Champion DATA Step Model
/* Set up the metadata connection system options. */
options metaPort=8561
metaServer=a123.us.company.com
metaRepository=Foundation
metaUser=myuserID
metaPass=sasuser1;

%mm_getmodelds2code(
  data=sampsio.hmeq,
  sampleSize=100,
  projectUUID=e526223d-0a78-0cd4-19dd-82b2a06c820c,
  ds2pkgname=mm_score_pkg,
  outZipFileName=champion_score,
  outdir=c:\sasdemo,
  trace=OFF
);
Output 8.4 Contents of the champion_score ZIP File for a Project Champion DATA Step Model

- inputVar.xml
- modelProperties.xml
- outputVar.xml
- sampleData.csv
- sampleDataESP.csv
- score.ep.ds2
- score.mas.ds2
- score.sas
Overview

What Are FCI Macros?

The feature contribution index (FCI) macros enable you to compute the feature contribution indices for interval and nominal predictors, and create an ad hoc report. Before you use these macros, the model outcome must already be available. That is, the input data set must be scored using the model first, and then the model outcomes must be saved in the scoring data set. This document contains the syntax and argument descriptions, as well as examples for the macros.

For SAS Model Manager 14.2M1 or later, the FCI macros are delivered with SAS Model Manager software and are available in the sashelp.sasmacr catalog.

Note: The data sets and code examples can be found in the SAS94_FCI_Macro_Examples.zip file. The ZIP file is available at SAS Model Manager Product Documentation page on support.sas.com.
Measuring Predictor Influence

When you train a model, you can evaluate the importance of predictors within that model. Some training algorithms (for example, decision trees) provide variable importance indices. Alternatively, they can be calculated using statistics on predictors (for example, multiplying –1 by the logarithm of a predictor’s significance value from a regression model). When you deploy a model, you might want to determine how much influence a predictor has on model outcomes. Because the distributions of predictors in a scoring data set might differ from those found in the training data set, the original variable importance metrics obtained from training the model might no longer be relevant.

Scoring Details

Therefore, the procedures used to compute the original variable importance metrics might not work for scoring because the observed target variables that are required by the procedures are not available in the scoring data set. If you face these constraints on the scoring data set, here are possible solutions to help you meet your users’ needs. When you follow these procedures, you do not need to wait for the observed target values to be available. Instead, you can use the predicted values for the interval target variable or the predicted probabilities for the nominal target variable. Instead of customizing procedures for different models, compute the FCI, a model-neutral procedure.

How the Model Outcome Is Determined

The contribution of a predictor or a feature of a model is defined as the aggregated influence of that predictor’s values on the spread of the model outcome. For classification models (nominal target variables), the model outcome consists of the predicted probabilities. For regression models (interval target variables), the model outcome is the predicted value. In both types of models, the model outcome consists of one or more numeric values. To measure the contribution, use the following procedure:

1. For each numeric value in the model outcome, build the main effect analysis of variance with each individual predictor.

2. Measure the contribution of a predictor by the R-squared statistic. (For nominal predictors, this is the full eta-squared statistic. For interval predictors, this is the squared Pearson correlation coefficient.)

Note: For nominal targets, the contribution indices are aggregated for each individual predicted probability.

The index is a numeric value between 0 and 1, inclusive. A value of 1 indicates that the variable contributes the most to the model and is most likely the only variable needed for the model. A value of 0 indicates that the variable contributes the least to the model, and its absence would have little or no impact on the model.

Aggregation for Nominal Targets

There is one FCI for each predicted probability. In order to provide a single index to users, calculate a weighted sum of the individual FCIs.
Here are two common options for weights:

- Each weight is equivalent to the reciprocal of the number of predicted probabilities (uninformative).
- Weights are equivalent to the observed relative frequencies (proportions) of the target categories in the training data set.

For a binary target variable, the choice of weight should not matter because the contribution of a predictor to either predicted probability is the same.

---

**Dictionary**

---

**%Compute_FCI Macro**

Calls the %Compute_FCI_NomPred and the %Compute_FCI_IntPred macros to compute the FCIs’ given input specifications. It overwrites the output FCI data set with the FCIs. Besides using a DATA step statement, this macro calls these procedures: CONTENTS, DATASETS, PRINT, and SORT.

**Syntax**

```
%Compute_FCI ( 
   InData=scoring-dataset-name, 
   TargetSpec=target-specification-dataset, 
   PredictorSpec=predictor-specification-dataset, 
   OutFCIData=output-FCI-dataset, 
   <NameFCI=variable-name-FCI-index>, 
   <WorkLib=work-library-reference>, 
   <Debug=Y | N>>, 
);
```

**Required Arguments**

- **InData**=scoring-dataset-name
  
  Specifies the name of the scoring data set.

- **TargetSpec**=target-specification-dataset
  
  Specifies the name of the data set that contains the target specifications.

- **PredictorSpec**=predictor-specification-dataset
  
  Specifies the name of the data set that contains the predictor specifications.

- **OutFCIData**=output-FCI-dataset
  
  Specifies the name of the output FCI data set.

**Optional Arguments**

- **NameFCI**=variable-name-FCI-index
  
  Specifies the name of the variable that contains the aggregated FCI. The default name is _FCI_.

- **WorkLib**=work-library-reference
  
  Specifies the name of the working library reference. The default library is WORK.
Debug
 Indicates whether to display debugging information. The default value is **N**.

Details
 The following data sets are associated with the %Compute_FCI macro: target specification, predictor specification, and output FCI.

The target specification data set has as many rows as the number of model outcomes. The %Compute_FCI macro looks for the following variables in the data set.

**Table 9.1 Target Specification Data Set**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>String; maximum of 32 characters</td>
<td>SAS Name</td>
<td>Specifies the name of a model outcome.</td>
</tr>
<tr>
<td>PRIOR</td>
<td>Numeric</td>
<td>Number</td>
<td>Specifies the weight for calculating the weighted sum of the individual FCI.</td>
</tr>
</tbody>
</table>

The predictor specification data set has as many rows as the number of predictors. The %Compute_FCI macro looks for the following variables in the data set.

**Table 9.2 Predictor Specification Data Set**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>String; maximum of 32 characters</td>
<td>SAS Name</td>
<td>Specifies the name of a predictor.</td>
</tr>
<tr>
<td>LEVEL</td>
<td>String; maximum of 8 characters</td>
<td>INTERVAL or NOMINAL</td>
<td>Specifies the level for the predictor.</td>
</tr>
<tr>
<td>QMISSNOM</td>
<td>String; maximum of 1 character</td>
<td>N or Y</td>
<td>Specifies whether to include missing values in nominal predictors. This value is ignored if LEVEL is INTERVAL.</td>
</tr>
</tbody>
</table>

The output FCI data set has as many rows as the number of predictors. The following variables are in this data set.

**Table 9.3 Output FCI Data Set**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>VARNAME</em></td>
<td>String; maximum of 32 characters</td>
<td>SAS Name</td>
<td>Specifies the name of a predictor.</td>
</tr>
</tbody>
</table>
User-supplied name | Numeric | Between 0 and 1, inclusive | Users supply a name for the aggregated (weighted sum) FCI.
---|---|---|---

**Example**

Code fragment:

```sas
%Compute_FCI (  
   InData = TESTLIB.ScoreData,  
   TargetSpec = TESTLIB.TargetSpec,  
   PredictorSpec = TESTLIB.PredictorSpec,  
   OutFCIData = TESTLIB.FCIData,  
   NameFCI = _FCI_BAD_,  
   WorkLib = WORK,  
   Debug = N);  
```

For a full example, see “Example 1: Running FCI Macros Using Base SAS or SAS Studio” on page 158.

---

**%Compute_FCI_NomPred Macro**

Computes FCIs for a list of nominal predictors. It appends FCIs to the output FCI data set. Besides using a DATA step statement, this macro calls these procedures: APPEND, DATASETS, PRINT, and SUMMARY.

---

**Syntax**

```sas
%Compute_FCI_NomPred (  
   InData=scoring-dataset-name,  
   DepVarList=list-of-variable-names,  
   NomPredList=list-of-nominal-predictors,  
   OutFCIData=output-FCI-dataset,  
   <qMissNom=Y | N>,  
   <WorkLib=work-library-reference>,  
   <Debug=Y | N>,  
);  
```

---

**Required Arguments**

- **InData=scoring-dataset-name**  
  Specifies the name of the scoring data set.

- **DepVarList=list-of-variable-names**  
  Lists the names of the numeric model output, separated by blanks.

- **NomPredList=list-of-nominal-predictors**  
  Lists the names of nominal predictors, separated by blanks.

- **OutFCIData=output-FCI-dataset**  
  Specifies the name of the output FCI data set.
Optional Arguments

qMissNom
Indicates whether to include missing values in nominal predictors. The default value is N.

WorkLib=work-library-reference
Specifies the name of the working library reference. The default library is WORK.

Debug
Indicates whether to display debugging information. The default value is N.

Example

To determine the FCIs of the nominal predictors, call the %Compute_FCI_NomPred macro directly. Here is the SAS code for the badloans_monitor_4 data set.

```sas
data badloans_score;
  set MMData.badloans_monitor_4;
  %include _LGRSCR_;
run;
%

%Compute_FCI_NomPred
  (InData = badloans_score,
   DepVarList = %str(P_BAD0 P_BAD1),
   IntPredList = &NomPred.,
   qMissNom = N,
   OutFCIData = OutFCIData_4_Nom,
   WorkLib = WORK,
   Debug = &Debug.
  )
;

proc print data = OutFCIData_4_Nom;
run;
```

For a full example that includes output results, see “Example 1: Running FCI Macros Using Base SAS or SAS Studio” on page 158.

%Compute_FCI_IntPred Macro

Computes FCIs for a list of interval predictors. It appends FCIs to the output FCI data set. Besides using a DATA step statement, this macro calls the APPEND procedure.

Syntax

```
%Compute_FCI_IntPred (InData=scoring-dataset-name,
DepVarList=list-of-variable-names,
IntPredList=list-of-interval-predictors,
OutFCIData=output-FCI-dataset,
<WorkLib=work-library-reference>,
<Debug=Y | N>,
);
```
**Required Arguments**

- **InData=** *scoring-dataset-name*
  - Specifies the name of the scoring data set.

- **DepVarList=** *list-of-variable-names*
  - Contains a list of names of the numeric model output, separated by blanks.

- **IntPredList=** *list-of-interval-predictors*
  - Contains a list of names of interval predictors, separated by blanks.

- **OutFCIData=** *output-FCI-dataset*
  - Specifies the name of the output FCI data set.

**Optional Arguments**

- **WorkLib=** *work-library-reference*
  - Specifies the name of the working library reference. The default library is WORK.

- **Debug**
  - Indicates whether to display debugging information. The default value is `N`.

**Example**

To determine the FCIs of the interval predictors, call the `%Compute_FCI_IntPred` macro directly. Here is the SAS code for the `badloans_monitor_4` data set.

```sas
data badloans_score;
  set MMData.badloans_monitor_4;
  %include _LGRSCR_
  run;

%Compute_FCI_IntPred
  (InData = badloans_score,
   DepVarList = %str(P_BAD0 P_BAD1),
   IntPredList = &IntPred.,
   OutFCIData = OutFCIData_4_Int,
   WorkLib = WORK,
   Debug = &Debug.
  );

proc print data = OutFCIData_4_Int;
  run;
```

For a full example that includes output results, see “Example 1: Running FCI Macros Using Base SAS or SAS Studio” on page 158.

---

**%MM_AdHocReport_FCI Macro**

Generates an ad hoc report in SAS Model Manager. It calls the `%Compute_FCI` macro to compute the FCIs for a list of scoring data sets. Besides using a DATA step statement and procedures that are used in the macros called, this macro also calls these procedures: APPEND, DELETE, PRINT, SGPANEL, SGPLOT, SQL, and TABULATE.
Syntax

```%MM_AdHocReport_FCI (  
    ScoreDataPrefix=scoring-dataset-prefix-name,  
    P_VarList=list-of-predicted-outcome-variables,  
    <ScoreDataCount=scoring-dataset-count>,  
    <QMissNomPred=Y | N>,  
    <Debug=Y | N>,  
);```

**Required Arguments**

**ScoreDataPrefix**=`scoring-dataset-prefix-name`
- Specifies the name of the common prefix (including the library name) of scoring data set names.

**P_VarList**=`list-of-predicted-outcome-variables`
- Contains a list of model outcome variables.

**Optional Arguments**

**ScoreDataCount**=`scoring-dataset-count`
- Specifies the number of scoring data sets. A positive numeric value is expected. The default value is 1.

**QMissNomPred**
- Indicates whether to include missing values in nominal predictors. The default value is N.

**Debug**
- Indicates whether to display debugging information. The default value is N.

**Details**

This macro assumes that the scoring data sets are named according to the convention specified by &ScoreDataPrefix.k, where k is an integer from 1 to &ScoreDataCount without any gaps. It reads the following XML files whose actual locations are pointed to by the SAS Model Manager macro variables.

<table>
<thead>
<tr>
<th>Table 9.4 XML Files Associated with MM_AdHocReport_FCI Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML File</td>
</tr>
<tr>
<td>InputVar.xml</td>
</tr>
<tr>
<td>OutputVar.xml</td>
</tr>
</tbody>
</table>

**Example: %MM_AdHocReport_FCI Macro Code Example**

```%let MyLib = %str(c:\FCI);  
libname TESTLIB "&MyLib.\Data";```
%NAME% %MM_AdHocReport_FCI Macro

/* Specify the folders that the tables and the charts are to be written to. */
ods html path = "&MyLib." gpath = "&MyLib.";

/* Load the macros inside this catalog for displaying the report. */
filename mmreport catalog "sashelp.modelmgr.reportexportmacros.source";
%include mmreport;
%m_exportReportsBegin (fileName = BadLoan_FCI_Report,
                reportFormat = HTML,
                reportstyle = seaside);

/* Create reports for the four data sets with scores. */
/* Include missing values for nominal predictors in the calculations. */
%M_AdHocReport_FCI (ScoreDataPrefix = %str(TESTLIB.BadLoans_Monitor_Scoring_),
              ScoreDataCount = 4,
              P_VarList = %str(P_BAD0 P_BAD1),
              QMissNomPred = N,
              Debug = N);

%m_exportReportsEnd(reportFormat = HTML);
/* End of the ad hoc report. */

The MM_AdHocReport_FCI macro generates one table and two charts:

- The table has two variables in the row dimension and one variable in the column dimension:
  - The two variables in the row dimension are names of predictors and their measurement levels.
  - The variable in the column dimension is the scoring data set sequential number (the integer k).
  - The cell contents are the FCI displayed in percentage format.

- The first chart is a panel series chart:
  - Each predictor name constitutes one panel.
  - Each series plots the FCIs versus the scoring data set sequential number.

- The second chart is an overlay series chart:
  - Each scoring data set sequential number constitutes one series.
  - Each series plots the FCIs versus names of the predictors.

For a full example that includes output results, see “Example 2: Running FCI Macros Using SAS Model Manager” on page 170.
Examples

Example 1: Running FCI Macros Using Base SAS or SAS Studio

Details

The %Compute_FCI macro assists in calculating FCIs. In addition to specifying the data set on which a model is deployed, this macro requires a specification data set for the target variable and another for the predictors. The target specification data set should have two columns (NAME and PRIOR), and as many rows as the number of levels of the target variable (it is assumed that an interval target variable has a single level). The predictor specification data set should have three columns (NAME, LEVEL, and QMISSNOM), and as many rows as the number of predictors. For more information, see “%Compute_FCI Macro” on page 151.

Note: The paths that you specify in your example code for the MMLib macro variable, as well as the MMData and MMPub libraries, should be the same as where the macros are installed. For example, in the code examples below, the Windows directory path for the data tables is c:\FCI\Data.

To run this example:

1. Specify the macro variables Debug and MMLib.
2. Set the Debug argument to Y, to request the debugging information from the macros.
3. Set the MMLib macro variable to the upper-level folder where you want to store your work.
4. Create a library named MMData and set the directory path to where the FCI data tables are located (for example, c:\FCI\Data).
5. Create a folder named Publication within the FCI upper-level folder, and then create a library named MMPub that points to the folder that you created.

Because the ad hoc report contains charts, the PATH and GPATH folders must be defined using the ODS HTML command, and Write permissions must exist for these two folders. It is recommended that the ODS GRAPHICS command be used to specify the WIDTH and the HEIGHT to be larger values because the panel chart can have as many panels as the number of predictors.

Compute FCIs

%let Debug = N;
%let MMLib = %str(c:\FCI);

libname MMData "&MMLib.\Data";
libname MMPub "&MMLib.\Publication";

/* Here you specify the rows in the predictor specification data set */
/* that are for the interval predictors. */
/* Only the names are specified. The other two columns are specified later. */
data MMPub.IntPredSpec;
   length NAME $ 32;
   input NAME $;
   datalines;
    CLAGE
    CLNO
    DELINQ
    DEROG
    MORTDUE
    NINQ
    YOJ

   /* Here you specify the rows in the predictor specification data set */
   /* that are for the nominal predictors. */
   /* Only the names are specified. The other two columns are specified later. */

data MMPub.NomPredSpec;
   length NAME $ 32;
   input NAME $;
   datalines;
    JOB
    REASON
    REASON

   /* The names are extracted into the respective macro variables. */
   /* These macro variables help you write a more elegant syntax */
   /* for building the logistic model. */

proc sql noprint;
   select NAME into :IntPred separated by ' ' from MMPub.IntPredSpec;
   select NAME into :NomPred separated by ' ' from MMPub.NomPredSpec;
quit;

   /* The logistic model is built using the HPLOGISTIC procedure, */
   /* which is a high-performance statistical procedure. */
   /* It requires a license for SAS High-Performance Statistics. */
   /* The procedure writes a SAS program code file that is used */
   /* for scoring the incoming data sets. */

filename _LGRSCR_ "&MMLib.\Publication\Badloans_Logistic_Score.sas";
proc hplogistic data = MMData.badloans_train
   maxiter = 100
   technique = newrap
   nameilen = 128;
   class &NomPred. / param = glm order = freq descending;
   model BAD (event = '1') = &IntPred. &NomPred. / link = logit rsquare association;
   selection method = stepwise;
   code file = _LGRSCR_;
run;

   /* The predictor specification data set is generated using the individual parts. */
   /* At the same time, the other two columns, LEVEL and QMISSNOM, */
   /* are populated with the appropriate values. */
data MMPub.PredictorSpec;
set MMPub.IntPredSpec (in = in1)
    MMPub.NomPredSpec (in = in2);

length LEVEL $ 16;
length QMISSNOM $ 1;

if (in1) then
do;
    LEVEL = 'INTERVAL';
    QMISSNOM = ' ';
end;
else if (in2) then
do;
    LEVEL = 'NOMINAL';
    QMISSNOM = 'N';
end;
run;

/* The predictor specification data set is printed so */
/* that you can see its contents. */

proc print data = MMPub.PredictorSpec;
run;

/* The target specification data set is generated from the output of the */
/* FREQ procedure because you want the weights to be proportional */
/* to the frequencies of the target values. */
/* The NAME column contains P_BAD0 and P_BAD1, */
/* which are the predicted probabilities of BAD = 0 and BAD = 1, */
/* respectively. These two probabilities are the model outcomes. */

proc freq data = MMData.badloans_train;
    table BAD / noprint out = BAD_Percent;
run;

data MMPub.TargetSpec;
set BAD_Percent;

length NAME $ 32;
length PRIOR 8;

NAME = cat('P_BAD', strip(vvalue(BAD)));
PRIOR = PERCENT / 100;

keep NAME PRIOR;
run;

/* The target specification data set is printed so that you can see its contents. */

proc print data = MMPub.TargetSpec;
run;

/* The data set is first scored by including the SAS program code */
/* from the HPLOGISTIC procedure. */
/* Next, the %Compute_PCI macro is executed by specifying the appropriate arguments. */
/* The FCIs are calculated for the badloans_train data set as the benchmark. */

data badloans_score;
    set MMData.badloans_train;
    %include _LGRSCR_;
run;

%Compute_FCI
{
    InData = badloans_score,
    TargetSpec = MMPub.TargetSpec,
    PredictorSpec = MMPub.PredictorSpec,
    OutFCIData = OutFCIData_0,
    NameFCI = %str(_FCI_),
    WorkLib = WORK,
    Debug = &Debug.
};

/* Calculate the FCIs for the badloans_monitor_1 data set. */

data badloans_score;
    set MMData.badloans_monitor_1;
    %include _LGRSCR_;
run;

%Compute_FCI
{
    InData = badloans_score,
    TargetSpec = MMPub.TargetSpec,
    PredictorSpec = MMPub.PredictorSpec,
    OutFCIData = OutFCIData_1,
    NameFCI = %str(_FCI_),
    WorkLib = WORK,
    Debug = &Debug.
};

/* Calculate the FCIs for the badloans_monitor_2 data set. */

data badloans_score;
    set MMData.badloans_monitor_2;
    %include _LGRSCR_;
run;

%Compute_FCI
{
    InData = badloans_score,
    TargetSpec = MMPub.TargetSpec,
    PredictorSpec = MMPub.PredictorSpec,
    OutFCIData = OutFCIData_2,
    NameFCI = %str(_FCI_),
    WorkLib = WORK,
    Debug = &Debug.
};

/* Calculate the FCIs for the badloans_monitor_3 data set. */
data badloans_score;
    set MMData.badloans_monitor_3;
    %include _LGRSCR_
run;

%Compute_FCI
{
    InData = badloans_score,
    TargetSpec = MMPub.TargetSpec,
    PredictorSpec = MMPub.PredictorSpec,
    OutFCIData = OutFCIData_3,
    NameFCI = %str(_FCI_),
    WorkLib = WORK,
    Debug = &Debug.
};

/* Calculate the FCI s for the badloans_monitor_4 data set. */

data badloans_score;
    set MMData.badloans_monitor_4;
    %include _LGRSCR_
run;

%Compute_FCI
{
    InData = badloans_score,
    TargetSpec = MMPub.TargetSpec,
    PredictorSpec = MMPub.PredictorSpec,
    OutFCIData = OutFCIData_4,
    NameFCI = %str(_FCI_),
    WorkLib = WORK,
    Debug = &Debug.
};

/* Combine all the results into one data set indexed by their times */
data MMPub.badloans_FCI_Result;
    set OutFCIData_0 (in = in0)
OutFCIData_1 (in = in1)
OutFCIData_2 (in = in2)
OutFCIData_3 (in = in3)
OutFCIData_4 (in = in4);
    if (in0) then _TIME_ = 0;
else if (in1) then _TIME_ = 1;
else if (in2) then _TIME_ = 2;
else if (in3) then _TIME_ = 3;
else if (in4) then _TIME_ = 4;
run;

/* Bring the measurement levels of the predictors in from the */
/* predictor specification data set. */
proc sort data = MMPub.badloans_FCI_Result;
    by _VARNAME_ _TIME_;
run;

proc sort data = MMPub.PredictorSpec;
    by NAME;
run;

data MMPub.badloans_FCI_Result;
    merge MMPub.badloans_FCI_Result (in = in0 rename = (_VARNAME_ = NAME))
        MMPub.PredictorSpec (in = in1);
    by NAME;
    keep NAME LEVEL _FCI_ _TIME_;
run;

/* Review and compare the FCIs. */

title2 "Feature Contribution Indices";
ods graphics / reset height = 7in;

proc tabulate data = MMPub.badloans_FCI_Result;
    class NAME LEVEL _TIME_;
    var _FCI_
    table LEVEL='Measurement Level' * NAME='Predictor',
            _TIME_='Monitor Time' * (sum = '' * f = percent10.1) * _FCI_ = '';
    format _FCI_ 10.4;
run;

proc sgpanel data = MMPub.badloans_FCI_Result;
    panelby NAME / onepanel novarname;
    series y = _FCI_ x = _TIME_ / markers;
    rowaxis grid label = 'Feature Contribution Index';
    colaxis grid integer label = 'Monitor Time';
run;

proc sgplot data = MMPub.badloans_FCI_Result;
    series y = _FCI_ x = NAME / markers group = _TIME_ name = 'series';
    keylegend 'series' / location = outside position = right title = 'Monitor Time';
    yaxis grid label = 'Feature Contribution Index';
    xaxis grid label = 'Predictor';
run;

/* The results indicate that the predicted probabilities are collectively */
/* correlated with the DELINQ variable. However, it substantially drops its */
/* contribution to the predicted probabilities in the badloans_monitor_3. */
/* You then can use the histograms to visually compare the distribution of DELINQ. */

proc sgplot data = MMData.badloans_train;
    histogram DELINQ / binwidth = 1 scale = percent;
    xaxis values = (0 to 20 by 1) offsetmin = 0.05 offsetmax = 0.05;
    yaxis values = (0 to 100 by 10) grid;
    footnote MMData.badloans_train;
run;

proc sgplot data = MMData.badloans_monitor_1;
    histogram DELINQ / binwidth = 1 scale = percent;
    xaxis values = (0 to 20 by 1) offsetmin = 0.05 offsetmax = 0.05;
yaxis values = (0 to 100 by 10) grid;
footnote MMData.badloans_monitor_1;
run;

proc sgplot data = MMData.badloans_monitor_2;
  histogram DELINQ / binwidth = 1 scale = percent;
  xaxis values = (0 to 20 by 1) offsetmin = 0.05 offsetmax = 0.05;
  yaxis values = (0 to 100 by 10) grid;
  footnote MMData.badloans_monitor_2;
run;

proc sgplot data = MMData.badloans_monitor_3;
  histogram DELINQ / binwidth = 1 scale = percent;
  xaxis values = (0 to 20 by 1) offsetmin = 0.05 offsetmax = 0.05;
  yaxis values = (0 to 100 by 10) grid;
  footnote MMData.badloans_monitor_3;
run;

proc sgplot data = MMData.badloans_monitor_4;
  histogram DELINQ / binwidth = 1 scale = percent;
  xaxis values = (0 to 20 by 1) offsetmin = 0.05 offsetmax = 0.05;
  yaxis values = (0 to 100 by 10) grid;
  footnote MMData.badloans_monitor_4;
run;

Compute the FCI for Interval Predictors and Nominal Predictors

/* Suppose you want to see the unweighted indices for the */
/* badloans_monitor_4 data set, or suppose that you want to skip */
/* the steps in creating the specification data sets. */
data badloans_score;
  set MMData.badloans_monitor_4;
  %include _LGRSCR_;
run;

%Compute_FCI_IntPred
{
  InData = badloans_score,
  DepVarList = %str(P_BAD0 P_BAD1),
  IntPredList = &IntPred.,
  OutFCIData = OutFCIData_4_Int,
  WorkLib = WORK,
  Debug = &Debug.
};

%Compute_FCI_NomPred
{
  InData = badloans_score,
  DepVarList = %str(P_BAD0 P_BAD1),
  NomPredList = &NomPred.,
  qMissNom = N,
  OutFCIData = OutFCIData_4_Nom,
  WorkLib = WORK,
  Debug = &Debug.
};
/* The output FCI data sets are printed so that you can see the output results. */

proc print data = OutFCIData_4_Int;
run;

proc print data = OutFCIData_4_Nom;
run;

Here are the output results that include the observations for the predictor specification data set, the observations for the target specification data set, the measurement level for the interval predictors and nominal predictors, and the calculated FCIs.

**Output 9.1  Predictor Specification Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>LEVEL</th>
<th>QMISSNOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLAGE</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CLNO</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DELINQ</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DEROQ</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MORTDUE</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NINO</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>YOJ</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>JOB</td>
<td>NOMINAL</td>
<td>N</td>
</tr>
<tr>
<td>9</td>
<td>REASON</td>
<td>NOMINAL</td>
<td>N</td>
</tr>
</tbody>
</table>

**Output 9.2  Target Specification Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>PRIOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P_BAD0</td>
<td>0.79766</td>
</tr>
<tr>
<td>2</td>
<td>P_BAD1</td>
<td>0.20212</td>
</tr>
</tbody>
</table>

**Output 9.3  FCI Measurement Levels for Internal and Nominal Predictors**

![Feature Contribution Indices Table]
**Output 9.4**  FCI Plotted against the Monitoring Time for Each Predictor

![Diagram showing FCI plotted against the monitoring time for each predictor.](image)

**Output 9.5**  FCI Plotted against Predictors for Each Monitoring Time

![Diagram showing FCI plotted against predictors for each monitoring time.](image)
For the histograms that are shown below, the horizontal axes are the values of DELINQ. The vertical axes are the percent of observations. The histograms correspond to the _TIME_ variable in the row-major order. The first histogram is for _TIME_ = 0 (in other words, the benchmark). The other histograms are for _TIME_ = 1, 2, 3, and 4.

The histogram that corresponds to _TIME_ = 3 shows that DELINQ = 0 for almost all of the observations. In other words, the DELINQ variable seems to be constant in the badloans_monitor_3 data set. Therefore, do not expect its correlations with the two predicted probabilities in the badloans_monitor_3 data set to be as high as those in other data sets. If you do not have the badloans_monitor_4 data set, then you can conclude that the model is stale and a refresh is necessary. Because the badloans_monitor_4 data set is present and the FCI goes up again, you can safely conclude that the findings in the badloans_monitor_3 data set are due to spurious fluctuation and the time to rebuild the model has not yet come.

**Output 9.6  FCI of the DELINQ Predictor versus Percent of Observations for the badloans_train Data Set**

![Feature Contribution Indices](image)
Output 9.7  FCI of the DELINQ Predictor versus Percent of Observations for the badloans_monitor_1 Data Set

Output 9.8  FCI of the DELINQ Predictor versus Percent of Observations for the badloans_monitor_2 Data Set
Output 9.9  FCI of the DELINQ Predictor versus Percent of Observations for the badloans_monitor_3 Data Set

Output 9.10  FCI of the DELINQ Predictor Versus Percent of Observations for the badloans_monitor_4 Data Set
Example 2: Running FCI Macros Using SAS Model Manager

Details

Overview

Classification models can be used to predict the likelihood of default of loan applications. These models implement the logistic regression algorithm that computes this probability. In this example, the algorithm is trained using the badloans_train data set. The target variable is BAD and has two values (0 and 1). If a loan application results in default, BAD = 1 (otherwise, BAD = 0). After assessing business value and satisfying legal requirements, consider the following predictors:

- Interval predictors:
  - CLAGE
  - CLNO
  - DELINQ
  - DEROG
  - MORTDUE
  - NINQ
  - YOJ

- Nominal predictors:
  - JOB
  - REASON
The final logistic model includes all of these predictors with the exception of REASON. This model is then used to score (that is, calculate the probability of) incoming observations regarding loan default according to the deployment schedule (for example, monthly, weekly, daily, or hourly).

The incoming observations are collected in these data sets:

- badloans_monitor_1
- badloans_monitor_2
- badloans_monitor_3
- badloans_monitor_4

Because the financial profiles of loan applicants change constantly, the classification must be regularly maintained in order to ensure prediction accuracy. On the other hand, the model should not capture any spurious signals that are due to temporary fluctuations of the financial market. Thus, after each data set is scored, the FCIs of the predictors are calculated and the indices are compared visually across the four data sets. The indices of the training data set are also calculated as a benchmark. If the index of a predictor substantially increases or decreases at a particular time, this indicates that the predictor might contribute to the model more or less than the normal level. This change might be due to an unexpected variation of the predictor’s distribution at that time. If the model is refreshed, the findings can help support a particular decision. For more information, see “%MM_AdHocReport_FCI Macro” on page 155.

Import and Score Models

After you have used your preferred tool to build the logistic regression model, you must import and score your model using the SAS Model Manager web application before you can run the %MM_AdHocReport_FCI macro. Sample scoring data sets are included in the SAS94_FCI_Macro_Examples.zip file, which is available on the SAS Model Manager Product Documentation page on support.sas.com. For more information, see “Importing Models” in SAS Model Manager: User’s Guide and “Overview of Scoring Tests” in SAS Model Manager: User’s Guide.

Create an Ad Hoc Report

1. On the Reports page of your project, create an ad hoc report, and name it BadLoan_FCI_Report.
2. Select the model that you previously imported.
3. After copying the following SAS code into the SAS Editor, click Run.

```sas
%let MyLib = %str(c:\FCI);

libname TESTLIB "&MyLib.\Data";
/* Specify the folders that the tables and the charts are to be written to. */
ods html path = "&MyLib." gpath = "&MyLib.";

/* Load the macros inside this catalog in order to display the report */
filename mmreport catalog "sashelp.modelmgr.reportexportmacros.source";
%include mmreport;
%mm_exportReportsBegin (
   fileName = BadLoan_FCI_Report,
   reportFormat = HTML,
   reportstyle = seaside);
```
Create reports for the four data sets with scores. */
/* Include missing values for nominal predictors in the calculations. */
%M_MMM_AdHocReport_FCI (ScoreDataPrefix = %str(TESTLIB.BadLoans_Monitor_Scoring_), ScoreDataCount = 4, P_VarList = %str(P_BAD0 P_BAD1), QMissNomPred = N, Debug = N);

%M_Mmm_exportReportsEnd(reportFormat = HTML);
/* End of the ad hoc report. */

For more information, see “Ad Hoc Reports” in SAS Model Manager: User’s Guide.

After you have successfully run the ad hoc report code, open the report to review the results. The target specification data set is printed below. The %MM_AdHocReport_FCI macro assigns equal priors for the values that are the reciprocal of the number of model outcome variables (which is 2 in this example).

Output 9.13 Target Specification Data Set and FCI Measurement Levels for Internal and Nominal Predictors

Model Outcome Variables and Their Priors

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>PRIOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P_BAD0</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>P_BAD1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Feature Contribution Indices

<table>
<thead>
<tr>
<th>Measurement Level</th>
<th>Predictor</th>
<th>Monitor Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>CLAGE</td>
<td>0.9%</td>
</tr>
<tr>
<td></td>
<td>CLNO</td>
<td>0.7%</td>
</tr>
<tr>
<td></td>
<td>DEBTINC</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>DELINQ</td>
<td>47.5%</td>
</tr>
<tr>
<td></td>
<td>DEROG</td>
<td>31.3%</td>
</tr>
<tr>
<td></td>
<td>LOAN</td>
<td>0.5%</td>
</tr>
<tr>
<td></td>
<td>MORTDUE</td>
<td>2.0%</td>
</tr>
<tr>
<td></td>
<td>NINQ</td>
<td>13.8%</td>
</tr>
<tr>
<td></td>
<td>VALUE</td>
<td>0.3%</td>
</tr>
<tr>
<td></td>
<td>YOJ</td>
<td>0.1%</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>JOB</td>
<td>4.0%</td>
</tr>
<tr>
<td></td>
<td>REASON</td>
<td>0.3%</td>
</tr>
</tbody>
</table>
The report indicates that the variable DELINQ contributes relatively less at Monitor Time 3 than at other times. You should act on this finding to compare the distributions of
DELINQ across the four times. Your goal is to determine whether the finding is due to spurious fluctuation.

Output 9.15  FCI Plotted against Predictors for Each Monitoring Time
Chapter 10
Using SAS Viya Models with SAS Model Manager

Overview
This document shows how to use SAS Studio to create models with SAS Visual Data Mining and Machine Learning procedures and then use them in SAS Model Manager 14.3. With the integration between SAS 9.4M6 and SAS Viya 3.4, you can run all of the included code examples from SAS Studio on SAS 9.4. First, you see how the procedures can be used to generate the SAS score code, input variables, and output variables that you need for a model. The document also explains how to create a SAS model package (SPK) file and register the model in the SAS Model Manager model repository. Then you can see the models in the SAS Model Manager web application.

Note that these procedures run on SAS Viya. When these procedures create SAS DATA step code, the code can be used with SAS Model Manager 14.3 on SAS 9.4M6 or later. For more information, see the SAS Viya Documentation page on support.sas.com.

Prerequisites for Running Code Examples
The code examples in this document are run on SAS Viya in a UNIX environment. Directory paths, servers, ports, and data set location depend on your environment and might need to be modified. To run the examples:

- Configure a SAS 9.4 client to work with SAS Viya.
- Obtain the values for the CASHOST= and CASPORT= system options from a SAS Administrator.
Configure a SAS 9.4 Client to Work with SAS Viya

Here is a way to configure a SAS 9.4 client to work with SAS Viya:

Note: This configuration must be performed by a system administrator. Contact your system administrator for assistance.

1. Copy the SAS Viya CA certificates (vault-ca.crt file) from the SAS Viya host to a location on your SAS 9.4 deployment where you can access the certificates. The SAS Viya CA certificates can be found in the directory /opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts.

   Note: The vault-ca.crt file contains two certificates. The first certificate is the SAS Viya root CA certificate issued by SAS Secrets Manager, and the second certificate is the SAS Viya intermediate CA certificate issued by SAS Secrets Manager.

2. Split the certificates within the value-ca.crt file into two separate files.

3. Use SAS Deployment Manager to add your certificates to the trusted CA bundle. Be sure to add the SAS Viya root CA certificate first.

For more information and for the most recent configuration options, see “Configure SAS 9.4 Clients to Work with SAS Viya” in Encryption in SAS Viya: Data in Motion.

Create a Model in SAS Viya

You can use SAS Studio on SAS 9.4 to run the code that creates a SAS Viya model and its score code. The training data set must be in a CAS library, and users must be authorized to connect to a session on the SAS Cloud Analytic Services (CAS) server. For more information, see SAS Cloud Analytic Services: Fundamentals and Getting Started with SAS Visual Data Mining and Machine Learning.

Generate SAS Score Code for a Model

The LOGSELECT procedure fits binary and binomial response models in SAS Viya. The procedure creates a SAS code file that can be used for scoring new data. This example demonstrates how to build a model, generate the SAS score code file, and create a sample of the training data set that is used when registering the model.

Note: For information about starting a CAS session and loading data, see SAS Viya 3.4 Quick Start.

```sas
/* Start a new CAS session. */
options cashost="myserver.com" casport=5570;
cas casauto;
```
/* Create macro variables. */
%let training_dataset=%quote(hmeq_train);
%let model_name1=logselect_badloans;

/**********************************************************
  Generate SAS Code for a Model
**********************************************************/

/* Create a libref that points to where your training data set is stored. */
libname qstut '~/QSTutorial/Data';

/* Load the training data set into the caslib CASUSER in session memory. */
proc casutil;
  load data=qstut.&training_dataset outcaslib=casuser replace;
  list tables;
quit;

/* Assign a libref with the CAS LIBNAME engine to access the in-memory table. */
libname mycas cas caslib=casuser;

/* Create a SAS Viya model file and a model output table. */
proc logselect data=mycas.&training_dataset;
  class job reason;
  model bad=delinq derog ninq clage clno debtinc loan mortdue value yoj job reason;
  partition frac(valid=0.2 seed=78900);
  code file="~/&model_name1..sas" pcatall replace=true;
  output out=mycas.&model_name1._output copyvars=(_all_) role;
run;
Output 10.1  Generated SAS Scoring Code

```sas
/*-----------------------------------------------
  Generated SAS Scoring Code
  Date: 26Jul2019:13:33:55
  -----------------------------------------------*/

drop _badval_ _linp_ _temp_ _i_ _j_;
_badval_ = 0;
_linp_   = 0;
_temp_   = 0;
_i_      = 0;
_j_      = 0;
drop MACLOGBIG;
MACLOGBIG= 7.0978271289338392e+02;

array _xrow_0_0_{19} _temporary_;
array _beta_0_0_{19} _temporary_ ( 4.84117022019437
  -0.75527303150374
  -0.75489831554816
  -0.0837826649534
  0.00551538465093
  0.02704356320614
  -0.10305346531346
  0.00001254508488
  -3.1825025509978E-7
  -1.1927173270349E-6
  -0.00122031322848
  0.78216672761072
  1.05461905358989
  0.80400513782685
  0.27217114260678
  -0.75976952372647
  0
  -0.45642539133325
  0);

length _JOB_ $7; drop _JOB_;
_JOB_ = left(trim(put(JOB,$7.)));
length _REASON_ $7; drop _REASON_;
_REASON_ = left(trim(put(REASON,$7.)));
if missing(CLNO)
  or missing(NINQ)
  or missing(MORTDUE)
  or missing(DEROG)
  or missing(VALUE)
  or missing(DELINQ)
  or missing(LOAN)
  or missing(YOJ)
  or missing(DEBTINC)
  or missing(CLAGE)
  then do;
    _badval_ = 1;
    goto skip_0_0;
end;
```
do _i_=1 to 19; _xrow_0_0_{_i_} = 0; end;

_xrow_0_0_[1] = 1;
_xrow_0_0_[2] = DELINQ;
_xrow_0_0_[3] = DEROG;
_xrow_0_0_[4] = NINQ;
_xrow_0_0_[5] = CLAGE;
_xrow_0_0_[6] = CLNO;
_xrow_0_0_[7] = DEBTINC;
_xrow_0_0_[8] = LOAN;
_xrow_0_0_[9] = MORTDUR;
_xrow_0_0_[10] = VALUE;
_xrow_0_0_[11] = YOJ;

_temp_ = 1;
select (_JOB_);
  when ('Mgr') _xrow_0_0_[12] = _temp_;
  when ('Office') _xrow_0_0_[13] = _temp_;
  when ('Other') _xrow_0_0_[14] = _temp_;
  when ('ProfExe') _xrow_0_0_[15] = _temp_;
  when ('Sales') _xrow_0_0_[16] = _temp_;
  when ('Self') _xrow_0_0_[17] = _temp_;
  otherwise do; _badval_ = 1; goto skip_0_0; end;
end;

_temp_ = 1;
select (_REASON_);
  when ('DebtCon') _xrow_0_0_[18] = _temp_;
  when ('HomeImp') _xrow_0_0_[19] = _temp_;
  otherwise do; _badval_ = 1; goto skip_0_0; end;
end;

do _i_=1 to 19;
  _linp_ + _xrow_0_0_{_i_} * _beta_0_0_{_i_};
end;

skip_0_0:
length I_BAD $1;
label I_BAD = 'Into: BAD';
array _levels_0_{2} $ 1 _TEMPORARY_ ('0', '1');
label P_BAD0 = 'Predicted: BAD=0';
if (_badval_ eq 0) and not missing(_linp_) then do;
  if (_linp_ > 0) then do;
    P_BAD0 = 1 / (1+exp(-_linp_));
  end; else do;
    P_BAD0 = exp(_linp_) / (1+exp(_linp_));
  end;
P_BAD1 = 1 - P_BAD0;
if P_BAD0 >= 0.5 then do;
  I_BAD = _levels_0_{1};
end; else do;
  I_BAD = _levels_0_{2};
end;
end; else do;
  _linp_ = .;
  P_BAD0 = .;
  P_BAD1 = .;
  I_BAD = ' ';
The LOGSELECT Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Response Variable</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Link Function</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Seed</td>
</tr>
<tr>
<td>Predicted Response Level</td>
</tr>
</tbody>
</table>

Number of Observations

<table>
<thead>
<tr>
<th>Description</th>
<th>Total</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>2381</td>
<td>1880</td>
<td>501</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>1364</td>
<td>1077</td>
<td>287</td>
</tr>
</tbody>
</table>

Response Profile

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>BAD</th>
<th>Total Frequency</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1243</td>
<td>982</td>
<td>201</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>121</td>
<td>95</td>
<td>26</td>
</tr>
</tbody>
</table>

Probability modeled is \( \text{BAD} = 0 \).

Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>6</td>
<td>Mgr, Office, Other, Prof, Exe, Sales, Self</td>
</tr>
<tr>
<td>REASON</td>
<td>2</td>
<td>Debt, Con, Home, Imp</td>
</tr>
</tbody>
</table>

Convergence criterion (GCONV=1E-8) satisfied.
### Create a Model in SAS Viya

#### Dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns in Design</td>
<td>19</td>
</tr>
<tr>
<td>Number of Effects</td>
<td>13</td>
</tr>
<tr>
<td>Max Effect Columns</td>
<td>6</td>
</tr>
<tr>
<td>Rank of Design</td>
<td>17</td>
</tr>
<tr>
<td>Parameters in Optimization</td>
<td>17</td>
</tr>
</tbody>
</table>

#### Testing Global Null Hypothesis: BETA=0

<table>
<thead>
<tr>
<th>Test</th>
<th>DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood Ratio</td>
<td>16</td>
<td>149.8225</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

#### Fit Statistics

<table>
<thead>
<tr>
<th>Description</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2 Log Likelihood</td>
<td>492.87085</td>
<td>148.44390</td>
</tr>
<tr>
<td>AIC (smaller is better)</td>
<td>526.87085</td>
<td>182.44390</td>
</tr>
<tr>
<td>AICC (smaller is better)</td>
<td>527.44875</td>
<td>184.71899</td>
</tr>
<tr>
<td>SBC (smaller is better)</td>
<td>611.55374</td>
<td>244.65509</td>
</tr>
<tr>
<td>Average Square Error</td>
<td>0.06038</td>
<td>0.06846</td>
</tr>
<tr>
<td>-2 Log L (Intercept-only)</td>
<td>842.69336</td>
<td>174.44212</td>
</tr>
<tr>
<td>R-Square</td>
<td>0.12987</td>
<td>0.08660</td>
</tr>
<tr>
<td>Max-rescaled R-Square</td>
<td>0.28898</td>
<td>0.19015</td>
</tr>
<tr>
<td>McFadden's R-Square</td>
<td>0.23312</td>
<td>0.14904</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.07242</td>
<td>0.08014</td>
</tr>
<tr>
<td>Difference of Means</td>
<td>0.23601</td>
<td>0.16968</td>
</tr>
</tbody>
</table>
Registering Models in SAS Model Manager

You can use the %AA_Model_Register macro to create a SAS model package (SPK) file, and then use the %MM_Register macro to register the model to an existing project.
version in the SAS Model Manager model repository. Alternatively, you can import the model using the SAS Model Manager web application. These macros is available on SAS 9.4 where SAS Model Manager is installed.

For more information, see “%AA_Model_Register Autocall Macro” on page 52.

Use Macros to Register Models into SAS Model Manager

SAS Model Manager provides macros that you can use in a SAS program to add folders, projects, and versions, and to set properties, as well as register models. These macros are available on SAS 9.4 where SAS Model Manager is installed. For more information, see “Adding Folders, Projects, Versions, and Properties Using Macros” on page 57 and “%MM_Register Macro” on page 24.

Here is an example of using the macros to create a folder, project, and version, and then import the SAS model package file.

```sas
/******************************************
 Run the %AA_Model_Register Macro
 ******************************************/

/* Autocall the SAS 9.4 macro for generating the model SPK file. */
%aamodel;

/* Create a logselect SAS model package (SPK) file, */
/* with input variables and output variables metadata. */
/* Do not register the model in the SAS Metadata Repository. */
%aa_model_register(
    modelname=&model_name1,
    modeldesc=%bquote(logselect_badloans model from CAS),
    data=mycas.&training_dataset,
    target=BAD,
    level=binary,
    scorecodefile=~/&model_name1..sas,
    scorecodeformat=DATASTEP,
    register=N,
    spk=Y,
    spkfolder=%str(~),
    miningfunction=classification,
    debug=N);

/* Get the location of the model SPK file. */
%let spkdir1 = &AA_SPK_FOLDERID;

/*********************************************************
 Use Macros to Register Models into SAS Model Manager
 **********************************************************/

/* Initialize the Model Management macros and variables. */

/* Include the Model Management macros that are needed. */
filename aaa catalog "sashelp.modelmgr.accessmacros.source"; %include aaa;
filename aaa catalog "sashelp.modelmgr.mdlmgr_addfolder.source"; %include aaa;
filename aaa catalog "sashelp.modelmgr.mdlmgr_addproject.source"; %include aaa;
filename aaa catalog "sashelp.modelmgr.mdlmgr_setproperty.source"; %include aaa;
filename aaa catalog "sashelp.modelmgr.mdlmgr_addversion.source"; %include aaa;
```
filename aaa catalog 'sashelp.modelmgr.mdlmgr_ep_publish.source'; %include aaa;
filename aaa catalog 'sashelp.modelmgr.mdlmgr_hadoop_publish.source'; %include aaa; filename aaa;
%global topLvlFolder project1 version1;

/* Add a folder, project, and version to contain the models. */
%mdlmgr_AddFolder(ParentId=//ModelManagerDefaultRepo/MMRoot,
   Name=SASViya,
   desc=%bquote(Only models imported from SAS Viya allowed.),
   NewFolderId=topLvlFolder);
%mdlmgr_AddProject(parentId=&topLvlFolder.,
   name=Badloans, desc=none,
   modelFunction=classification,
   inputVarTable=qstut.hmeq_project_input,
   outputVarTable=qstut.hmeq_project_output,
   newProjectId=project1);
%mdlmgr_AddVersion(parentId=&project1.,
   newVersionId=version1);

/* Register the model into the model repository. */
filename mdlspk1 "&spkdir1./miningResult.spk";
%M_M_Register(
   VersionId=&version1.,
   EMModelPackage=mdlspk1,
   ModelName=&model_name1,
   Description=none
);

/* Cleanup files and data. */
filename score1 "~/&model_name1..sas";
data _NULL_;
   rc = fdelete("score1");
   rc = fdelete("mdlspk1");
   rc = fdelete("subdir1", "&spkdir1");
   rc = fdelete("subdir1");
run;

**View SAS Viya Models in SAS Model Manager**

You can now sign in to the SAS Model Manager web application to view the SAS Viya model that was registered. The retraining of DATA step score code models that have been created with SAS Visual Data Mining and Machine Learning procedures is not currently supported in SAS Model Manager. For more information about creating a project, importing models, and what tasks can be performed, see *SAS Model Manager: User’s Guide*. 
Figure 10.2  SAS Viya Folder with Badloans Project

Figure 10.3  List of Registered Models
Gain Greater Insight into Your SAS® Software with SAS Books.

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