THE POWER TO KNOW.

SAS® 9.3 Interface to Application Response Measurement (ARM): Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 Interface to Application Response Measurement (ARM): Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Interface to Application Response Measurement (ARM): Reference

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

1st electronic book, July 2011

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

Contents

Recommended Reading

- *SAS Logging: Configuration and Programming Reference*
- *SAS Intelligence Platform: System Administration Guide*

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

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

Part 1

Application Response Measurement (ARM)

Chapter 1

SAS ARM Interface Overview

What Is ARM?

Application Response Measurement (ARM) enables you to monitor the availability and performance of transactions within and across diverse applications. ARM enables enterprise management tools to extend directly to applications to measure application availability, performance, usage, and transaction response time.

The SAS ARM interface implements a number of features, which are compliant with the ARM 4.0 standards. SAS cooperates with open source ARM agents or vendor products that implement the ARM open API standard.

Why Is ARM Needed?

There are many techniques for measuring response times, but only ARM measures them accurately. Other techniques, although useful in other ways, might measure business service levels by assuming or guessing what a business transaction is, and when it begins and ends. Also, other techniques cannot provide the important information that ARM can, such as whether a transaction completed successfully.

Using ARM, you can log transaction records from an application to do the following:

- determine the application response times
- determine the workload and throughput of your applications
- verify that service-level objectives are being met
- determine why the application is not available
- verify who is using an application
- determine why a user is experiencing poor response time

- determine what queries are being issued by an application
- determine the subcomponents of an application's response time
- determine which servers are being used
- calculate the load time for data warehouses

Will ARM Affect an Application's Performance?

ARM is designed to be a high-speed interface that has minimal impact on applications. An ARM agent is designed to quickly extract the information that is needed and to return control to the application immediately. Processing of the information is done in a different process that can run when the application is otherwise idle.

What Are the SAS ARM Interface Features?

The SAS ARM interface uses the following features to measure and log application availability, performance, usage, and transaction response time:

- ARM agent, which is an executable program that contains an implementation of the ARM API
- ARM appender on page 59, which processes ARM transaction events and sends the events to a specified output destination
- ARM performance macros on page 95, which you strategically place in your SAS programs to define, start, and stop ARM data collection and contain default user metrics
- ARM system options:
  - ARMAGENT= on page 103, which specifies an executable module or keyword
  - ARMLOC= on page 104, which specifies the location of an ARM log
  - ARMSUBSYS= on page 105, which specifies whether to initialize the ARM subsystems
- default correlators on page 32, which are used to track parent and child transactions
- default user metrics on page 31, which are used to measure start, update, and stop times
- performance macros on page 95, which contain default user metrics
- SAS logging facility on page 7, which enables more flexibility and control of the ARM log destinations and message formats

Comparing the SAS 9.1 ARM Interface with the SAS 9.3 ARM Interface

The following table lists the differences between the SAS 9.1 ARM interface and the SAS 9.3 ARM interface.

Table 1.1 *SAS 9.1 ARM Interface Compared to SAS 9.3 ARM Interface*

Feature	SAS 9.1 ARM Interface	SAS 9.3 ARM Interface	SAS 9.3 ARM Interface Using the SAS Logging Facility
ARM macros	%ARMEND %ARMGTID %ARMINIT %ARMSTRT %ARMSTOP %ARMUPDT	%ARMEND %ARMGTID %ARMINIT %ARMSTRT %ARMSTOP %ARMUPDT %PERFEND %PERFINIT %PERFSTOP %PERFSTRT	%ARMEND %ARMGTID %ARMINIT %ARMSTRT %ARMSTOP %ARMUPDT %PERFEND %PERFINIT %PERFSTOP %PERFSTRT
ARMAGENT= *ARM agent* or SAS (default)	ARMAGENT= *ARM agent* or SAS (default)	ARMAGENT= *ARM agent* or SAS (default)	ARMAGENT=LOG4SAS
ARMLOC=	yes	yes	no
ARMSUBSYS=	yes	yes	yes
Correlators	yes, user-defined correlators	yes, user-defined correlators	by default
Post-processing macros	%ARMJOIN %ARMPROC	%ARMJOIN %ARMPROC	none
User metrics	limited user-defined metrics	user-defined metrics; additional user metrics are memory, thread count, and Read and Write statistics, which are default metrics when using %PERFSTRT and %PERFSTOP	all user metrics, by default, when using %PERFSTRT and %PERFSTOP macros

Chapter 2

ARM Logging

ARM Logging Overview

Logging is an integral part of ARM processing. The SAS 9.3 ARM interface offers two logging methods, the SAS logging facility and the ARM log. As the SAS ARM agent collects ARM information and statistics for the transactions, they are written to the SAS logging facility or the ARM log. If neither logging method is used, ARM information and statistics for the transactions are written to the SAS log.

Note: If you perform a planned installation, then the SAS Deployment Wizard provides logging configuration files for your SAS servers. You can dynamically adjust thresholds by using the server manager features of the SAS Management Console.

SAS Logging Facility

The SAS logging facility is a flexible and configurable framework that you can use to collect, categorize, and filter events, and write them to a variety of output devices. The logging facility supports problem diagnosis and resolution, performance and capacity management, and auditing and regulatory compliance.

To use the SAS logging facility, you can define a logging configuration file, which configures appenders and loggers. You can use the SAS logging facility in SAS programs. (For information, see “ARM Logging Using the SAS Language ” on page 12.) SAS provides sample SAS logging facility configuration files in the SAS Help and Documentation. To access the sample configuration files, do the following:

1. From the SAS main window, select **Help** ⇨ **SAS Help and Documentation**.

2. From SAS Help and Documentation, expand **Learning to Use SAS** ⇨ **Base SAS**.
3. Select **Samples**, and scroll to the SAS logging facility configuration file examples.

The SAS logging facility provides flexibility for processing transactions, and enables you to customize the formatting of messages that can be written to logs.

The ARM appender processes all ARM messages submitted by an external ARM agent or by SAS ARM processing. ARM messages are formatted based on various diagnostic contexts. To log ARM messages using a configuration file, you configure an ARMAppender, a FileAppender, and a logger. The ARMAppender definition specifies ARM appender parameters. The FileAppender definition contains the log file location and the message pattern layout. The logger specifies the PERF (performance) message category. You can also configure appenders and loggers in SAS programs.

For ARM appender information, see "ARM Appender Overview" on page 59. For logger information, see "ARM Logger Overview" on page 47.

In the programming environment, if the SAS logging facility is initialized for SAS server logging, messages are written to SAS logging facility locations. If the SAS logging facility is not initialized for SAS server logging, messages are written only to SAS logging facility locations that are created in a SAS program and written to the SAS log.

SAS Logging Facility Process

To use the SAS logging facility, do the following:

- Define a logging configuration file, which configures appenders and loggers. You can define the configuration in an XML file, or by using SAS language elements. If you perform a planned installation, then logging configuration files are provided for your SAS servers.
- Specify the LOGCONFIGLOC= system option to enable logging if you are using configuration files. The LOGCONFIGLOC= system option does not have to be specified for the logging facility to operate in SAS programs. If you perform a planned installation, then the LOGCONFIGLOC= system option is included in the configuration files for your SAS servers.
- Issue log events in a format that can be processed by the SAS logging facility if you are developing your own SAS programs.

Configuring ARM Logging in a Configuration File

Using the ARM interface, the SAS logging facility, and a configuration file, one or more logs can be created. Each log is a file appender that contains pattern layouts. The file appender is defined in the configuration file. The following syntax creates a file appender:

```
<appender class="FileAppender" name="LOG">
          <param name="File" value="logs/trace.log"/>
          <param name="Append" value="false"/>
          <layout>
               <param name="ConversionPattern"
```

```
                        value="%d %-5p [%t] %c (%F:%L) - %m"/>
          </layout>
</appender>
```

For more information about the configuration parameters in the file appender syntax, see "ARM Appender Configuration Parameters" on page 67.

A pattern layout is needed to create the output message format.

```
    <layout>
              <param name="ConversionPattern"
                    value="%d,
                     %X{App.Name},
                     %X{ARM.GroupName},
                     %X{ARM.TranName},
                     %X{ARM.TranState},
                     %X{ARM.ParentCorrelator},
                     %X{ARM.CurrentCorrelator},
                     %X{ARM.TranStatus},
                     %X{ARM.TranStart.Time},
                     %X{ARM.TranStop.Time},
                     %X{ARM.TranResp.Time}
               "/>
          </layout>
```

For more information about pattern layouts, see "ARM Appender Pattern Layouts for ARM Messages" on page 69.

The following configuration file creates three logs:

```
<?xml version="1.0"?>
<logging:configuration xmlns:log4sas="http://www.sas.com/xml/logging/1.0/"

<appender class="FileAppender" name="LOG">
          <param name="File" value="logs/trace.log"/>
          <param name="Append" value="false"/>
          <layout>
               <param name="ConversionPattern"
                         value="%d %-5p [%t] %c (%F:%L) - %m"/>
          </layout>
</appender>

<appender class="FileAppender" name="ARM2LOG">
          <param name="File" value="logs/arm2.log"/>
          <param name="Append" value="false"/>
          <layout>
               <param name="ConversionPattern "
                          value="%X{ARM2.Record}"/>
          </layout>
</appender>

<appender class="FileAppender" name="ARM4LOG">
          <param name="File" value="logs/arm4.log"/>
          <param name="Append" value="false"/>
          <layout>
              <param name="ConversionPattern"
                    value="%d,
                     %X{App.Name},
                     %X{ARM.GroupName},
```

```
                        %X{ARM.TranName},
                        %X{ARM.TranState},
                        %X{ARM.ParentCorrelator},
                        %X{ARM.CurrentCorrelator},
                        %X{ARM.TranStatus},
                        %X{ARM.TranStart.Time},
                        %X{ARM.TranStop.Time},
                        %X{ARM.TranResp.Time}
                    "/>
            </layout>
</appender>

<appender class="ARMAppender" name="ARM">
            <param name="GetTimes" value="TRUE"/>
            <appender-ref ref="ARM4LOG"/>
            <appender-ref ref="ARM2LOG"/>
</appender>

<logger name="Perf.ARM" additivity="true">
<level value="all"/>
<appender-ref ref="ARM"/>
</logger>

<root>
<level value="trace"/>
<appender-ref ref="LOG"/>
</root>

</logging:configuration>
```

Here are the three logs that are created by the configuration file (ARM2.LOG, ARM4.LOG, and TRACE.LOG):

Output 2.1 *ARM2.LOG*

```
I,1523810344.972000,1,0.062500,0.453125,SAS,
G,1523810344.972000,1,1,SAS,MVA SAS session
S,1523810344.972000,1,1,1,0.062500,0.453125
G,1523810344.972000,1,2,PROCEDURE,PROC START/STOP,PROC_NAME,
      ShortStr,PROC_IO,Count64,PROC_MEM,Count64,PROC_LABEL,LongStr
I,1523810345.566000,2,0.203125,0.734375,OpenCodeARMGTID test w/ user metrics,*
G,1523810345.691000,2,3,OpenCode02,,ShtStr,ShortStr,cnt32,Count32
S,1523810345.831000,2,3,2,0.359375,0.828125
S,1523810345.847000,1,2,3,0.359375,0.828125,DATASTEP,0,0,
P,1523810345.956000,1,2,3,0.375000,0.890625,0,DATASTEP,311110,283296,
S,1523810345.972000,1,2,4,0.375000,0.890625,SORT    ,0,0,
P,1523810346.128000,1,2,4,0.421875,0.953125,0,SORT    ,532806,2103048,
P,1523810346.191000,2,3,2,0.453125,0.984375,0
E,1523810346.238000,2,0.468750,1.000000
P,1523810346.285000,1,1,1,0.468750,1.015625,0
E,1523810346.300000,1,0.484375,1.015625
```

Output 2.2 *ARM4.LOG*

```
2008-04-14T12:39:04,972 | SAS | USERID |  | INIT |  |  |  |  |  |
2008-04-14T12:39:04,972 | SAS | USERID | SAS | REGISTER |  |  |  |  |  |
2008-04-14T12:39:04,972 | SAS | USERID | SAS | START |  |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |  |
        1523810344.972000 |  |
2008-04-14T12:39:04,972 | SAS | USERID | PROCEDURE | REGISTER |  |  |  |  |  |
2008-04-14T12:39:05,565 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID |  | INIT |  |  |  |  |  |
2008-04-14T12:39:05,690 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | REGISTER |  |  |  |  |  |
2008-04-14T12:39:05,830 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | START | ACj/
        ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |  |
        1523810345.831000 |  |
2008-04-14T12:39:05,847 | SAS | USERID | PROCEDURE | START |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AERGNDc3OEFELTlCNzItNEE3NC1CNjNFLTU5OTI2QTFERTQ4Ng== |  |
        1523810345.847000 |  |
2008-04-14T12:39:05,955 | SAS | USERID | PROCEDURE | STOP |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AERGNDc3OEFELTlCNzItNEE3NC1CNjNFLTU5OTI2QTFERTQ4Ng== |
        GOOD | 1523810345.847000 | 1523810345.956000 | 0.109000
2008-04-14T12:39:05,972 | SAS | USERID | PROCEDURE | START |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AEQ5QkY3MkE3LTVBMzQtNDg1Qi05MUQyLUI0MjZDMzE2MDk0NA== |  |
        1523810345.972000 |  |
2008-04-14T12:39:06,128 | SAS | USERID | PROCEDURE | STOP |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AEQ5QkY3MkE3LTVBMzQtNDg1Qi05MUQyLUI0MjZDMzE2MDk0NA== |
        GOOD | 1523810345.972000 | 1523810346.128000 | 0.156000
2008-04-14T12:39:06,190 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | STOP |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        GOOD | 1523810345.831000 | 1523810346.191000 | 0.360000
2008-04-14T12:39:06,237 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID |  | TERM |  |  |  |  |  |
2008-04-14T12:39:06,284 | SAS | USERID | SAS | STOP |  |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        GOOD | 1523810344.972000 | 1523810346.285000 | 1.313000
2008-04-14T12:39:06,299 | SAS | USERID |  | TERM |  |  |  |  |  |
```

Output 2.3 *TRACE.LOG*

```
2008-04-14T12:39:04,924 DEBUG [00000003] Logging (l4sasutil.c:830) -
        Loading the tk4afile support extension (1.0.0).
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender LOG
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender ARM2LOG
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender ARM4LOG
2008-04-14T12:39:04,955 DEBUG [00000003] Logging (l4sasutil.c:830) -
        Loading the tk4aarm4 support extension (1.0.0).
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:1192) - Creating ARM Appender ARM
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:1345) - Created ARM Appender ARM (0x1fa5f40)
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:947) - ARM Appender SetOption(GetTimes, TRUE)
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:275) - INIT SAS 13eec00 I,1523810344.972000,1,0.062500,
        0.453125,SAS,
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_2api.c:576) - REGISTER SAS 13eee58 G,1523810344.972000,1,1,
        SAS,MVA SAS session
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:1107) - START SAS 13eee58 0 S,1523810344.972000,1,1,1,
        0.062500,0.453125
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.PROC
        (tka_2api.c:576) - REGISTER PROCEDURE 13ef148 G,1523810344.972000,
        1,2,PROCEDURE,PROC START/STOP,PROC_NAME,ShortStr,
        PROC_IO,Count64,PROC_MEM,Count64,PROC_LABEL,LongStr
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) - 1
         The SAS System               12:39 Monday, April 14, 2008
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: SAS (r) Proprietary Software 9.3 (TS2B0)
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
         Licensed to SAS Institute Inc., Site 0000000001.
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: This session is executing on the XP_PRO  platform.
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
...
```

ARM Logging Using the SAS Language

The SAS language enables you to use the SAS logging facility in a DATA step and in macro programs. Using SAS logging facility language elements, you can create appenders, loggers, and log events in your SAS programs. Loggers that you create in your SAS program can reference appenders that are created in your SAS program, or appenders that are defined in a logging configuration file. When you create a log event in your SAS program, the logger that you specify in the log event can be one that was created in your SAS program, or one that is defined in the logging configuration file. For more information, see *SAS Logging: Configuration and Programming Reference*.

SAS processes the logging facility language elements whether the LOCCONFIGLOC= system option specifies a logging configuration file. If the LOGCONFIGLOC= system option does not specify a logging configuration file, all SAS logging facility messages are written to the SAS log. The messages also are written to the SAS logging facility location specified in the log event.

The following list contains other information needed to use the SAS logging facility and the SAS language:

- The logging facility is enabled for SAS programs at all times. The LOGCONFIGLOC= system option does not have to be specified for the logging facility to operate in SAS programs.
- Initializing the logging facility for SAS programs is necessary only if you use the logging facility autocall macros. SAS has no initialization process for the logging facility functions and DATA step objects.
- You create appenders in your SAS program before you create loggers or invoke a log event. The only appender class that you can create is FileRefAppender, which specifies to write messages to a file that is referenced by a fileref. For information about ARMAppender, see "ARM Appender Overview" on page 59.
- You create loggers in your SAS program by using either the %LOG4SAS_LOGGER autocall macro, the LOG4SAS_LOGGER function, or the logger object DECLARE statement. Loggers must be created after you create appenders and before you invoke a log event. For information about the Perf.ARM logger, see "ARM Logger Overview" on page 47.
- After loggers and appenders are created, you can add log events to your SAS program. You insert log events anywhere in your program or DATA step that you want to log a message. A log event takes three arguments, a logger name, a level, and the log message.

Traditional ARM Log

The SAS 9.3 ARM interface preserves the existing SAS 9.1 ARM interface as much as possible, including the ARM log. After you install SAS 9.3, you do not have to use the SAS logging facility or the performance macros. You can continue using user-defined correlators, ARM macros with user-defined metrics, and the ARMLOC= system option.

The ARM log is a preformatted output file, which contains comma-separated contents. The format includes a one-character identifier that represents the ARM call, a date.time group, transaction ID, user CPU time, and system CPU time. This information is followed by transaction-unique details, such as correlators and user metrics.

Instructions for defining correlators and user metrics are in the SAS 9.1 documentation.

The following code uses the ARM macros, user-defined metrics, and correlators:

```
/*****************************************************/
/* ARM sample and ARMLOG.LOG output with ARM macros    */
/* enabled outside the data step.                      */
/*                                                     */
/* The default name of the log file is ARMLOG.LOG. To  */
/* specify a unique name, use the ARMLOC system option */
/*****************************************************/
/*****************************************************/
```

```
/* Enable ARM Sub-System ARM_PROC                       */
/*******************************************************/
options ARMSUBSYS=(ARM_PROC);

/*******************************************************/
/* Enable ARM macro support                            */
/*******************************************************/
%let _armexec=1;

/*******************************************************/
/* Enable ARM macro support outside of the data step   */
/*******************************************************/
%let _armacro=1;

%arminit(appname="OpenCodeARMGTID test w/ user metrics");
%armgtid(txnname="OpenCode02",txnidvar=txn1,
     metrNam1="ShtStr",
     metrDef1=short,
     metrNam2="cnt32",
     metrDef2=count32);

%armstrt(txnidvar=txn1,metrval1=&syshostname);

 data x;
 do i=1 to 10000;
 x=i; y=0-i;
 output;
 end; run;

proc sort  data=x threads; by y;  run;

%armstop;
%armend;
```

The following is the ARM log output with **`_armacro=1`** and **`ARMSUBSYS=(ARM_PROC)`**:

```
I,1568904424.107000,1,0.796875,0.921875,SAS,sasgyc
G,1568904424.107000,1,1,PROCEDURE,PROC START/STOP,PROC_NAME,ShortStr,PROC_IO,
  Count64,PROC_MEM,Count64,PROC_LABEL,LongStr
I,1568904424.482000,2,0.796875,0.953125,OpenCodeARMGTID test w/ user metrics,*
G,1568904424.514000,2,2,OpenCode02,,ShtStr,ShortStr,cnt32,Count32
S,1568904424.529000,2,2,1,0.812500,0.984375,L13021  ,
S,1568904424.545000,1,1,2,0.812500,1.015625,DATASTEP,0,0,
P,1568904424.873000,1,1,2,0.812500,1.046875,0,DATASTEP,287803,260680,
S,1568904424.967000,1,1,3,0.812500,1.078125,SORT    ,0,0,
P,1568904424.982000,1,1,3,0.828125,1.093750,0,SORT    ,509499,1945640,
P,1568904424.998000,2,2,1,0.843750,1.093750,0
E,1568904424.998000,2,0.843750,1.093750
```

```
/*******************************************************/
/* ARM sample and ARMLOG.LOG output with ARM macros    */
/* enabled within a data step.                         */
/*******************************************************/
```

```
/*******************************************************/
/* Set ARM Sub-System ARM_NONE                         */
/*******************************************************/
options ARMSUBSYS=(ARM_NONE);

/*******************************************************/
/* Enable ARM macro support                            */
/*******************************************************/
%let _armexec=1;
/*******************************************************/
/* Set _armacro=0, therefore, ARM macros must be within*/
/* a data step boundary.                               */
/*******************************************************/
%let _armacro=0;
data x;
%arminit(appname="DataStepCode test w/ user metrics");
%armgtid(txnname="DataStepTxn1",txnidvar=txn1,
     metrNam1="ShtStr",
     metrDef1=short,
     metrNam2="cnt32",
     metrDef2=count32);
%armstrt(shdlvar=startid,txnidvar=txn1,metrval1="&syshostname");

do i=1 to 10000;
  x=i; y=0-i;
  output;
  end;

%armstop(shandle=startid);
%armend;
run;
```

The following is the ARM log output with **_armacro=0** and **ARMSUBSYS=(ARM_NONE)**:

```
I,1568907360.964000,1,0.703125,0.718750,SAS,sasgyc
I,1568907361.120000,2,0.750000,0.750000,DataStepCode test w/ user metrics,*
G,1568907361.120000,2,1,DataStepTxn1,,ShtStr,ShortStr,cnt32,Count32
S,1568907361.120000,2,1,1,0.750000,0.750000,L13021  ,
P,1568907361.136000,2,1,1,0.750000,0.765625,0
E,1568907361.136000,2,0.750000,0.765625
```

Note: If you use the SAS 9.3 ARM interface, the SAS logging facility, and the performance macros, do not use the postprocessing macros.

The following example uses the %ARMPROC and %ARMJOIN postprocessing macros:

```
filename ARMLOG 'd:\armlog';
  %armproc();
  %armjoin();
```

Key Behaviors That Change with the SAS Logging Facility

When you migrate an application containing ARM to the SAS logging facility, and the application was created in SAS 9.1 or earlier, the SAS 9.3 ARM interface enables the application, tools, and facilities to execute without change.

The following table lists the key behaviors or results for the SAS 9.1 ARM interface option or configuration value when the ARM appender is configured and enabled through the SAS logging facility.

Table 2.1 *SAS 9.1 ARM Application and SAS 9.3 ARM Integration*

Option or Configuration Value	Behavior or Results	SAS Logging Facility Processing
SAS ARM or PERF macro events, such as %ARMINIT or %PERFINIT	SAS macro processing is the same.	The SAS logging facility namespace is Perf.ARM.*APPL*. [1]
ARMAGENT=LOG4SAS	Enables the SAS logging facility ARM appender. ARM transactions are processed by the ARM appender.	SAS 9.1 ARM interface record formats are available through the SAS logging facility %X{ARM2.Record} pattern layout.
ARMSUBSYS=ARM_DSIO	Enables the ARM_DSIO subsystem.	The SAS logging facility namespace is Perf.ARM.*DSIO*.
ARMSUBSYS=ARM_PROC	Enables the ARM_PROC subsystem.	The SAS logging facility namespace is Perf.ARM.*PROC*.
ARMLOG/ARMLOC	ARM transaction records are emitted into the SAS logging facility architecture.	SAS 9.1 ARM interface record formats are available through the SAS logging facility %X{ARM2.Record} pattern layout. This option is ignored.
%ARMGTID and %ARMSTRT macros with user metrics	SAS macro processing is the same. Metrics are sent to the SAS logging facility ARM appender and made available as SAS logging facility pattern layout values.	SAS 9.1 ARM interface record formats are available via the SAS logging facility %X{ARM2.Record} pattern layout.

If the SAS logging facility is disabled, existing SAS applications that include ARM transaction support continue to execute without change.

[1] The name value is obtained from the SAS system option LOGAPPLNAME=, %ARMINIT macro, or the SAS logging facility configuration specifications.

Part 2

ARM Macro Environment

Chapter 3

Enabling ARM Macro Execution

Setting the _ARMEXEC Macro Variable

All performance macros and ARM macros are disabled by default so that inserting these macros within code does not result in inadvertent, unwanted logging. To globally enable performance macros and ARM macros, you must set the _ARMEXEC macro variable to a value of 1. Any other value for _ARMEXEC disables the macros.

There are two methods of setting the _ARMEXEC macro variable. The first method sets the variable during DATA step or SCL program compilation uses the %LET macro statement:

```
%let _armexec = 1;
```

The second method uses CALL SYMPUT statement during execution. To set the _ARMEXEC macro variable during DATA step or SCL program execution using CALL SYMPUT:

```
call symput('_armexec', '1');
```

With this method, the macro checks the _ARMEXEC variable during program execution and the ARM function call is executed or bypassed as appropriate.

If the _ARMEXEC value is not set to 1, then no code is generated and a message is written in the log:

```
NOTE: ARMSTRT macro bypassed by _armexec.
```

Enabling ARM Macro Execution with SCL

The two methods of setting the _ARMEXEC macro variable are during compilation or execution. Both methods are explained in “Setting the _ARMEXEC Macro Variable” on page 19. You can use a combination of these methods. For example, set _ARMEXEC to 1 using the compilation method (perhaps in an autoexec file at SAS initialization). Then

code a menu option or element within the application to turn _ARMEXEC on and off dynamically using the CALL SYMPUT statement.

In SCL, if _ARMEXEC is not set to 1, when the program compiles, all macros are set to null. The ARM interface is also unavailable until it is recompiled with _ARMEXEC set to 1.

In addition, to enable proper compilation of the macros within SCL, you must set the _ARMSCL global macro variable to 1 before issuing any ARM macros. The _ARMSCL macro variable suppresses the generation of DROP statements, which are invalid in SCL.

Conditional ARM Macro Execution for ARM

It is useful to code the ARM macros in your program, but to execute them only when they are needed. All ARM macros support a LEVEL= option that specifies the execution level of that particular macro.

If the LEVEL= option is coded, then the execution level of the macro is compared to two global macro variables, _ARMGLVL and _ARMTLVL. _ARMGLVL is the global level macro variable. If the LEVEL= value on the ARM macro is less than or equal to the _ARMGLVL value, then the macro is executed. If the LEVEL= value on the performance or ARM macro is greater than the _ARMGLVL value, then the macro is not executed:

```
        /*  Set the global level to 10 */
    %let _armglvl = 10;

  data _null_;
    %arminit(appname='Appl 1', appuser='userid' );
    %armgtid(txnname='Txn 1', txndet='Transaction #1 detail' );

      /*  These macros are executed */
    %armstrt( level=9 );
    %armstop( level=9 );

      /*  These macros are executed */
    %armstrt( level=10 );
    %armstop( level=10 );

      /*  These macros are NOT executed */
    %armstrt( level=11 );
    %armstop( level=11 );

    %armend
  run;
```

_ARMTLVL is the target level macro variable and works similarly to _ARMGLVL, except the LEVEL= value on the ARM macro must be equal to the _ARMTLVL value for the macro to execute:

```
        /*  Set the target level to 10 */
    %let _armtlvl = 10;

  data _null_;
    %arminit(appname='Appl 1', appuser='userid' );
```

```
  %armgtid(txnname='Txn 1', txndet='Transaction #1 detail' );

    /*  These macros are NOT executed */
  %armstrt( level=9 );
  %armstop( level=9 );

    /*  These macros are executed */
  %armstrt( level=10 );
  %armstop( level=10 );

    /*  These macros are NOT executed */
  %armstrt( level=11 );
  %armstop( level=11 );

  %armend
run;
```

The LEVEL= option can be used in any ARM macro, which is highly recommended. It enables you to design more granular levels of logging that can serve as a filter by logging only as much data as you want. If you set both _ARMGLVL and _ARMTLVL at the same time, then both values are compared to determine whether the macro should be executed.

Part 3

Using the ARM Interface

Chapter 4

Using the ARM Interface

ARM Interface Overview

The SAS 9.3 ARM interface preserves the existing SAS 9.1 ARM interface as much as possible. The ARM interface enables applications, tools, and facilities developed in ARM 2.0 to execute without change using ARM 4.0. The ARM interface offers several simplifications for ARM reporting transactions, and interface enhancements to customize ARM transaction reports. The performance (PERF) macros contain default user metrics. Default correlators are available by setting the MANAGECORRELATORS on page 67 parameter in the configuration file or within your SAS program. The ARM interface has been integrated with the SAS logging facility. You can customize how your ARM logs are formatted and where they are stored.

How the ARM Interface Works

The ARM API is an application programming interface that a vendor, such as SAS, can implement to monitor the availability and performance of transactions in distributed or client/server applications. The ARM API consists of definitions for a standard set of function calls that are callable from an application.

You determine the transactions within your application that you want to measure. To log specific SAS subsystem transactions, simply use the ARMSUBSYS= system option to turn on the transactions that you want to log.

You insert performance macros at strategic points in the application's code where you want transaction response times and other statistics collected. The performance macros generate calls to the ARM API function calls within the ARM agent. The SAS program accepts the function call parameters, checks for errors, and passes the ARM data to the ARM agent to calculate the statistics and to log the records.

Typically, an ARM API function call occurs just before a transaction is initiated to signify the beginning of the transaction. Then, an associated ARM API function call occurs in the application where the transaction is known to be completed. Basically, the application calls the ARM agent before a transaction starts, and then calls again after it ends, enabling the transaction to be measured and monitored. The transaction's response time and statistics are routed to the ARM agent, which logs the information.

In addition to the ARM API, SAS implemented several key elements that create the ARM environment. These key elements include:

ARMAGENT= system option
: is an executable module that contains an implementation of the ARM API. See "ARMAGENT= System Option" on page 103.

ARM macros
: are macros that are placed strategically in SAS programs to create and manage ARM transactions. You must create the user-defined metrics and correlators. See ARM Macros on page 79.

ARMSUBSYS= system option
: enables you to collect ARM data on internal SAS components, such as procedures and DATA steps. See "ARMSUBSYS= System Option" on page 105.

default user metrics
: are metrics that are collected in ARM transaction details using the performance macros. See "Default User Metrics and Performance Macros" on page 31.

default correlators
: are correlators that are collected in ARM transaction details using the performance macros. See "Default Correlators" on page 32.

performance macros
: are macros that are placed strategically in SAS programs to create and manage ARM transactions. Default user metrics and correlators are included in the performance macros. See "Using Performance Macros" on page 28.

SAS logging facility
: can produce one log or several logs that contain ARM transaction details. See "SAS Logging Facility" on page 7.

Using ARM System Options

SAS provides ARM system options, which are SAS system options that manage the ARM environment. The ARM system options enable you to log internal SAS processing transactions, such as file opening and closing, and DATA step and procedure response time.

The following is a list of ARM system options:

- "ARMAGENT= System Option" on page 103, which specifies another vendor's ARM agent that is an executable module that contains an implementation of the

ARM API. By default, SAS uses ARMAGENT=SAS. To use the SAS logging facility, set ARMAGENT=LOG4SAS.

- "ARMLOC= System Option" on page 104, specifies the location of the ARM log. ARMLOC= is not used with the SAS logging facility.
- "ARMSUBSYS= System Option" on page 105, enables you to use internal SAS components, such as procedures and DATA steps.

You can specify the ARM system options in the following ways:

- in a configuration file so that they are set automatically when you invoke SAS
- on the command line when you invoke SAS
- using the global OPTIONS statement in the SAS program or in an autoexec file
- from the System Options window

ARM API Function Calls

The ARM API function calls are contained in the SAS ARM agent. For the SAS implementation, you do not insert ARM API function calls in a SAS application, you insert performance macros, which generate calls to the ARM API function calls.

Here are the six ARM API function calls:

ARM_INIT
: names the application and the users of the application and initializes the ARM environment for the application.

ARM_GETID
: names a transaction.

ARM_START
: signals the start of a unique transaction.

ARM_UPDATE
: (optional) provides information about the progress of a transaction.

ARM_STOP
: signals the end of a unique transaction.

ARM_END
: terminates the ARM environment and signals the end of an application.

ARM API function calls use numeric identifiers (IDs) to uniquely identify the ARM objects that are input and output from the calls. There are three classes of IDs:

- application IDs
- transaction class IDs
- start handles (start times) for each instance of a transaction

IDs are numeric, assigned integers. The ARM agent assigns IDs. The scheme for assigning IDs varies from one vendor's agent to another, but, at a minimum, a unique ID within a single session is guaranteed. Some agents enable you to pre-assign IDs.

The following table shows the relationships between the ARM API function calls, the ARM macros, and the performance macros:

Table 4.1 Relationships between ARM API Function Calls, ARM Macros, and Performance Macros

ARM API Function Calls	ARM Macros	Performance Macros
ARM_INIT	%ARMINIT	%PERFINIT
ARM_GETID	%ARMGTID	
ARM_START	%ARMSTRT	%PERFSTRT
ARM_UPDATE	%ARMUPDT	
ARM_STOP	%ARMSTOP	%PERFSTOP
ARM_END	%ARMEND	%PERFEND

Using the SAS Logging Facility and the ARM Appender

The SAS logging facility provides greater flexibility and control for processing the ARM log, tracing, and providing better diagnostic messages. The SAS logging facility incorporates the ARM appender, which is configured and customized using the SAS logging facility. The primary role of the ARM appender is:

- capture ARM transaction events
- process the events
- route the events to an appropriate output destination
- emit ARM 4.0 compatible events

For more information about the ARM appender, see "ARM Appender Overview" on page 59. For more information about the SAS logging facility, see *SAS Logging: Configuration and Programming Reference*.

Using Performance Macros

When you use the performance macros, you do not need to define user metrics. There are default user metrics within the %PERFSTRT and %PERFSTOP macros. The default user metrics simplify performance tracking. To compare the features of performance macros and ARM macros, see "Comparing the SAS 9.1 ARM Interface with the SAS 9.3 ARM Interface" on page 5. There are four performance macros:

%PERFINIT
: names the application instance and initializes the ARM interface. Typically, you insert this macro in your code once. See "%PERFINIT Macro" on page 96.

%PERFSTRT
: specifies the start of an instance of a transaction. Insert the %PERFSTRT macro before each transaction that you want to log. See "%PERFSTRT Macro" on page 97.

%PERFSTOP
: ends an instance of a transaction. Insert the %PERFSTOP macro where the transaction is known to be completed. See"%PERFSTOP Macro" on page 96.

%PERFEND
: signals the termination of the application. See "%PERFEND Macro" on page 95.

The %PERFSTRT and %PERFSTOP macros contain default user metrics, which alleviates defining and adding user-metric definitions, types, and values to the macros.

The following program uses the performance macros:

```
%log4sas();
%log4sas_logger(Perf.ARM, 'level=info');
options armagent=log4sas;
options armsubsys=(arm_proc);
%let _armexec = 1;
%perfinit(appname="Perf_App");

%perfstrt(txnname="Perf_Tran_1");
  data x;
  do i=1 to 10000;
  x=i; y=0-i;
  output;
  end; run;

  proc sort  data=x threads; by y;  run;
%perfstop;

%perfstrt(txnname="Perf_Tran_2");
  data x;
  do i=1 to 10000;
  x=i; y=0-i;
  output;
  end;
  run;

 proc sort  data=x threads; by y;  run;
%perfstop;

%perfend;
run;
```

Here is the output to the SAS log:

```
39   %log4sas();
40   %log4sas_logger(Perf.ARM, 'level=info');
41   options armagent=log4sas;
42   options armsubsys=(arm_proc);
43   %let _armexec = 1;
44   %perfinit(appname="Perf_App");
NOTE: INIT Perf_App 13f01e0
I,1533752349.599000,3,2.093750,8.437500,Perf_App,userid
45
46   %perfstrt(txnname="Perf_Tran_1");
NOTE: REGISTER Perf_Tran_1
13f0438
      G,1533752349.599000,3,5,Perf_Tran_1,,_IOCOUNT_,Count64,_MEMCURR_,
Gauge64,_MEMHIGH_,Gauge64,_THREADCURR_,Gauge32,_THREADHIGH_,      Gauge32NOTE: START
Perf_Tran_1 13f0438 0

S,1533752349.599000,3,5,8,2.093750,8.437500,266202463,13754368,15114240,3,7
47     data x;
NOTE: START PROCEDURE 13ef148 0 S,1533752349.614000,1,2,9,2.125000,8.437500,
               DATASTEP,0,0,
48     do i=1 to 10000;
49     x=i; y=0-i;
50     output;
51     end; run;

NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: STOP PROCEDURE 13ef148 0
      P,1533752349.630000,1,2,9,2.125000,8.453125,0,DATASTEP,278854,164944,
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


52
53     proc sort  data=x threads; by y;  run;
NOTE: START PROCEDURE 13ef148 0 S,1533752349.630000,1,2,10,2.125000,8.453125,
                SORT    ,0,0,

NOTE: There were 10000 observations read from the data set WORK.X.
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: STOP PROCEDURE 13ef148 0
P,1533752358.681000,1,2,10,2.171875,8.453125,0,SORT
      ,524967,1989360,
NOTE: PROCEDURE SORT used (Total process time):
      real time           9.05 seconds
      cpu time            0.04 seconds


54   %perfstop;
NOTE: STOP Perf_Tran_1 13f0438 0

P,1533752358.697000,3,5,8,2.171875,8.468750,0,267014988,13754368,15114240,3,7
5556
%perfstrt(txnname="Perf_Tran_2");
NOTE: REGISTER Perf_Tran_2
13f0b68
      G,1533752358.697000,3,6,Perf_Tran_2,,_IOCOUNT_,Count64,_MEMCURR_,

Gauge64,_MEMHIGH_,Gauge64,_THREADCURR_,Gauge32,_THREADHIGH_,Gauge32
NOTE: START Perf_Tran_2 13f0b68 0

S,1533752358.697000,3,6,11,2.171875,8.468750,267019084,13754368,15114240,3,7
57     data x;
NOTE: START PROCEDURE 13ef148 0 S,1533752358.712000,1,2,12,2.187500,8.468750,
               DATASTEP,0,0,
58     do i=1 to 10000;
59     x=i; y=0-i;
60     output;
61     end;
62     run;
```

```
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: STOP PROCEDURE 13ef148 0
      P,1533752358.728000,1,2,12,2.187500,8.484375,0,DATASTEP,278854,164944,
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


63
64    proc sort  data=x threads; by y;  run;
NOTE: START PROCEDURE 13ef148 0 S,1533752358.728000,1,2,13,2.187500,8.484375,
                SORT    ,0,0,

NOTE: There were 10000 observations read from the data set WORK.X.
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: STOP PROCEDURE 13ef148 0
P,1533752358.822000,1,2,13,2.234375,8.484375,0,SORT
      ,524614,1989840,
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds


65   %perfstop;
NOTE: STOP Perf_Tran_2 13f0b68 0

P,1533752358.837000,3,6,11,2.250000,8.484375,0,267835352,13754368,15114240,3,7
66
67   %perfend;
NOTE: END Perf_App 13f01e0
E,1533752358.837000,3,2.250000,8.484375
68   run;
```

Default User Metrics and Performance Macros

All of the ARM user metrics are defaults within the %PERFSTRT and %PERFSTOP macros. You do not have to specify user-metric definitions, types, and values within your code.

The following table shows the relationships between the %PERFSTRT macro, %PERFSTOP macro, and the default user metrics.

Table 4.2 *Default User Metrics within Performance Macros*

Metric Name*	Metric Type	Description
IOCOUNT	COUNT64	The metric value is the total number of disk, tape, or related input and output operations at each %PERFSTRT and %PERFSTOP event. The metric value is obtained from the host operating system and is associated with the input and output operations for that process. The value is a running count at the time of the event.

Metric Name*	Metric Type	Description
MEMCURR	GAUGE64	The metric value is the current value for memory used in the process at each %PERFSTRT and %PERFSTOP event. The metric value is obtained from the host operating system.
MEMHIGH	GAUGE64	The metric value is the highest amount of memory used for the life cycle of the current process at each ARM event. The metric value is obtained from the host operating system.
_THREADCURR	GAUGE32	The metric value is the current thread count of the process at each ARM event. The metric value is obtained from internal SAS counters.
_THREADHIGH	GAUGE32	The metric value is the highest number of active threads for the life cycle of the current process at each ARM event. The metric value is obtained from internal SAS counters.

* The predefined metric names and metric types on the %PERFSTRT macro can be specified in the %ARMGTID macro. The SAS 9.3 ARM interface processes the predefined metrics. The results are sent to the SAS logging facility ARM appender or to the ARM log.

Default Correlators

Correlators are used to track parent and child transactions. A primary or parent transaction can contain several component or child transactions nested within it. Child transactions can contain other child transactions. It can be useful to know how much each child transaction contributes to the total response time of the parent transaction. If a parent transaction fails, knowing which child transaction contains the failure is useful.

Correlators can be recorded when you use the ARM appender. You obtain correlated transaction by default when you use the MANAGECORRELATORS parameter. When specified, transaction correlation is enabled for all ARM transactions processed by the ARM appender. The transaction correlators are in the SAS logging facility configuration file or in your SAS program. See “ARM Appender Configuration Parameters” on page 67 for the appropriate format and specification for the configuration file.

The following table shows the transaction correlation behavior in certain events when the MANAGECORRELATORS parameter is enabled (VALUE=TRUE) by the SAS logging facility:

Table 4.3 *SAS 9.3 Transaction Correlator Behavior*

Event	SAS Logging Facility Transaction Correlator	Behavior
SAS initialization	ARM.CurrentCorrelator	A default SAS transaction is started and ARM.CurrentCorrelator is created by the default SAS transaction. The default SAS ARM.CurrentCorrelator becomes the ARM.ParentCorrelator. Each %ARMSTRT or %PERFSTRT macro creates a new ARM.CurrentCorrelator.
SAS 9.1 and 9.3 %ARMSTRT macro; SAS 9.3 %PERFSTRT macro	ARM.ParentCorrelator	The previous ARM.CurrentCorrelator becomes the ARM.ParentCorrelator.
	ARM.CurrentCorrelator	The value is created by the correlator associated with the %ARMSTRT or %PERFSTRT transaction event.
SAS 9.3 ARMSUBSYS=START transaction event	ARM.ParentCorrelator	The previous ARM.CurrentCorrelator becomes the ARM.ParentCorrelator.
	ARM.CurrentCorrelator	The value is created by the correlator associated with the %ARMSTRT or %PERFSTRT transaction event.
SAS 9.1 and SAS 9.3 %ARMSTOP macro; SAS 9.3 %PERFSTOP macro; or ARMSUBSYS=STOP transaction event	ARM.ParentCorrelator ARM.CurrentCorrelator	All information associated with the current transaction is cleared from the ARM infrastructure and the parent transaction is maintained.

If the MANAGECORRELATORS parameter has VALUE=FALSE, you must define and manage the parent correlators.

Chapter 5

Using SAS 9.3 ARM Interface with Existing ARM Applications

SAS 9.3 ARM Interface with Existing SAS Applications Overview

The SAS 9.3 ARM interface preserves the existing SAS 9.1 ARM interface as much as possible. The ARM interface enables applications, tools, and facilities developed in SAS 9.1 ARM interface and ARM 2.0 to execute without change using ARM 4.0. But, adding the new performance (PERF) macros reduces performance overhead. The following sections document how to add the SAS 9.3 ARM interface to existing SAS applications:

- “Adding ARM to an Existing SAS Application” on page 36
- “Adding ARM to an Existing SAS Application that Contains Basic ARM Instrumentation” on page 36
- “Adding ARM to an Existing SAS Application that Contains Extensive Use of ARM Instrumentation” on page 37

Requirement for ARM Appender

When using ARM, a certain order must be followed when setting up the ARM appender:

1. the ARM appender must be enabled at the INFO diagnostic level before specifying the ARMAGENT= option
2. the ARMAGENT= option must be specified before the ARMSUBSYS= option

These options can be specified using the SAS command line or OPTIONS statement. Here is an example using a SAS command line:

```
"<SAS92_installation_path\sas.exe>"
 -CONFIG "<SAS92_installation_path\nls\en\SASV9.CFG>"
-logconfigloc logconfig.xml  -armagent log4sas  -armsubsys=(arm_proc)
```

Here is an example using the OPTIONS statement:

```
%log4sas();
%log4sas_logger(Perf.ARM, 'level=info');
options armagent=log4sas;
options armsubsys=(arm_proc);
```

Adding ARM to an Existing SAS Application

You have a SAS application, and you want to track the application's performance. Using the SAS logging facility and the ARM interface, you can get the performance information that you want with little development time. There are two options available:

- enable the ARMSUBSYS= option (for example, in ARM_PROC), which gathers metric data at the PROC or DATA step boundaries
- enable the performance macros at key points in the SAS code to obtain metric data

Adding performance macros results in reduced performance overhead. Use the following steps to add ARM to an application using a configuration file:

1. Create a SAS logging facility -logconfigloc configuration file. For an example of a SAS logging facility configuration file, see "SAS Logging Facility Configuration File" on page 115. For more information about the SAS logging facility, see *SAS Logging: Configuration and Programming Reference*.
2. Specify **`ARMAGENT=LOG4SAS`** in a configuration file or within SAS language code.
3. Enable the ARMSUBSYS= system option (ARMSUBSYS= can be enabled in a SAS option).
4. Add %PERFINIT and %PERFEND macros to the beginning and end of your SAS language functions.
5. Add %PERFSTRT and %PERFSTOP macros around key events, such as procedures or DATA step functions.
6. Specify the SAS macro variable **`_armexec=1`** in a SAS configuration file to enable or disable ARM.

The SAS logging facility contains transaction metrics for key events. The output destination, output format, and quantity of the information can be defined in a file appender.

Adding ARM to an Existing SAS Application that Contains Basic ARM Instrumentation

The SAS 9.3 ARM interface does the following:

- enables applications to conform to ARM 2.0 without modification
- supports enhanced metric capabilities
- provides greater flexibility and control of the output data formats by using the SAS logging facility.

Basic ARM 2.0 instrumentation does not contain user-defined metrics or correlation.

The following steps are the recommended changes to an application that contains basic ARM instrumentation:

1. Create a SAS logging facility -logconfigloc configuration file. For an example, see "SAS Logging Facility Configuration File" on page 115. For more information about the SAS logging facility, see *SAS Logging: Configuration and Programming Reference*.
2. Specify **`ARMAGENT=LOG4SAS`** in a configuration file or within SAS language code.
3. Replace the %ARMINIT and %ARMEND macros with the %PERFINIT and %PERFEND macros.
4. Replace the %ARMSTRT and %ARMSTOP macros with the %PERFSTRT and %PERFSTOP macros.

   Note: The simplified syntax of the %PERFSTRT and %PERFSTOP macros contains the additional default user metrics, which include memory, thread count, and disk Read and Write statistics.

The SAS logging facility contains transaction metrics for key events. The output destination, output format, and quantity of the information can be defined in a file appender.

Adding ARM to an Existing SAS Application that Contains Extensive Use of ARM Instrumentation

An application created before SAS 9.3 that extensively uses ARM user-defined metrics and correlators continues to execute without modification.

Although you can execute the application without modification, you can enhance performance by making the following changes to the application:

1. Create a SAS logging facility -logconfigloc configuration file. For an example, see "SAS Logging Facility Configuration File" on page 115. For more information about the SAS logging facility, see *SAS Logging: Configuration and Programming Reference*.
2. Specify **`<name="ManageCorrelators">`** and **`<value="FALSE">`** (if correlators are defined) in the SAS logging facility configuration file.
3. Specify **`ARMAGENT=LOG4SAS`** in a configuration file or within SAS language code.
4. Review the additional user metrics in Table 4.2 on page 31, and add these metric definitions and values to your %ARMGTID, %ARMSTRT, and %ARMSTOP macros.

The SAS logging facility, using the pattern layouts and conversion specifiers, enables you to customize output format and layout. Using default correlation and the additional user metrics enables you to isolate performance issues within the application.

Chapter 6

ARM Interface and SAS Logging Facility

Creating Logs Using a Configuration File

Using the ARM interface, the SAS logging facility, and a configuration file, the following two logs are created:

```
<?xml version="1.0"?>
<logging:configuration xmlns:log4sas="http://www.sas.com/xml/logging/1.0/"

<appender class="FileAppender" name="LOG">
          <param name="File" value="logs/trace.log"/>
          <param name="Append" value="false"/>
          <layout>
               <param name="ConversionPattern"
                         value="%d %-5p [%t] %c (%F:%L) - %m"/>
          </layout>
</appender>

<appender class="FileAppender" name="ARM4LOG">
          <param name="File" value="logs/arm4.log"/>
          <param name="Append" value="false"/>
          <layout>
              <param name="ConversionPattern"
                    value="%d,
                     %X{App.Name},
                     %X{ARM.GroupName},
                     %X{ARM.TranName},
                     %X{ARM.TranState},
                     %X{ARM.ParentCorrelator},
                     %X{ARM.CurrentCorrelator},
                     %X{ARM.TranStatus},
                     %X{ARM.TranStart.Time},
                     %X{ARM.TranStop.Time},
                     %X{ARM.TranResp.Time}
                    "/>
          </layout>
</appender>
```

```
<appender class="ARMAppender" name="ARM">
          <param name="GetTimes" value="TRUE"/>
          <appender-ref ref="ARM4LOG"/>
 </appender>

<logger name="Perf.ARM" additivity="true">
<level value="all"/>
<appender-ref ref="ARM"/>
</logger>

<root>
<level value="trace"/>
<appender-ref ref="LOG"/>
</root>

</logging:configuration>
```

The ARM4.LOG file and TRACE.LOG file are created by the configuration file:

Output 6.1 *ARM4.LOG*

```
2008-04-14T12:39:04,972 | SAS | USERID |  | INIT |  |  |  |  |  |
2008-04-14T12:39:04,972 | SAS | USERID | SAS | REGISTER |  |  |  |  |  |
2008-04-14T12:39:04,972 | SAS | USERID | SAS | START |  |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |  |
        1523810344.972000 |  |
2008-04-14T12:39:04,972 | SAS | USERID | PROCEDURE | REGISTER |  |  |  |  |  |
2008-04-14T12:39:05,565 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID |  | INIT |  |  |  |  |  |
2008-04-14T12:39:05,690 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | REGISTER |  |  |  |  |  |
2008-04-14T12:39:05,830 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | START | ACj/
        ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |  |
        1523810345.831000 |  |
2008-04-14T12:39:05,847 | SAS | USERID | PROCEDURE | START |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AERGNDc3OEFELTlCNzItNEE3NC1CNjNFLTU5OTI2QTFERTQ4Ng== |  |
        1523810345.847000 |  |
2008-04-14T12:39:05,955 | SAS | USERID | PROCEDURE | STOP |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AERGNDc3OEFELTlCNzItNEE3NC1CNjNFLTU5OTI2QTFERTQ4Ng== |
        GOOD | 1523810345.847000 | 1523810345.956000 | 0.109000
2008-04-14T12:39:05,972 | SAS | USERID | PROCEDURE | START |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AEQ5QkY3MkE3LTVBMzQtNDg1Qi05MUQyLUI0MjZDMzE2MDk0NA== |  |
        1523810345.972000 |  |
2008-04-14T12:39:06,128 | SAS | USERID | PROCEDURE | STOP |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        ACj/AEQ5QkY3MkE3LTVBMzQtNDg1Qi05MUQyLUI0MjZDMzE2MDk0NA== |
        GOOD | 1523810345.972000 | 1523810346.128000 | 0.156000
2008-04-14T12:39:06,190 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID | OpenCode02 | STOP |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        ACj/AEY3MjJDNDJCLTAxNkEtNDMxNy1BNzc4LTc2OTA5Mjg1QzRGNQ== |
        GOOD | 1523810345.831000 | 1523810346.191000 | 0.360000
2008-04-14T12:39:06,237 | OpenCodeARMGTID_test_w/_user_metrics |
        USERID |  | TERM |  |  |  |  |  |
2008-04-14T12:39:06,284 | SAS | USERID | SAS | STOP |  |
        ACj/ADExOUUzQkI3LUJGQTgtNDlBOS04RTg1LUVDOTJCRTQ2RTY4OA== |
        GOOD | 1523810344.972000 | 1523810346.285000 | 1.313000
2008-04-14T12:39:06,299 | SAS | USERID |  | TERM |  |  |  |  |  |
```

Output 6.2 *TRACE.LOG*

```
2008-04-14T12:39:04,924 DEBUG [00000003] Logging (l4sasutil.c:830) -
        Loading the tk4afile support extension (1.0.0).
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender LOG
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender ARM2LOG
2008-04-14T12:39:04,924 DEBUG [00000003] Logging.Appender.File
        (tk4afile.c:1082) - Creating FileAppender ARM4LOG
2008-04-14T12:39:04,955 DEBUG [00000003] Logging (l4sasutil.c:830) -
        Loading the tk4aarm4 support extension (1.0.0).
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:1192) - Creating ARM Appender ARM
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:1345) - Created ARM Appender ARM (0x1fa5f40)
2008-04-14T12:39:04,955 DEBUG [00000003] Logging.Appender.ARM
        (tk4aarm4.c:947) - ARM Appender SetOption(GetTimes, TRUE)
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:275) - INIT SAS 13eec00 I,1523810344.972000,1,0.062500,
        0.453125,SAS,
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_2api.c:576) - REGISTER SAS 13eee58 G,1523810344.972000,1,1,
        SAS,MVA SAS session
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:1107) - START SAS 13eee58 0 S,1523810344.972000,1,1,1,
        0.062500,0.453125
2008-04-14T12:39:04,972 INFO  [00000003] Perf.ARM.SAS.PROC
        (tka_2api.c:576) - REGISTER PROCEDURE 13ef148 G,1523810344.972000,
        1,2,PROCEDURE,PROC START/STOP,PROC_NAME,ShortStr,
        PROC_IO,Count64,PROC_MEM,Count64,PROC_LABEL,LongStr
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) - 1
         The SAS System               12:39 Monday, April 14, 2008
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: SAS (r) Proprietary Software 9.3 (TS2B0)
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
         Licensed to SAS Institute Inc., Site 0000000001.
2008-04-14T12:39:05,299 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: This session is executing on the XP_PRO  platform.
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,315 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,315 INFO  [00000003] Admin.Session (yaxbtch.c:555) -
        SAH231999I Batch, State, started, Startup command-line is:
         "C:\SASv9\sasgen\dev\mva-v920\sas_dvd\com\dntnd\sas.exe" test1.sas
        -config "C:\SASv9\tmp\SASv9-832.cfg" -
        sashost "C:\SASv9\sasgen\dev\mva-v920\sas_dvd\com\dntnd\sashost.dll"
        -logconfigloc ./log4sasarm.xml -armsubsys "(arm_proc)" -armagent log4sas
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: SAS initialization used:
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
        real time           0.79 seconds
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
        cpu time            0.78 seconds
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,393 INFO  [00000008] App.Program (ynl4sas.c:123) - 1
        %let _armexec = 1;
2008-04-14T12:39:05,393 INFO  [00000008] App.Program (ynl4sas.c:123) - 2

        %let _armacro = 1;
2008-04-14T12:39:05,455 INFO  [00000008] App.Program (ynl4sas.c:123) - 3

2008-04-14T12:39:05,455 INFO  [00000008] App.Program (ynl4sas.c:123) - 4

        %arminit(appname="OpenCodeARMGTID test w/ user metrics");
```

```
2008-04-14T12:39:05,565 INFO  [00000003]
        Perf.ARM.OpenCodeARMGTID_test_w/_user_metrics.APPL (tka_api.c:275) -
        INIT OpenCodeARMGTID_test_w/_user_metrics 13f01e0 I,1523810345.566000,2,
        0.203125,0.734375,OpenCodeARMGTID test w/ user metrics,*
2008-04-14T12:39:05,565 INFO  [00000008] App.Program (ynl4sas.c:123) - 5

        %armgtid(txnname="OpenCode02",txnidvar=txn1,
2008-04-14T12:39:05,597 INFO  [00000008] App.Program (ynl4sas.c:123) - 6

         metrNam1="ShtStr",
2008-04-14T12:39:05,612 INFO  [00000008] App.Program (ynl4sas.c:123) - 7

        metrDef1=short,
2008-04-14T12:39:05,612 INFO  [00000008] App.Program (ynl4sas.c:123) - 8

          metrNam2="cnt32",
2008-04-14T12:39:05,612 INFO  [00000008] App.Program (ynl4sas.c:123) - 9

         metrDef2=count32);
2008-04-14T12:39:05,690 INFO  [00000003]
        Perf.ARM.OpenCodeARMGTID_test_w/_user_metrics.APPL (tka_api.c:632) -
        REGISTER OpenCode02 13f0438 G,1523810345.691000,2,3,OpenCode02,,
        ShtStr,ShortStr,cnt32,Count32
2008-04-14T12:39:05,690 INFO  [00000008] App.Program (ynl4sas.c:123) - 10

2008-04-14T12:39:05,705 INFO  [00000008] App.Program (ynl4sas.c:123) - 11

         %put SYSPARM=&sysparm
2008-04-14T12:39:05,705 INFO  [00000008] App.Program (ynl4sas.c:123) -
         SYSPARM=
2008-04-14T12:39:05,705 INFO  [00000008] App.Program (ynl4sas.c:123) - 12

        %armstrt(txnidvar=txn1,metrval1=&sysparm);
2008-04-14T12:39:05,830 INFO  [00000003]
        Perf.ARM.OpenCodeARMGTID_test_w/_user_metrics.APPL (tka_api.c:1120) -
        START OpenCode02 13f0438 0 S,1523810345.831000,2,3,2,0.359375,0.828125
2008-04-14T12:39:05,830 INFO  [00000008] App.Program (ynl4sas.c:123) - 13

2008-04-14T12:39:05,847 INFO  [00000003] Perf.ARM.SAS.PROC (tka_api.c:1107) -
        START PROCEDURE 13ef148 0 S,1523810345.847000,1,2,3,0.359375,0.828125,
        DATASTEP,0,0,
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) - 14

        data x;
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) - 15

        do i=1 to 10000;
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) - 16

        x=i; y=0-i;
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) - 17

        output;
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) - 18

         end; run;
2008-04-14T12:39:05,878 INFO  [00000009] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,940 INFO  [00000009] App.Program (ynl4sas.c:123) -
        NOTE: The data set WORK.X has 10000 observations and 3 variables.
2008-04-14T12:39:05,955 DEBUG [00000003] App.Program (yaktksri.c:418) -
        Remove appenders in a reb.
2008-04-14T12:39:05,315 INFO  [00000003] Admin.Session (yaxbtch.c:555) -
        SAH231999I Batch, State, started, Startup command-line is:
         "C:\SASv9\sasgen\dev\mva-v920\sas_dvd\com\dntnd\sas.exe" test1.sas
        -config "C:\SASv9\tmp\SASv9-832.cfg" -
        sashost "C:\SASv9\sasgen\dev\mva-v920\sas_dvd\com\dntnd\sashost.dll"
        -logconfigloc ./log4sasarm.xml -armsubsys "(arm_proc)" -armagent log4sas
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: SAS initialization used:
2008-04-14T12:39:05,362 INFO  [00000008] App.Program (ynl4sas.c:123) -
        real time           0.79 seconds
```

```
2008-04-14T12:39:05,955 INFO  [00000003] Perf.ARM.SAS.PROC
        (tka_api.c:1750) - STOP PROCEDURE 13ef148 0 P,1523810345.956000,1,2,
        3,0.375000,0.890625,0,DATASTEP,311110,283296,
2008-04-14T12:39:05,955 INFO  [00000009] App.Program (ynl4sas.c:123) -
        NOTE: DATA statement used (Total process time):
2008-04-14T12:39:05,955 INFO  [00000009] App.Program (ynl4sas.c:123) -
         real time           0.10 seconds
2008-04-14T12:39:05,955 INFO  [00000009] App.Program (ynl4sas.c:123) -
        cpu time            0.07 seconds
2008-04-14T12:39:05,955 INFO  [00000009] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,955 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:05,955 INFO  [00000008] App.Program (ynl4sas.c:123) - 19

2008-04-14T12:39:05,972 INFO  [00000003] Perf.ARM.SAS.PROC
        (tka_api.c:1107) - START PROCEDURE 13ef148 0 S,1523810345.972000,1,
        2,4,0.375000,0.890625,SORT    ,0,0,
2008-04-14T12:39:05,972 INFO  [00000010] App.Program (ynl4sas.c:123) -
        20         proc sort  data=x threads; by y;  run;
2008-04-14T12:39:05,972 INFO  [00000010] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,065 INFO  [00000010] App.Program (ynl4sas.c:123) -
        NOTE: There were 10000 observations read from the data set WORK.X.
2008-04-14T12:39:06,128 INFO  [00000010] App.Program (ynl4sas.c:123) -
        NOTE: The data set WORK.X has 10000 observations and 3 variables.
2008-04-14T12:39:06,128 DEBUG [00000003] App.Program (yaktksri.c:418) -
        Remove appenders in a reb.
2008-04-14T12:39:06,128 INFO  [00000003] Perf.ARM.SAS.PROC
        (tka_api.c:1750) - STOP PROCEDURE 13ef148 0 P,1523810346.128000,
        1,2,4,0.421875,0.953125,0,SORT    ,532806,2103048,
2008-04-14T12:39:06,128 INFO  [00000010] App.Program (ynl4sas.c:123) -
        NOTE: PROCEDURE SORT used (Total process time):
2008-04-14T12:39:06,128 INFO  [00000010] App.Program (ynl4sas.c:123) -
         real time           0.15 seconds
2008-04-14T12:39:06,128 INFO  [00000010] App.Program (ynl4sas.c:123) -
        cpu time            0.10 seconds
2008-04-14T12:39:06,128 INFO  [00000010] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,128 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,143 INFO  [00000008] App.Program (ynl4sas.c:123) - 21

2008-04-14T12:39:06,143 INFO  [00000008] App.Program (ynl4sas.c:123) - 22

         %armstop;
2008-04-14T12:39:06,190 INFO  [00000003]
        Perf.ARM.OpenCodeARMGTID_test_w/_user_metrics.APPL (tka_api.c:1761) -
        STOP OpenCode02 13f0438 0 P,1523810346.191000,2,3,2,0.453125,0.984375,0
2008-04-14T12:39:06,205 INFO  [00000008] App.Program (ynl4sas.c:123) - 23

        %armend;
2008-04-14T12:39:06,237 INFO  [00000003]
        Perf.ARM.OpenCodeARMGTID_test_w/_user_metrics.APPL (tka_api.c:1927) -
        END OpenCodeARMGTID_test_w/_user_metrics 13f01e0 E,1523810346.238000,
        2,0.468750,1.000000
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,237 INFO  [00000003] Admin.Session (yaxbtch.c:886) -
        SAH239999I Batch, State, stopped
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
        NOTE: The SAS System used:
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) - 2
         The SAS System                              12:39 Monday, April 14,
2008
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
        real time           1.67 seconds
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
         cpu time            1.46 seconds
```

```
2008-04-14T12:39:06,237 INFO  [00000008] App.Program (ynl4sas.c:123) -
2008-04-14T12:39:06,253 DEBUG [00000003] App.Program (yaktksri.c:418) -
        Remove appenders in a reb.
2008-04-14T12:39:06,268 DEBUG [00000003] Logging.Appender.FileRef
        (tk4afref.c:1072) - FileRef appender factory is being destroyed
2008-04-14T12:39:06,284 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:1750) - STOP SAS 13eee58 0 P,1523810346.285000,1,1,1,
        0.468750,1.015625,0
2008-04-14T12:39:06,299 INFO  [00000003] Perf.ARM.SAS.APPL
        (tka_api.c:1927) - END SAS 13eec00 E,1523810346.300000,
        1,0.484375,1.015625
2008-04-14T12:39:06,378 DEBUG [00000000] Logging.Appender.Boot
        (tk4aboot.c:228) - Destroying Boot appender <2008-04-14T12:39:06,
        378 DEBUG [00000000] Logging.Appender.ARM (tk4aarm4.c:797) -
        Destroying ARMAppender ARM
2008-04-14T12:39:06,378 DEBUG [00000000] Logging.Appender.File
        (tk4afile.c:562) - Destroying FileAppender ARM2LOG
2008-04-14T12:39:06,378 DEBUG [00000000] Logging.Appender.File (tk4afile.c:562)
-
        Destroying FileAppender ARM4LOG
```

Chapter 7

The ARM Logger

ARM Logger Overview

A logger is a named entity that identifies a message category. The logger includes a level and one or more appenders. The appenders process the log events for the message category. The name of the ARM message category is Perf.ARM. The level indicates the threshold (or lowest event) that will be processed for this message category.

Loggers are specified in log events. This associates the log event with a message category. By categorizing log events, messages of the same category are written to the same location. You configure loggers in a logging configuration file for SAS server logging. Or, you configure loggers by using SAS language elements in a DATA step or macro program. The following defines the Perf.ARM performance message logger in the configuration file:

```
<logger name="Perf.ARM" additivity="true">
   <level value="info"/>
   <appender-ref ref="ARM"/>
</logger>
```

For an example of a configuration file, see “SAS Logging Facility Configuration File” on page 115.

You create loggers in SAS programs using the following SAS language elements:

- %LOG4SAS, which initializes the autocall macro logging environment. For more information, see “Using Autocall Macros to Log Messages” in Chapter 11 of *SAS Logging: Configuration and Programming Reference*.
- %LOG4SAS_LOGGER() autocall macro for macro programming. For more information, see “Using Autocall Macros to Log Messages” in Chapter 11 of *SAS Logging: Configuration and Programming Reference*.
- LOG4SAS_LOGGER function in a DATA step. For more information, see “Using the Logging Facility Functions in the DATA Step” in Chapter 12 of *SAS Logging: Configuration and Programming Reference*.
- DCL logger object constructor in a DATA step. For more information, see “The Logger and Appender Component Object Interface” in Chapter 13 of *SAS Logging: Configuration and Programming Reference*.

Loggers that are created using SAS language elements exist for the duration of the SAS session. You define the ARM performance message logger using the following SAS language elements:

```
%log4sas();
%log4sas_logger(Perf.ARM, 'level=info');
```

For more information about loggers, see “Logger” in *SAS Logging: Configuration and Programming Reference*.

Chapter 8

ARM and SAS OLAP Server

Using ARM with SAS OLAP Server

A SAS OLAP Server is started as a system process and waits for clients to connect to it. Each client connection establishes a *session* on the SAS OLAP Server. Each session can then submit MDX *queries*. Typically, each MDX query is split into one or more *regions*, which translate calls against the OLAP cube's data, which are called *data queries*.

ARM writes SAS OLAP Server records to an ARM log using the SAS logging facility. There are two ways to activate ARM logging:

- on a running SAS OLAP Server. No server restart is necessary. Logging is valid for the currently running server only, and will not continue when the server is restarted.
- by making a change to the logconfig.xml file. Logging will activate automatically, but requires a server restart.

For information, see “Using ARM with SAS OLAP Server” in *SAS Intelligence Platform: System Administration Guide*.

Understanding the ARM Records Written for SAS OLAP Server

The initialization and termination records give you summary information for each SAS OLAP Server session. The ARM records for the SAS OLAP Server are written to the SAS logging facility or the ARM log. The following explains the output:

I (initialization) record
is an initialization record, with one record written per SAS OLAP Server invocation when the ARM subsystem is initialized. It starts with an I, followed by:

- the SAS datetime value of the SAS OLAP Server invocation
- an application ID
- a user CPU (start) time

- a system CPU (start) time
- an application name (OLAP_SERVER)

Output:

```
I,1320592800.822000,2,0.380547,0.630907,OLAP_SERVER,
```

E (end) record
is a termination record, with one record written per SAS OLAP Server termination. It starts with an E, followed by:

- the SAS datetime value of the SAS OLAP Server termination
- the application ID from the I record
- a user CPU (stop) time
- a system CPU (stop) time

Output:

```
E,1320592802.805000,2,1.281843,0.791137
```

Note: The E records are written for the OLAP_SESSION transaction, which records each SAS OLAP Server session that is started by a client connection. The E records provide the user ID of the client user that started the SAS OLAP Server session.

G (GetID) record
is an OLAP_SESSION transaction record, with one record written per SAS OLAP Server invocation. It starts with a G, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- a transaction class ID
- a transaction class name (OLAP_SESSION)
- a transaction class description (OLAP Session)
- a description of the values that are provided in subsequent S (start) and P (stop) records

Output:

```
G,1337615817.801000,2,1,OLAP_SESSION,OLAP Session,User
Name,LongStr
```

Note: **`User Name`** is the user ID of the client user that started the SAS OLAP Server session.

S (start) record
is a start record, with one record written for each SAS OLAP Server session. It starts with an S, followed by:

- the SAS datetime value when the SAS OLAP Server session started
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- a user CPU (start) time
- a system CPU (start) time
- the user ID of the client user

Output:

```
S,1337615818.502000,2,1,2,2.52952,0.901296,sasabc
```

P (stop) record
is a stop record, with one record written for each SAS OLAP Server session. It starts with a P, followed by:

- the SAS datetime value when the SAS OLAP Server session ended
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the associated S record
- a user CPU (stop) time
- a system CPU (stop) time
- the status 0=OK

Output:

```
P,1337615819.383000,2,1,2,2.113038,0.931339,0
```

U (update) record
is an update record, with one record written for each hierarchy in each OLAP cube. The U record for the OLAP_SESSION transaction is written only when the DATA_QUERY transaction occurs. It starts with a U, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the S record
- a user CPU time
- a system CPU time
- a buffer type (always 2, which indicates that 1,024 bytes of text follows)
- a string with the following format:
  - char(32) cube name
  - char(4) hierarchy number—used in the DATA_QUERY update record to identify region and aggregation
  - char(32) hierarchy name
  - char(4) number of hierarchy levels

Output:

```
U,1355324046.943000,2,1,2,1.625000,2.15625,2,SALES 1CUSTOMER
4 U,1355324046.943000,2,1,2,1.625000,2.15625,2,SALES
2PRODUCT 5U,1355324046.943000,2,1,2,1.625000,2.15625,2,SALES
3TIME 4
```

The following records are written for MDX_QUERY transactions, which log each query that is sent to the OLAP cube. These records provide the cube name and the size of the result set in cells:

G (GetID) record
is an MDX_QUERY transaction record, with one record written per SAS OLAP Server invocation. It starts with a G, followed by:

- the SAS datetime value when the record was written

- the application ID from the I record
- a transaction class ID
- a transaction name (MDX_QUERY)
- a transaction description (MDX Query)
- a description of the values that are provided in subsequent S or C (start) and P (stop) records

Output:

```
G,1320594857.747000,3,2,MDX_QUERY,MDX Query,Result Set
Size,Gauge32,Cube Name,LongStr
```

Note: **`Result Set Size`** is the size of the result set in cells. **`Cube Name`** is the name of the cube that is being queried.

S or C (start) record
is a start record, with one record written for each MDX_QUERY transaction. It starts with an S, followed by:

- the SAS datetime value when the MDX_QUERY started
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- a user CPU (start) time
- a system CPU (start) time

Output:

```
S,1320594857.787000,3,2,2,1.341929,0.731051
```

If the OLAP_SESSION level was also requested, then the MDX_QUERY transaction record is correlated to its parent session record, and starts with a C, followed by:

- the SAS datetime value when the MDX_QUERY started
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- the parent transaction class ID
- the parent transaction ID
- a user CPU (start) time
- a system CPU (start) time

Output:

```
C,1320594857.787000,3,2,2,1,2,1.341929,0.731051
```

P (stop) record
is a stop record, with one record written for each MDX_QUERY transaction. It starts with a P, followed by:

- the SAS datetime value when the MDX_QUERY ended
- the application ID from the I record
- the transaction class ID from the G record

- the transaction ID from the associated S or C record
- a user CPU (stop) time
- a system CPU (stop) time
- a status (0=OK, 2=query failed)
- the size of the result set in cells
- the name of the cube that was queried

Output:

```
P,1320594858.948000,3,2,2,2.52952,0.781123,0,5,MDDBCARS
```

The following records are written for DATA_QUERY transactions, which log each region execution (that is, each data retrieval from stored OLAP cube aggregations or from the cache). The following records provide region IDs, aggregate IDs, the number of returned records, the source type, and the thread index. The DATA_QUERY transaction is the primary input for both automatic and manual cube optimization.

G (GetID) record
is a DATA_QUERY transaction ID record, with one record written per SAS OLAP session. It starts with a G, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- a transaction class ID
- the transaction name (DATA_QUERY)
- the transaction description (Plug-in Call)
- a description of the values that are provided in subsequent S or C (start) and P (stop) records. The values are:
  - query aggregate
  - source aggregate
  - result set size
  - source type
  - thread index
  - cube name

Output:

```
G,1359310645.798000,2,4,DATA_QUERY,Plugin Call,Query
Aggregate,Id32,Source Aggregate,Id32,Result Set
Size,Gauge32,Source Type,Id32,Thread Index,Gauge32,Cube
Name,LongStr
```

S or C (start subquery) record
is a start subquery record, with one record written for each data access. It starts with an S, followed by:

- the SAS datetime value when the subquery started
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- the CPU (start) time

- the system CPU (start) time

Output:

```
S,1320596204.653000,2,2,2,1.51512,0.630907
```

If the MDX_QUERY level was also requested, then the DATA_QUERY transaction record is correlated to its parent MDX_QUERY record, and starts with a C, followed by:

- the SAS datetime value when the subquery started
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- the parent transaction class ID
- the parent transaction ID
- a user CPU (start) time
- a system CPU (start) time

Output:

```
C,1320596204.653000,2,2,2,1,1,1.51512,0.630907
```

P (stop subquery) record
is a stop subquery record, with one record written for each data access. It starts with a P, followed by:

- the SAS datetime value when the subquery ended
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the S or C record
- a user CPU (stop) time
- a system CPU (stop) time
- a status (0=OK, 2=subquery failed)
- a region sequential number (per update record)
- an aggregation sequential number (per update record)
- the size of the result set in records
- the plug-in type (0=CSPDS, 1=CSAS, 2=CACHE, 3=MOLAP)
- the thread index
- the cube name

Output:

```
P,
1320596205.485000,2,2,2,1.181699,0.670964,0,1,31,5,0,0,SA
LES
```

U (update) record
is an update record, with one record written for each new region and stored aggregation. It starts with a U, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record

- the transaction class ID from the G record
- the transaction ID from the S record
- a user CPU time
- a system CPU time
- a buffer type (always 2, which indicates that 1,024 bytes of text follows)
- a string with the following format:
  - char(32) cube name
  - char(16) unique sequential number—used in the DATA_QUERY stop record to identify region and aggregation
  - char(4) number of hierarchies in the region or aggregation repeated for each hierarchy in the region or aggregate:
  - char(4) hierarchy number (per OLAP_SESSION update record)
  - char(3) hierarchy level

Output:

```
U,1355324092.960000,2,3,61,6.93750,5.734375,2,SALES 4 1 1 1
```

The following update record is written for the MDX_STRING transaction, which writes an additional record for the MDX_QUERY transaction. The record contains the actual MDX query string.

U (update) record
is an update record, with one record written for each MDX_STRING transaction. It starts with a U, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the S record
- a user CPU time
- a system CPU time
- a buffer type (always 2, which indicates that 1,024 bytes of text follows)
- the actual MDX string

Output:

```
U,1320589796.262000,2,1,1,0.670964,2,SELECT {[DATE].
[DATEH].[ALL DATE]. CHILDREN} ON COLUMNS,
{[MEASURES].[MEASURES].[SALES_SUM]} ON ROWS FROM MDDBCARS
```

When using the SAS logging facility and ARM 2.0, these records are written to ARM2.RECORD.

Part 4

Logging Facility ARM Appender

Chapter 9

The ARM Appender

ARM Appender Overview

The ARM appender is a standard SAS logging facility appender, which is configured and customized for accessing performance data. The primary role of the ARM appender is to capture ARM transaction events, process the events, route the events to an output destination, and emit ARM 4.0 compatible events.

Key features of the ARM appender are:

- supports ARM 2.0 and ARM 4.0 standards
- supports default transaction correlation
- converts SAS 9.1 and earlier ARM transaction events into SAS logging facility events
- is controlled by the SAS logging facility configuration, pattern layouts, and output methods

The ARM appender can be defined in the SAS logging facility configuration file. The output destination can be a file appender, an external agent, or an internal SAS appender. You can also use the SAS logging facility in SAS programs.

Chapter 10

ARM Appender Syntax

ARMAppender Syntax

ARMAppender syntax is case sensitive.

XML Configuration

```
<appender class="FileAppender" name="ARM-log-name">
    <param name="File" value="file-name"/>
    <layout>
    <param name="ConversionPattern"
      value="%d,
      %X{App.Name},
      %X{ARM.Id},
      %X{ARM.GroupName},
      %X{ARM.TranName},
      %X{ARM.TranState},
      %X{ARM.TranId},
      %X{ARM.TranHandle},
      %X{ARM.ParentCorrelator},
      %X{ARM.CurrentCorrelator},
      %X{ARM.TranStatus},
      %X{ARM.TranStart.Time},
      %X{ARM.TranStop.Time},
      %X{ARM.TranBlocked.Time},
      %X{ARM.TranResp.Time}
      "/>
    </layout>
</appender>
```

```
<appender class="ARMAppender" name="ARM">
    <param name="Agent" value="libarm4"/>
      <param name="Encoding" value="encoding-value"/>
      <param name="GetTimes" value="TRUE | FALSE"/>
      <param name="ManageCorrelators" value= "TRUE | FALSE"/>
      <param name="AppName" value="application-name"/>
      <param name="GroupName" value="group-name"/>
      <appender-ref ref="ARM-log-names"/>

</appender>
```

ARMAppender Syntax Description

appender class="ARMAppender" name="ARM"
specifies ARM as the appender name. The ARMAppender name *must* be ARM.

Default None

Restriction Only a single instance of an ARMAppender per process.

Requirement Yes. ARM must be the name of the ARMAppender.

name="Agent" value="*library-name*"
specifies the name of the library that contains the external ARM 4.0 agent library that receives the events. See your vendor documentation for the correct library name. Two values that can be used:

value=" "
if no agent is specified, output is sent to any referenced appenders. In the syntax example, the output is sent to the file appender, "*ARM-log-name*".

value="*library-name*"
specifies the name of the library that contains the external ARM 4.0 agent library that receives the events.

Default Output is sent to any referenced appenders

Requirement No

name="AppName" value="*application-name*"
specifies the name of the application. The maximum length of the value is 128 characters, which includes the termination character (/). This value is sent to the ARM_REGISTER_APPLICATION() function call. To override this value, specify the SAS start-up option LOGAPPLNAME=*application-name*.

Default SAS

Requirement No

name="ConversionPattern" value="*conversion-pattern*"
specifies how the log event is written to the ARM log.

Default None. If a conversion pattern is not specified, then the log event produces an empty string.

Requirement No

name="Encoding" value="*encoding-value*"
specifies the type of character set encoding that is used for strings that are sent to and calls that are received by the ARM 4.0 agent library.

Default Native Unicode character set for the host, or UTF-8 as required by the ARM 4.0 standards.

Requirement No

name="File" value="*path-and-filename*"
specifies the path and filename of the file to which ARM messages are written.

Default None

Requirement Yes

name="GetTimes" value="TRUE | FALSE"
enables the ARM appender to compute transaction response time metrics.

TRUE
enables the appender to compute transaction response times.

FALSE
disables the appender to compute transaction response times.

Default False

Requirement No

name="ManageCorrelators" value="TRUE | FALSE"
specifies whether ARMAppender manages transaction correlation.

TRUE
enables automatic transaction correlation. The true value might affect existing benchmarks for ARM 2.0 records.

FALSE
enables the application to manage transaction correlation.

Default True

Requirement No

name="GroupName" value="*group-name*"
specifies the name of a group of application instances, if any. Application instances that are started with a common run-time purpose are candidates for using the same group name. The maximum length of the value is 256 characters. This value is passed to the ARM_START_APPLICATION() function call.

Default Current user ID if available, otherwise NULL

Requirement No

ARMAppender Example

The following example is a SAS logging facility configuration file that includes ARMAppender:

```
<?xml version="1.0" encoding="UTF-8"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">

<appender class="FileAppender" name="ARM2LOG">
   <param name="File" value="arm2.log"/>
   <param name="ImmediateFlush" value="true"/>
   <layout>
      <param name="ConversionPattern" value="%X{ARM2.Record}"/>
   </layout>
</appender>

<appender class="FileAppender" name="ARM4LOG">
   <param name="File" value="arm4.log"/>
   <param name="ImmediateFlush" value="true"/>
   <layout>
      <param name="ConversionPattern"
       value="%d,
       %12X{App.Name},
       %14X{ARM.GroupName},
       %12X{ARM.TranName},
       %8X{ARM.TranState},
       %8X{ARM.TranStatus},
       %20X{ARM.TranStart.Time},
       %20X{ARM.TranStop.Time},
       %56X{ARM.ParentCorrelator},
       %56X{ARM.CurrentCorrelator}
       "/>
   </layout>
</appender>

<appender class="ARMAppender" name="ARM">
   <param name="Encoding" value="UTF-8"/>
   <param name="GetTimes" value="true"/>
   <param name="ManageCorrelators" value="true"/>
   <param name="AppName" value="yourSampleApp"/>
   <param name="GroupName" value="SAS"/>
   <appender-ref ref="ARM4LOG"/>
   <appender-ref ref="ARM2LOG"/>
</appender>

<appender class="FileAppender" name="LOG">
   <param name="File" value="root.log"/>
   <param name="ImmediateFlush" value="true"/>
   <layout>
      <param name="ConversionPattern" value="%d %c %m"/>
   </layout>
</appender>

<logger name="Perf.ARM" additivity="false">
   <level value="info"/>
   <appender-ref ref="ARM"/>
</logger>

<root>
   <level value="info"/>
   <appender-ref ref="LOG"/>
```

```
</root>

</logging:configuration>
```

Chapter 11

ARM Appender Configuration Parameters

ARM Appender Configuration Parameters

The ARM appender includes unique configuration parameters to modify the appender execution behavior. These parameters are configured within the SAS logging facility configuration file. The location of the file is specified using the LOGCONFIGLOC= SAS system option. The syntax within the configuration file is:

```
<param name="<config.param>" value="<config_value>"
```

The following table lists the SAS logging facility ARM appender configuration parameters, values, and descriptions:

Table 11.1 *SAS 9.3 ARM Appender Configuration Parameters*

Configuration Parameter	Value	Description
Agent	value="" value="libarm4"	The value represents the name of an external ARM 4.0 agent library that receives ARM events. See your vendor documentation.
Encoding	value=""	The value represents the character set encoding to be used for strings passed to and calls received by the ARM agent library. The default is the native Unicode character set for the host or UTF-8 as required by the ARM standards.
GetTimes	value="TRUE" value="FALSE"	The TRUE value enables the appender to compute transaction response times. The default value is FALSE.

Configuration Parameter	Value	Description
ManageCorrelators	value="TRUE" value="FALSE"	The default value is TRUE. This option specifies whether the ARM appender manages transaction correlation. Setting the value to TRUE enables automatic transaction correlation. Setting this value to TRUE might affect existing benchmarks for ARM 2.0 records. Setting the value to FALSE enables the application to manage transaction correlation.
AppName	value="SAS"	The value represents the name of the application. The maximum length of the value is 127 characters. This value is passed to the ARM_REGISTER_APPLICATION() function call. To override this value, specify the SAS start-up option LOGAPPLNAME=*value*. The default value is SAS.
GroupName	value="*user ID*"	The value represents the name of a group of application instances, if any. Application instances that are started with a common run-time purpose are good candidates for using the same group name. The maximum length of the value is 256 characters. This value is passed to the ARM_START_APPLICATION() function call. The default value is the current user ID if available, otherwise NULL.

Chapter 12

ARM Appender Pattern Layouts for ARM Messages

ARM Appender Pattern Layouts for ARM Messages

The ARM appender includes options to create the output message format. The options are in the form of pattern layouts, which consist of a named diagnostic context recognized by the ARM appender. The following tables list the diagnostic context pattern layouts for the ARM appender.

All text is case sensitive in the SAS logging configuration file. Enter elements, names, attribute names, and literal values as they are shown in the ConversionPattern syntax.

Table 12.1 *Application Level Specifications*

Specification	Description
%X{App.Name}	Application name that is specified in the configuration file. A SAS ARM macro initializes the application call or the LOGAPPLNAME= option.
%X{ARM.AppHandle}	A unique ID that is associated with an application instance.
%X{ARM.AppId}	A unique ID that is associated with the application class.
%X{ARM.GroupName}	The group name of the application instances. The value is specified in the configuration file or user ID, if available.

Table 12.2 *Application Metric Specifications*

Specification	Description
%X{ARM.AppStart.Time}	The time-of-day value for the ARM application start event.

Specification	Description
%X{ARM.AppStart.System_CPU_Time}	The process system CPU time at the ARM application start event.
%X{ARM.AppStart.User_CPU_Time}	The process user CPU time at the ARM application start event.
%X{ARM.AppStop.Time}	The time-of-day value for the ARM application end event.
%X{ARM.AppStop.System_CPU_Time}	System CPU time for the ARM application end event.
%X{ARM.AppStop.User_CPU_Time}	User CPU time for the ARM application end event.

All text is case sensitive in the SAS logging configuration file. Enter elements, names, attribute names, and literal values as they are shown in the ConversionPattern syntax.

Table 12.3 *Transaction Level Specifications*

Specification	Description
%X{ARM.TranName}	Transaction name, specified in the appropriate ARM start transaction call.
%X{ARM.TranState}	The transaction state: init, start, stop, update, block, unblock, or discard.
%X{ARM.TranId}	A unique ID that is associated with the transaction class.
%X{ARM.TranHandle}	A unique ID that is associated with a transaction instance.
%X{ARM.CurrentCorrelator}	The transaction correlator for the transaction instance returned by the start transaction event.
%X{ARM.ParentCorrelator}	The transaction correlator for the direct ancestor transaction, automatically generated when the ManageCorrelators parameter is enabled.
%X{ARM.TranStatus}	The value specified for the stop transaction event, such as good, aborted, failed, or unknown.
%X{ARM.Userid}	The current user ID that is associated with the transaction.

Table 12.4 *Transaction Metric Specifications*

Specification	Description
%X{ARM.TranBlocked.Time}	The time-of-day value for the current transaction block event. (Reserved for future use.)
%X{ARM.TranStop.Time}	The time-of-day value for the current transaction stop event.
%X{ARM.IO_Count}*	The total number of process disk, tape, or related input and output operations for a transaction. The computed delta between start and stop transaction events.*
%X{ARM.System_CPU_Time}	Process current system CPU time for the ARM event.
%X{ARM.TimeStamp}	Current time-of-day value for the ARM event.
%X{ARM.TranBlocked.System_CPU_Time}	Process system CPU time for the current transaction block event. (Reserved for future use.)
%X{ARM.TranBlocked.User_CPU_Time}	Process user CPU time for the current transaction block event. (Reserved for future use.)
%X{ARM.TranResp.System_CPU_Time}	Calculated system CPU time for the duration of the transaction.
%X{ARM.TranResp.Time}	Calculated elapsed time for the duration of the transaction.
%X{ARM.TranResp.User_CPU_Time}	Calculated user CPU time for the duration of the transaction.
%X{ARM.TranStart.IO_Count}*	The total number of process disk, tape, or related input and output operations for the transaction event.*
%X{ARM.TranStart.Mem_Current}	Current process memory utilization for the transaction event.
%X{ARM.TranStart.Mem_High}	Process the highest amount of memory used for the transaction event.
%X{ARM.TranStart.System_CPU_Time}	Process system CPU time for the current transaction start event.
%X{ARM.TranStart.Thread_Current}	Current process thread count for the transaction event.
%X{ARM.TranStart.Thread_High }	Process the highest thread count for the transaction event.
%X{ARM.TranStart.Time}	The time-of-day value for the current transaction start event.

Specification	Description
%X{ARM.TranStart.User_CPU_Time}	Process user CPU time for the current transaction start event.
%X{ARM.TranStop.IO_Count}*	The total number of process disk, tape, or related input and output operations for the transaction event.*
%X{ARM.TranStop.Mem_Current}	Current process memory utilization for the transaction event.
%X{ARM.TranStop.Mem_High}	Process the highest amount of memory used for the transaction event.
%X{ARM.TranStop.System_CPU_Time}	Process system CPU time for the current transaction stop event.
%X{ARM.TranStop.Thread_Current}	Current process thread count for the transaction event.
%X{ARM.TranStop.Thread_High }	Process the highest thread count for the transaction event.
%X{ARM.TranStop.User_CPU_Time}	Process user CPU time for the current transaction stop event.
%X{ARM.User_CPU_Time}	Process current user CPU time for the ARM event.

* %X{ARM.IO_Count} is the computed delta between start and stop transaction events. %X{ARM.TranStart.IO_Count} and %X{ARM.TranStop.IO_Count} are just counts, not computed deltas.

Table 12.5 *ARM 2.0 Transaction Metric Specifications*

Specification	Description
%X{ARM2.Metric.Data}	ARM 2.0 format 1 metric data.
%X{ARM2.F2.Data}	ARM 2.0 format 2 data buffer contents for ARM update records.
%X{ARM2.Metric.MData}	ARM 2.0 format 101 metric metadata data buffer contents.
%X{ARM2.Record}	ARM 2.0 output record format message contents.
%X{ARM2.TranIdentity}	ARM 2.0 transaction detail information.

ARM 2.0 data buffers are changed to ARM 4.0 sub-buffers.

Table 12.6 *ARM Sub-Buffer Metric Values*

Specification	Description
%X{ARM.Metric*n*.Type}	ARM sub-buffer metric type (*n* value can be 1 ... 7).
%X{ARM.Metric*n*.Usage}	ARM sub-buffer metric usage (*n* value can be 1 ... 7), values are general, transaction, or transaction name.
%X{ARM.Metric*n*.Value}	ARM sub-buffer metric value (*n* value can be 1 ... 7).
%X{ARM.Metric*n*.Unit}	String description of the metric, such as files transferred (*n* value can be 1 ... 7).

ARM 2.0 data buffers are changed to ARM 4.0 sub-buffers.

Table 12.7 *ARM Transaction Identity and Context Sub-Buffers*

Specification	Description
%X{ARM.TranIdentity*n*.Name}	Optional transaction class name properties.
%X{ARM.TranIdentity*n*.Value}	Optional transaction class value properties.
%X{ARM.TranIdentity.URI}	Optional transaction URI.
%X{ARM.TranContext*n*.Name}	Optional transaction context instance name.
%X{ARM.TranContext*n*.Value}	Optional transaction context instance value.
%X{ARM.TranContext.URI}	Optional transaction context instance URI.

Chapter 13

ARM Category Table

ARM Category Table

Table 13.1 *ARM Categories and Descriptions*

ARM Categories	Description
ARM Macros	
"%ARMEND Macro" (p. 79)	Indicates the termination of an application.
"%ARMGTID Macro" (p. 81)	Assigns a unique identifier to a transaction class.
"%ARMINIT Macro" (p. 83)	Initializes an application.
"%ARMSTOP Macro" (p. 91)	Specifies the end of a transaction instance.
"%ARMSTRT Macro" (p. 88)	Specifies the start of a unique transaction, and returns a handle that is passed to the %ARMUPDT and %ARMSTOP macros.
"%ARMUPDT Macro" (p. 92)	Updates a transaction instance that was previously started.
ARM Performance Macros	
"%PERFEND Macro" (p. 95)	Indicates the termination of an application.
"%PERFINIT Macro" (p. 96)	Names the application instance and initializes the ARM interface.
"%PERFSTOP Macro" (p. 96)	Specifies the end of a transaction.
"%PERFSTRT Macro" (p. 97)	Specifies the start of a transaction.
ARM Post-processing Macros	

ARM Categories	Description
"%ARMJOIN Macro" (p. 85)	Reads the six SAS data sets that are created by the %ARMPROC macro, and creates SAS data sets and SQL views that contain common information about applications and transactions.
"%ARMPROC Macro" (p. 86)	Processes the ARM log, and creates six SAS data sets that contain the information from the log.
ARM System Options	
"ARMAGENT= System Option" (p. 103)	Specifies another vendor's ARM agent, which is an executable module or keyword (such as, &LOG4SAS) that contains a specific implementation of the ARM API.
"ARMLOC= System Option" (p. 104)	Specifies the location of the ARM log.
"ARMSUBSYS= System Option" (p. 105)	Specifies whether to initialize the ARM subsystems, which determine the internal SAS processing transactions to be monitored.

Part 5

Language Reference Dictionary

Chapter 14

ARM Macros

Introduction to ARM Macros

When you use ARM macros, you must define user metrics and correlators. You can use your existing programs that contain ARM macros and continue to get similar results written to the ARM log or written to the SAS logging facility. However, changing the ARM macros to performance macros is recommended when using the SAS logging facility. To compare the features of ARM macros and performance macros, see "Comparing the SAS 9.1 ARM Interface with the SAS 9.3 ARM Interface" on page 5.

Dictionary

%ARMEND Macro

Indicates the termination of an application.

Category: ARM Macro

Syntax

%ARMEND< (*option-1 <, ...option-n*>)>;

Optional Arguments

APPID=*numeric variable or constant*
is the application ID to use on the ARM_GETID function call. The value must be a numeric variable or constant.

Note: Use APPIDVAR= instead of APPID= in new applications. APPID= is obsolete.

APPIDVAR=*numeric variable*
is the application ID. The value must be a numeric variable.

LEVEL=*numeric variable or constant*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMEND macro to be issued in open code. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

Details

Use the %ARMEND macro when you are finished initiating new activity with the ARM interface. The %ARMEND macro is typically called when an application or user instance is terminating. Each %ARMEND macro is paired with an %ARMINIT. The %ARMEND macro means that the application does not issue any more ARM calls. ARM calls issued after an application has been terminated with the %ARMEND macro result in an error. All transaction class identifiers are cleared and are no longer available after the %ARMEND macro. Note that you must terminate ARM with an %ARMEND macro to avoid getting a warning or an error from the %ARMPROC macro.

The *input* is an application ID that is generated by a previous %ARMINIT macro. If the APPID= or APPIDVAR= option is provided, the specified value is used as the application ID. Otherwise, the value of the global macro variable _ARMAPID is used.

The *output* is the _ARMRC variable, which is the error status code that was returned from the ARM_END function call.

Examples

Example 1: Basic Usage

```
data _null_;
   %armend;
run;
```

Example 2: Supplying an Application ID Using APPIDVAR=

```
%let _armexec=1;
%let _armacro=1;
data _null_;
    %arminit(appname=application-name, appuser='sasxyz', appidvar=myapp);
```

```
run;
data _null_;
     %armend(appidvar=myapp);
run;
```

%ARMGTID Macro

Assigns a unique identifier to a transaction class.

Category: ARM Macro

Syntax

%ARMGTID (TXNNAME='*transaction-name*' < *option-1* <, ...*option-n*>>);

Required Argument

TXNNAME='*transaction-name*'
is a transaction name. The value is a SAS character variable or quoted literal value.

Restriction: The transaction name has a 127-character limit.

Optional Arguments

APPID=*numeric variable or constant*
is the application ID to use on the ARM_GETID function call. The value must be a numeric variable or constant.

Note: Use APPIDVAR= instead of APPID= in new applications. APPID= is obsolete.

APPIDVAR=*numeric variable*
is the application ID. The value must be a numeric variable.

LEVEL=*numeric constant or variable*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMINIT macro to be issued in open code, outside of a DATA step. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

METRNAM1–7='*name*'
is the name of the user-defined metric. The value must be a SAS character variable or quoted literal value.

Requirement: The name and user-defined metric definition must be specified.

METRDEF1–7=*option*
is the definition of the user-defined metric. The value must be one of the following:

COUNT32, COUNT64, or COUNTDIV
use the counter to sum the values of an interval. A counter can calculate average values, maximums, and minimums per transaction, and other statistics.

GAUGE32, GAUGE64, or GAUGEDIV
use the gauge when a sum of values is not needed. A gauge can calculate average values, maximums, and minimums per transaction, and other statistics.

ID32 or ID64
use the numeric ID as an identifier, but not as a measurement value. A numeric might be an error code or an employee ID. No calculations can be performed on the numeric ID.

SHORTSTR or LONGSTR
use the string ID as an identifier. No calculations can be performed on the string ID.

Restriction: METRDEF7= can equal only LONGSTR and can be a long string of 32 bytes. METRDEF1–6= cannot equal LONGSTR.

Requirement: The user name and user-defined metric definition must be specified.

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

TXNDET='*name*'
is a transaction detail. The value is a SAS character variable or quoted literal.

Restriction: The transaction detail has a 127-character limit.

TXNIDVAR=*numeric variable*
is a numeric variable that contains the value of the transaction ID.

Details

Use the %ARMGTID macro to name a transaction class. Transaction classes are related units of work within an application. One or more %ARMGTID macros are typically issued when the application starts to name each of the transaction classes used by the application. The %ARMGTID macro produces only one record for each transaction class, even if there are multiple %ARMGTID macros for the same transaction class.

The *input* is an application ID that is generated by a previous %ARMINIT macro. If the APPID= or APPIDVAR= option is provided, the specified value is used as the application ID. Otherwise, the value of the global macro variable _ARMAPID used.

The *output* is the _ARMTXID variable, which is the transaction class ID that was returned from the ARM_GETID function call. Any variable for the TXNIDVAR= option is updated.

Examples

Example 1: Basic Usage

```
data _null_;
  %armgtid(txnname='txn OE', txndet='Order Entry txn class');
run;
```

Example 2: Saving the Transaction ID

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname=application-name, appuser='sasxyz');
  %armgtid(txnname='txn OE', txndet='Order Entry txn class',
           txnidvar=txn1);
```

```
  put "transaction id is " txn1;
run;
```

%ARMINIT Macro

Initializes an application.

Category: ARM Macro

Syntax

%ARMINIT (APPNAME='*application-name*' <, *option-1* <, ...*option-n*>>);

Required Argument

APPNAME='*application-name*'
is the application name. The value is a SAS character variable or quoted literal.

Restriction: The application name has a 127-character limit.

Optional Arguments

APPIDVAR=*numeric variable*
is the application ID. The value must be a numeric variable.

APPUSER='*application-userID*'
is the application user ID. The value is a SAS character variable or quoted literal.

Restriction: The application user ID has a 127-character limit.

GETID=NO | YES
is optional and denotes whether to generate an ARM_GETID function call after ARM_INIT. If the value is YES, you can define the user metrics.

Default: NO

Requirement: TXNNAME= is required when you use GETID=YES.

LEVEL=*numeric constant or variable*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMINIT macro to be issued in open code. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

TXNIDVAR=*numeric variable*
is a numeric variable that contains the value of the transaction ID.

Restriction: Use TXNIDVAR= only when you use GETID=YES.

TXNDET='*name*'
is a transaction detail. The value is a SAS character variable or quoted literal.

Restriction: The transaction detail has a 127-character limit. Use TXNDET= only when you use GETID=YES.

TXNNAME='*transaction-name*'
is the transaction name. The value is a SAS character variable or quoted literal value.

Requirement: TXNNAME= is required only when you use GETID=YES.

Details

A %ARMINIT macro call names the application and the users of the application. In addition, it initializes the ARM interface if a previous %ARMINIT macro has not been issued. Typically, it is executed when the application initializes.

Note: You must globally enable ARM macros by setting the _ARMEXEC macro variable to a value of 1. For more information, see "Setting the _ARMEXEC Macro Variable" on page 19.

There is no *input*.

The output is the _ARMAPID variable, which is the application ID that was returned from the ARM_INIT function call. If GETID=YES, then _ARMTXID is returned also. Any variables for APPIDVAR= and TXNIDVAR= are updated.

Examples

Example 1: Basic Usage

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname='General Ledger');
run
```

Example 2: Supplying the User ID

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  name='Order Entry Application';
  %arminit(appname=application-name, appuser='sasxyz');
run;
```

Example 3: Generating an ARM_GETID in Addition to an ARM_INIT

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname='Warehouse App', getid=YES,
           txnname='Query 1', txndet='My long query');
run;
```

Example 4: Saving the Application ID

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname=application-name, appuser='sasxyz', appidvar=appl);
  put "application id is " appl;
run;
```

%ARMJOIN Macro

Reads the six SAS data sets that are created by the %ARMPROC macro, and creates SAS data sets and SQL views that contain common information about applications and transactions.

Category: ARM Postprocessing Macro

Restriction: Do not use with ARMAGENT=LOG4SAS.

Syntax

%ARMJOIN (<*option-1 <, ...option-n*>>);

Optional Arguments

LIBIN=*libref*
is the libref for the SAS library that contains the six SAS data sets that are created by the %ARMPROC macro.

Default: WORK

LIBOUT=*libref*
is the libref for the SAS library that contains the application and transaction data sets.

Default: WORK

Restriction: If a Read-only library is specified in the LIBOUT= option, an error message is written to the ARM log and processing is stopped.

TXNDS=YES | NO
specifies whether the transaction data sets are to be created.

Default: YES

UPDTDS=YES | NO
specifies whether the update data sets are to be created.

Default: YES

Details

The %ARMJOIN macro reads the six SAS data sets that are created by the %ARMPROC macro. It merges the information from those data sets to create data sets and SAS views for easier reporting of ARM data.

Note: The %ARMJOIN macro does not work from SCL. It must be run in a DATA step.

The *input* is the SAS data sets from the %ARMPROC macro. You must run the %ARMPROC macro before running the %ARMJOIN macro.

The *output* is a single SAS library that contains the following:

- information about applications (APP)
- a DATA step view that contains information about all start handles, including parent correlator class and parent start handles (TXNVIEW)
- a SAS view that contains information about all update transactions (UPDTVIEW)
- one transaction data set for each application
- one update data set for each application

The application data set is named APP and contains one observation for every application that is found in the input data. Each observation contains information such as application name, user ID, transaction counts, average application response time, and so on. In addition, each observation contains a numeric variable APPNO that is the identifier of the related transaction or update data set that contains more detailed transaction information.

The transaction data sets are named TXN1, TXN2, TXN3, and so on. Each data set corresponds to a single application, and each observation represents a single ARM transaction containing start and stop times, elapsed times, and CPU time.

The TXNVIEW view joins all transaction data sets into a single data set. Start handle elapsed time and CPU time are calculated from the start and stop transactions. If the start handle has a parent start handle, then the class ID and start handle of the parent are included using the variables PARCLS= and PARHDL=. If no parent is specified, these variables contain missing values.

The update data sets are named UPDT1, UPDT2, UPDT3, and so on. Each data set corresponds to a single application, and contains multiple observations for each ARM transaction. Each observation contains the ARM call datetime, an ARM call sequence ID, and, if applicable, elapsed time, CPU time, and update data.

The UPDTVIEW view joins all update data sets into a single data set.

The transaction data sets are easier to use for analyzing individual ARM transactions because all information about a transaction is represented in one observation. However, the transaction data sets do not contain any information from %ARMUPDT macros.

The update data sets are similar to the transaction data sets. However, information about a single transaction is spread over several observations. Update data sets contain logged data buffer information from all %ARMUPDT macros.

Examples

Example 1: Basic Usage

```
filename ARMLOG 'd:\armlog';
  %armproc();
  %armjoin();
```

Example 2: Defining a Permanent Library to Read %ARMPROC Macro Output and Store %ARMJOIN Macro Views

```
libname user 'c:\arm\user';
    %armjoin(libin=user,libout=user);
run;
```

%ARMPROC Macro

Processes the ARM log, and creates six SAS data sets that contain the information from the log.

Category: ARM Postprocessing Macro

Restriction: Do not use with ARMAGENT=LOG4SAS.

Syntax

%ARMPROC (<*option-1 <, ...option-n>>*);

Optional Arguments

LIB=*libref*
is the libref for the SAS library that contains the six SAS data sets.

Default: WORK

LOG=*pathname*
is the pathname for the physical location of the ARM log. If a pathname is not specified, you must pre-assign the ARMLOG fileref before calling the macro.

LOGNEW=*pathname*
is the pathname of the physical location of the new ARM log. It is used when ARM processing is resumed.

Details

The %ARMPROC macro reads an ARM log and creates six SAS data sets that contain the information from the log. This macro reads the variable name and value pairs from the ARM log as named input (VAR=VALUE). You should pre-assign the ARMLOG fileref before calling the macro or supply the LOG= option. If the ARMLOC= option is ignored, an actual FILENAME statement is required to pre-assign the ARMLOG fileref.

Note: The %ARMPROC macro does not work from SCL. A comma in the name of the log causes the log to be parsed incorrectly. A comma in the data of the UPDATE record does not cause any issues.

The *input* is the external file containing the ARM log.

The %ARMPROC macro creates six SAS data sets. These SAS data sets contain information from calls to the ARM API function calls. The following lists the six SAS data sets:

- INIT—contains information from all ARM_INIT calls
- GETID—contains information from all ARM_GETID calls
- START—contains information from all ARM_START calls
- UPDATE—contains information from all ARM_UPDATE calls
- STOP—contains information from all ARM_STOP calls
- END—contains information from all ARM_END calls

Examples

Example 1: Defining a Permanent Library to Store %ARMPROC Macro Output

```
libname user 'f:\arm\user';
    %armproc(lib=user);
run;
```

Example 2: Supplying the LIB= and LOG= Options

```
libname armout 'sas library name';
    %armproc(lib=armout,log=c:\userID\arm\armlog);
```

%ARMSTRT Macro

Specifies the start of a unique transaction, and returns a handle that is passed to the %ARMUPDT and %ARMSTOP macros.

Category: ARM Macro

Syntax

%ARMSTRT (*option-1 <, ...option-n>*);

Optional Arguments

APPID=*numeric variable or constant*
is the application ID to use on the ARM_GETID function call. The value must be a numeric variable or constant.

Restriction: Use APPID= only when you use GETID=YES. See the %ARMINIT macro for information about GETID=YES.

Note: Use APPIDVAR= instead of APPID= in new applications. APPID= is obsolete.

APPIDVAR=*numeric variable*
is the application ID. The value must be a numeric variable.

Restriction: Use APPIDVAR= only when you use GETID=YES. See the %ARMINIT macro for information about GETID=YES.

CORR=*n*
defines the type of parent and child transactions.

Default: 0

Requirement: Use CORR= only when you use correlators.

GETID=NO | YES
is optional and denotes whether to generate an ARM_GETID function call before ARM_START. If the value is YES, you can define the user metrics.

Default: NO

Requirement: TXNNAME= is required when you use GETID=YES.

LEVEL=*numeric constant or variable*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMSTRT macro to be issued in open code. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

METRVAL1–7='*name*'
is the value of the user-defined metric. The value must be a SAS character variable or a quoted literal. The value can be up to eight characters in length.

Requirement: The value of the user-defined metric must correspond to the user metric defined in the %ARMGTID macro.

PARNTVAR=*numeric variable*
is a numeric variable that contains the value of the parent transaction start handle. Use PARNTVAR= only when you define a child transaction and only when the CORR= option has a value of 2 or 3.

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

SHDLVAR=*numeric variable*
is a numeric variable that contains the value of the start handle. SHDLVAR= is required when you use correlators to define parent and child transactions.

TXNDET='*name*'
is a transaction detail. The value is a SAS character variable or quoted literal.

Restriction: The transaction detail has a 127-character limit.

Requirement: Use TXNDET= only when you use GETID=YES.

TXNID=*numeric variable or constant*
is the transaction ID to use in the ARM_START function call. The value must be a numeric variable or constant.

Note: Use TXNIDVAR= instead of TXNID= in new applications. TXNID= is obsolete.

TXNIDVAR=*numeric variable*
is a numeric variable that contains the value of the transaction ID when GETID=NO. It contains the value of the TXNID when GETID=YES.

TXNNAME='*transaction-name*'
is the transaction name. The value is a SAS character variable or quoted literal.

Restriction: The transaction name has a 127-character limit.

Requirement: TXNNAME= is required only when you use GETID=YES.

Details

The %ARMSTRT macro signals the start of a unique transaction, also known as a transaction instance. A transaction instance is an instantiation of a transaction class that was previously defined by the %ARMGTID macro. If user metrics are defined for a transaction class using the %ARMGTID macro, the values for the user metrics begin with the METRVAL1–7= option.

The CORR= option defines the type of parent (primary) and child (component) transactions using the following values:

0
not part of a related group

1
parent transaction

2
child transaction

3
child of one transaction and parent of another

Note: You use CORR= only when you use correlators.

Each child start handle variable must be defined with a parent start handle variable. Here is a code fragment that shows the use of correlator types and the SHLDVAR= and PARNTVAR= options:

```
%armstrt(txnidvar=txnid,corr=1,shdlvar=HDL100);
%armstrt(txnidvar=txnid,corr=0,shdlvar=HDL200<,...user metrics>);
%armstrt(txnidvar=txnid,corr=2,shldvar=HDL110,parntvar=HDL100);
%armstrt(txnidvar=txnid,corr=3,shldvar=HDL120,parntvar=HDL100);
```

The *input* is the transaction class ID that is generated by a previous %ARMGTID macro. If the TXNID= or TXNIDVAR= option is specified, the value is used as the transaction ID. Otherwise, the value of the global macro variable _ARMTXID is used.

If GETID=YES and the APPID= or APPIDVAR= options are supplied, the supplied value is used as the application ID. Otherwise, the value of the global macro variable _ARMAPID is used.

The *output* is the _ARMSHDL variable, which is the start handle that was returned from the ARM_START function call. If GETID=YES, then the _ARMTXID variable is updated. Any variables for TXNIDVAR= and SHDLVAR= are updated.

Examples

Example 1: Basic Usage

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit (appname='Forecast');
  %armgtid (txnname='Txn 1A', txndet='Forecasting Txn Class');
  %armstrt;
run;
```

Example 2: Supplying the Transaction ID Using TXNIDVAR=

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname=application-name, appuser='sasxyz');
  %armgtid(txnname='txn OE', txndet= 'Order Entry txn class'
           txnidvar=txnnum);
data _null_;
  %armstrt(txnidvar=txnname);
run;
```

Example 3: Generating an ARM_GETID Call and an ARM_START

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname='Forecast', appidvar=savapp);
run;
data _null_;
  %armstrt(getid=YES, txnname='Txn 1A',
           txndet='Forecasting Txn Class',
           appidvar=savapp);
run;
```

%ARMSTOP Macro

Specifies the end of a transaction instance.

Category: ARM Macro

Syntax

%ARMSTOP (<*option1 <, ...option-n>>*);

Optional Arguments

LEVEL=*numeric constant or variable*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMSTOP macro to be issued in open code. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

METRVAL1–7='*name*'
is the value of the user-defined metric. The value must be a SAS character variable or a quoted literal value up to eight characters in length.

Requirement: The value of the user-defined metric must correspond to the user metric defined in %ARMGTID.

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

SHANDLE=*numeric variable or constant*
is the start handle to use on the ARM_UPDATE function call. The value must be a numeric variable or constant.

SHDLVAR=*numeric variable*
is a numeric variable that contains the value of the start handle.

STATUS=*numeric variable or numeric constant*
is a transaction status value to pass to the ARM_STOP function call. The value must be a numeric variable or numeric constant 0, 1, or 2. The default is 0.

Default: NO

Details

The %ARMSTOP macro signals the end of a transaction that was started using an %ARMSTRT macro.

The *input* is a start handle that is generated by a previous %ARMSTRT macro. If the SHANDLE= or SHDLVAR= option is specified, the value is used as the start handle. Otherwise, the value of the global macro variable _ARMSHDL is used.

The *output* is the _ARMRC variable, which contains the error status code that was returned from the ARM_STOP function call.

Examples

Example 1: Basic Usage

```
data _null_;
  %armstop;   /* status  defaults to zero*/
run;
```

Example 2: Supplying a Nonzero Status

```
data _null_;
  rc=2;
  %armstop(status=rc);
run;
```

Example 3: Supplying a Start Handle Using SHDLVAR=

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname=application-name, appuser='sasxyz');
  %armgtid(txnname='txn OE', txndet='Order Entry txn class');
  %armstrt(shdlvar=sh1);
run;
data _null_;
  %armstop(shdlvar=sh1);
run;
```

%ARMUPDT Macro

Updates a transaction instance that was previously started.

Category: ARM Macro

Syntax

%ARMUPDT (*DATA=<option>* , *<option-1 <, ...option-n>>*);

Required Argument

DATA='*variable*'

is a SAS character variable or a quoted literal from the user-supplied data buffer that contains text to pass to the ARM_UPDATE function call. DATA= is not required, but it is highly recommended. This information is mutually exclusive of user-defined metric values.

Restriction: The data value has a 1,020-character limit.

Optional Arguments

LEVEL=*numeric constant or variable*
is a variable that specifies the execution level. The value must be a numeric constant or variable.

MACONLY=NO | YES
enables the %ARMUPDT macro to be issued in open code. You set the value to YES if the macro can be issued in open code, and NO if it can be issued only in a DATA step.

Default: NO

METRVAL1–7='*name*'
is the value of the user-defined metric. The value must be a SAS character variable or a quoted literal. The value can be up to eight characters in length. The value is ignored if the DATA= option is used.

Requirement: The value of the user-defined metric must correspond to the user metric defined in the %ARMGTID macro.

SCL=NO | YES
is used only in SCL programs and specifies whether the macro is in an SCL environment. Set the value to YES if the macro is in an SCL environment, and NO if it is not.

Default: NO

SHANDLE=*numeric or constant*
is the start handle to use on the ARM_UPDATE function call. The value must be a numeric or constant.

Note: Use SHDLVAR= instead of SHANDLE= in new applications. SHANDLE= is obsolete.

SHDLVAR=*numeric variable*
is a numeric variable that contains the value of the start handle.

Details

The %ARMUPDT macro can be executed multiple times after the %ARMSTRT macro and before the %ARMSTOP macro. It enables you to get information about the transaction instance that is in progress. The %ARMUPDT macro provides a snapshot of information for the transaction instance.

The *input* is a start handle that is generated by a previous %ARMSTRT macro. If the SHANDLE= or SHDLVAR= option is specified, the value is used as the start handle. Otherwise, the value of the global macro variable _ARMSHDL is used.

Note: User-metric values and user-supplied data buffers are mutually exclusive parameters. Each requires its own update call to get both types of data into the update records.

The *output* is the _ARMRC variable, which contains the error status code that was returned from the ARM_UPDATE function call.

Examples

Example 1: Basic Usage

```
data _null_;
  updtdata='Txn still running at' || put (time(),time.);
```

```
  %armupdt(data=updtdata);
run;
```

Example 2: Supplying a Start Handle Using SHDLVAR=

```
%let _armexec=1;
%let _armacro=1;
data _null_;
  %arminit(appname=applicaiton-name, appuser='sasxyz');
  %armgtid(txnname='txn OE', txndet='Order Entry txn class');
  %armstrt(shdlvar=sh1);
run;
data _null_;
  %armupdt(data='OE txn pre-processing complete', shdlvar=sh1 );
run;
```

Chapter 15

ARM Performance Macros

Introduction to ARM Performance Macros

Four ARM performance (PERF) macros replace eight ARM macros. With performance macros, you do not have to specify user-metric definitions, types, and values in your code.

Dictionary

%PERFEND Macro

Indicates the termination of the application.

Category: ARM Performance Macro

Restriction: SAS 9.3 and later

Syntax

%PERFEND;

Details

There are no input parameters for the %PERFEND macro.

Use the %PERFEND macro to terminate an application or user event. Each %PERFEND macro is paired with one %PERFINIT macro to mark the end of an application. The

%PERFEND means that the application does not issue any more ARM calls. ARM calls issued after an application has been terminated with the %PERFEND macro result in an error. All transaction class identifiers are cleared and are no longer available after the %PERFEND macro. For an example, see "ARM Performance Macros" on page 98.

Note: If the %PERFEND macro is not set, the ARM application terminates at the end of the SAS session and you receive a warning.

%PERFINIT Macro

Names the application instance and initializes the ARM interface.

Category: ARM Performance Macro

Restriction: SAS 9.3 and later

Syntax

%PERFINIT (APPNAME='*application-name*');

%PERFINIT (APPLNAME='*application-name*');

Required Argument

APPNAME='*application-name*'; APPLNAME='*application-name*';
is the application name, which must be a SAS character variable or a literal enclosed in single or double quotation marks. If both APPNAME= and APPLNAME= arguments are used, APPLNAME= takes precedence. If neither APPNAME= or APPLNAME= argument is used, then the value of SAS is the default.

Restriction: The application name has a 127-character limit.

Details

The %PERFINIT macro names the application and initializes the ARM interface if a previous %PERFINIT macro has not been issued. Typically, it is executed when the application initializes. For an example, see "ARM Performance Macros" on page 98.

Note: You must globally enable ARM macros by setting the _ARMEXEC macro variable to a value of 1. For more information, see "Setting the _ARMEXEC Macro Variable" on page 19.

%PERFSTOP Macro

Specifies the end of a transaction.

Category: ARM Performance Macro

Restriction: SAS 9.3 and later

Syntax

%PERFSTOP;

Details

There are no input parameters for the %PERFSTOP macro.

Use the %PERFSTOP macro to signal the end of a transaction that was started using the %PERFSTRT macro. The %PERFSTOP macro contains default user metrics. To see the relationships between %PERFSTOP and the default user metrics, see "Default User Metrics and Performance Macros" on page 31. For an example, see "ARM Performance Macros" on page 98.

The %PERFSTRT and %PERFSTOP macros can be nested in other %PERFSTRT and %PERFSTOP macros. When nested, each %PERFSTOP macro that is initiated is paired with the currently active %PERFSTRT macro. In the figure on page 98, there are three %PERFSTRT and %PERFSTOP macro pairs. The first %PERFSTOP macro terminates the transaction for the third %PERFSTRT macro, and so on. For an example, see "ARM Performance Macros" on page 98.

%PERFSTRT Macro

Specifies the start of a transaction.

Category: ARM Performance Macro

Restriction: SAS 9.3 and later

Syntax

%PERFSTRT (TXNNAME='*transaction-name*');

Required Argument

TXNNAME='*transaction-name*'
is the transaction name, which must be a SAS character variable or a literal enclosed in single or double quotation marks.

Restriction: The transaction name has a 127-character limit.

Details

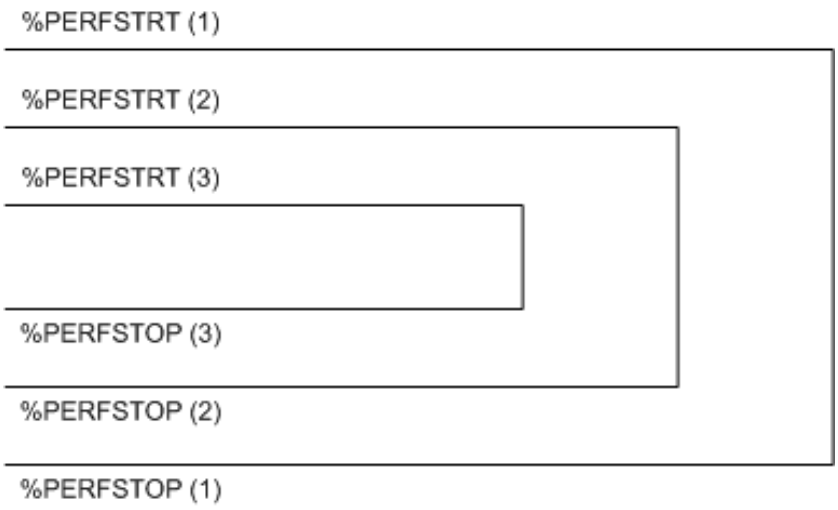
Use the %PERFSTRT macro to signal the start of a transaction. The %PERFSTRT macro contains default user metrics. To see the relationships between %PERFSTRT and the default user metrics, see "Default User Metrics and Performance Macros" on page 31. For an example, see "ARM Performance Macros" on page 98.

The %PERFSTRT and %PERFSTOP macros can be nested in other %PERFSTRT and %PERFSTOP macros. When nested, each %PERFSTOP macro that is initiated is paired with the currently active %PERFSTRT macro. In the following figure, there are three %PERFSTRT and %PERFSTOP macro pairs. The first %PERFSTOP macro terminates the transaction for the third %PERFSTRT macro, and so on. The following code creates nested macro pairs:

```
/*
* Enable ARM subsystem; if needed, set the _armexec=1
*/
%let _armexec=1;
%PERFINIT(APPNAME='SAS 9.3 Studio Test Application');
%PERFSTRT(TXNNAME='SAS 9.3 Studio Transaction One');
%PERFSTRT(TXNNAME='First Nested Macro Pair');
```

```
%PERFSTRT(TXNNAME='Second Nested Macro Pair');
/*
* *
SAS code that represents a discrete unit of work,
* *
*/
%PERFSTOP;        /* PERFSTOP for the Second Nested Macro Pair */
%PERFSTOP;        /* PERFSTOP for the First Nested Macro Pair */
%PERFSTOP;        /* PERFSTOP for the SAS 9.3 Studio Transaction One */
%PERFEND;
```

Figure 15.1 *Nested %PERFSTRT and %PERFSTOP Macros*



Example: ARM Performance Macros

The following example code writes output to the SAS log. The application name is **Perf_App**. There are two transactions, **Perf_Tran_1** and **Perf_Tran_2**. The performance macros are highlighted within the code.

```
%log4sas();
%log4sas_logger(Perf.ARM, 'level=info');
options armagent=log4sas;
%let _armexec=1;
%perfinit(appname="Perf_App");

%perfstrt(txnname="Perf_Tran_1");
  data x;
  do i=1 to 10000;
  x=i; y=0-i;
  output;
  end;
run;

  proc sort  data=x threads; by y;
run;
```

```
%perfstop;

%perfstrt(txnname="Perf_Tran_2");
  data x;
  do i=1 to 10000;
  x=i; y=0-i;
  output;
  end;
run;
  proc sort  data=x threads; by y;
run;
%perfstop;

%perfend;
run;
```

The following is the SAS log output for the example code. Note the performance macros that were highlighted in the example code.

Output 15.1 *SAS Log*

```
1    %log4sas();
2    %log4sas_logger(Perf.ARM, 'level=info');
3    options armagent=log4sas;
NOTE: INIT SAS 13eec00 I,1533837807.104000,1,0.468750,2.437500,SAS,
NOTE: REGISTER SAS 13eee58 G,1533837807.386000,1,1,SAS,MVA SAS session
NOTE: START SAS 13eee58 0 S,1533837807.386000,1,1,1,0.468750,2.437500
4    %let _armexec = 1;
5    %perfinit(appname="Perf_App");
NOTE: INIT Perf_App 13efa30
I,1533837840.315000,2,0.500000,2.484375,Perf_App,userID
6
7    %perfstrt(txnname="Perf_Tran_1");
NOTE: REGISTER Perf_Tran_1 13efc88
      G,1533837840.487000,2,2,Perf_Tran_1,,_IOCOUNT_,Count64,_MEMCURR_,
      Gauge64,_MEMHIGH_,Gauge64, THREADCURR_,Gauge32,_THREADHIGH_,Gauge32
NOTE: START Perf_Tran_1 13efc88
0

S,1533837840.487000,2,2,2,0.515625,2.500000,68346709,5095424,5095424,3,3
8      data x;
9      do i=1 to 10000;
10     x=i; y=0-i;
11     output;
12     end; run;

NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           5.67 seconds
      cpu time            0.17 seconds


13
14     proc sort  data=x threads; by y;  run;

NOTE: There were 10000 observations read from the data set WORK.X.
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           3.78 seconds
      cpu time            0.14 seconds


15   %perfstop;
NOTE: STOP Perf_Tran_1 13efc88 0

P,1533837851.224000,2,2,2,0.562500,2.812500,0,79383442,6144000,7606272,3,7
16
17   %perfstrt(txnname="Perf_Tran_2");
NOTE: REGISTER Perf_Tran_2 13f0338
      G,1533837851.239000,2,3,Perf_Tran_2,,_IOCOUNT_,Count64,_MEMCURR_,
      Gauge64,_MEMHIGH_,Gauge64,_THREADCURR_,Gauge32,_THREADHIGH_,
      Gauge32
NOTE: START Perf_Tran_2 13f0338
0

S,1533837851.239000,2,3,3,0.578125,2.812500,79395730,6144000,7606272,3,7
18     data x;
19     do i=1 to 10000;
20     x=i; y=0-i;
21     output;
22     end; run;
```

```
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


23     proc sort  data=x threads; by y;  run;

NOTE: There were 10000 observations read from the data set WORK.X.
NOTE: The data set WORK.X has 10000 observations and 3 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.09 seconds
      cpu time            0.04 seconds


24   %perfstop;
NOTE: STOP Perf_Tran_2 13f0338 0

P,1533837851.364000,2,3,3,0.625000,2.843750,0,80203806,6144000,7606272,3,7
25
26   %perfend;
NOTE: END Perf_App 13efa30
E,1533837851.458000,2,0.625000,2.859375
27   run;
```

Chapter 16

ARM System Options

Dictionary

ARMAGENT= System Option

Specifies another vendor's ARM agent, which is an executable module or keyword (such as, LOG4SAS), that contains a specific implementation of the ARM API.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: System administration: Performance

PROC OPTIONS GROUP= PERFORMANCE

Restrictions: After you initialize the ARM subsystem, you cannot specify a different ARM agent using ARMAGENT=.

If ARMAGENT=LOG4SAS, ARMLOC= is ignored.

See: ARMAGENT= System Option under z/OS in the documentation for your operating environment.

Syntax

ARMAGENT=*module*

Syntax Description

module
is the name of the executable module that contains an ARM agent, which is a program that contains a vendor's implementation of the ARM API.

Operating Environment Information

The maximum length for the module name is specific to your operating environment. For many operating environments, the maximum length is 32 characters. For the z/OS operating environment, see *SAS 9.3 Companion to z/OS*.

Default: for SAS 9.3 ARM interface: SAS

Details

An ARM agent is an executable module that contains an implementation of the ARM API. The ARM agent contains executable routines that are called from an application. The ARM agent and SAS run concurrently. SAS passes transaction information to the ARM agent, which collects, manages, and writes the ARM records to the ARM log or the SAS logging facility. SAS and other vendors provide an ARM agent.

By default, SAS uses ARMAGENT=SAS. Use ARMAGENT= to specify another executable module or keyword to monitor the internal SAS processing transactions (using ARMSUBSYS=) and user-defined transactions using ARM macros. If you specify ARMAGENT=LOG4SAS, the output is sent to the SAS logging facility, which enables you to have several logs.

See Also

System Options

- "ARMLOC= System Option" on page 104
- "ARMSUBSYS= System Option" on page 105

ARMLOC= System Option

Specifies the location of the ARM log.

Valid in: configuration file, SAS invocation, OPTIONS statement, System Options window

Category: System administration: Performance

Restriction: If ARMAGENT=LOG4SAS, ARMLOC= is ignored.

Syntax

ARMLOC=*fileref* | *filename*

Syntax Description

fileref
is a SAS name that is associated with the physical location of the ARM log. To assign a fileref, use the FILENAME statement.

'*filename*'
is the physical location of the log. Include the complete pathname and the filename. You can use single or double quotation marks.

Default: for SAS 9.1 ARM Interface: ARMLOG.LOG

for 9.3 ARM Interface: none

Restriction: For all operating environments except z/OS, if you specify the ARMLOC= system option in your configuration file, you must specify the filename, not a fileref.

Details

The ARM log is an external output file that contains the logged ARM transaction records. The ARM log gathers transaction information for the internal SAS processing transactions (depending on the value of the ARMSUBSYS= system option) and for user-defined transactions (using ARM macros).

You can change the location of the ARM log after initializing an ARM subsystem. Any records that were written to the ARM log in the old location are not copied to the ARM log in the new location. Therefore, you should issue ARMLOC= before initializing the first ARMSUBSYS= so that all records are written to the same ARM log.

See Also

System Options

- "ARMAGENT= System Option" on page 103
- "ARMSUBSYS= System Option" on page 105

ARMSUBSYS= System Option

Specifies whether to initialize the ARM subsystems, which determine the internal SAS processing transactions to be monitored.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: System administration: Performance

Default: (ARM_NONE)

Restriction: After you initialize the ARM subsystems, you cannot specify a different ARM agent using ARMAGENT=.

Syntax

ARMSUBSYS=(ARM_NONE | ARM_ALL | *subsystem1 <item1<item2<...>>>* <, *subsystem2<item1<item2<...>>>>* <OFF>)

Syntax Description

ARM_NONE
specifies that no internal SAS processing transactions are written to the SAS logging facility or the ARM log. This is the default setting.

ARM_ALL
specifies that all internal SAS processing transactions are written to the SAS logging facility or the ARM log.

subsystem
specifies an ARM subsystem, which is a group of internal SAS processing transactions that are to be written to the SAS logging facility or the ARM log. The following subsystems are available:

ARM_DSIO
collects SAS data set I/O processing information.

ARM_PROC
collects SAS procedure and DATA step processing information.

item
specifies a name that determines the type and amount of transaction logging for each subsystem. Use item specifications as filters so that only the information that you are interested in is logged. For example, if you want one type of transaction, list the single item. If you want multiple transactions for a subsystem, list each item. Items are associated with each subsystem as follows:

ARM_DSIO

OPENCLOSE
logs a SAS data set open and close transactions as a start record when a data set is opened, and as a stop record when it is closed.

VARDEF
logs OPENCLOSE records and an update record for each defined variable (output opens).

VARSEL
logs OPENCLOSE records and an update record for each selected variable (input and update opens).

VARINFO
logs OPENCLOSE, VARDEF, and VARSEL records.

WHEREOPT
logs OPENCLOSE records and an update record for each selected index from a WHERE processing optimization. Available for the default Base SAS engine and the V6 compatibility engine only.

WHEREINFO
logs OPENCLOSE, WHEREOPT, and WHERETXT records.

WHERETXT
logs OPENCLOSE records and one or more update records that contain a textual representation of the active WHERE expression. Each record can hold approximately 1,000 bytes.

MIN
logs the minimum amount of information. For SAS 9, MIN logs the OPENCLOSE records.

MAX
logs the maximum amount of information. For SAS 9 and later, MAX logs all of the ARM_DSIO records. This is the default for ARM_DSIO.

LEVEL1
logs OPENCLOSE, VARDEF, and VARSEL records.

LEVEL2
logs LEVEL1, WHEREOPT, and WHERETXT records.

For more information about the logged records, see "Understanding the Records Written by the ARM_DSIO Subsystem" on page 107.

ARM_PROC
For more information about the logged records, see "Understanding the Records Written by the ARM_PROC Subsystem" on page 109.

OFF
disables the specified subsystem. In the following code, all subsystems are enabled for the DATA step, and then the ARM_PROC subsystem is disabled for the PRINT procedure:

```
options armsubsys=(arm_all);
data a;
   x=1;
run;
options armsubsys=(arm_proc off);
proc print;
run;
```

Details

Overview of ARM Subsystems

The ARMSUBSYS= system option specifies whether to initialize the ARM subsystems, which determine the internal SAS processing transactions to be monitored. An ARM subsystem is a group of internal SAS processing transactions. When using the SAS logging facility and ARM 2.0, the records are routed to ARM2.Record.

If you want to specify a different ARM log location by using the ARMLOC= system option, be sure to issue the ARMLOC= option before you initialize an ARM subsystem. The subsystem start record is written to the new ARM log using the ARM2.Record pattern layout. For more information about pattern layouts, see "ARM Appender Pattern Layouts for ARM Messages" on page 69.

Understanding the Records Written by the ARM_DSIO Subsystem

The ARM_DSIO subsystem writes records to the SAS logging facility or the ARM log. This subsystem collects SAS data set I/O processing information. The records that are written to the SAS logging facility or the ARM log are:

I (initialization) record
is an initialization record, with one record written per session when the ARM subsystem is initialized. It starts with an I, followed by:

- the SAS datetime value for session start
- an application ID
- a user start time
- a system start time
- an application name
- a user ID

Output
I,1326479452.427000,1,1.171684,1.532203,SAS,sasabc"

G (GetID) record
is a transaction ID record, with one record written per transaction. It starts with a G, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- a transaction class ID
- a transaction name

- a transaction description
- a description of the values provided in subsequent S (start) and P (stop) records

Output:

```
G,1326479452.447000,1,1,MVA_DSIO.OPEN_CLOSE,DATA SET OPEN/CLOSE,
   LIBNAME,ShortStr,MEMTYPE,ShortStr,MEMNAME,LongStr
```

LIBNAME refers to the libref for a SAS library, MEMTYPE refers to the member type (DATA or VIEW), and MEMNAME refers to a SAS data set name.

S (start) record
is a start record, with one record written each time a file is opened. It starts with an S, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- the transaction class ID from the G record
- a transaction ID
- a user (start) time
- a system (start) time
- the actual libref, member type, and member name of the opened file

Output:

```
S,1326479486.396000,1,1,1,1.311886,2.22908,WORK,DATA,GRADES
```

P (stop) record
is a stop record, with one record written each time a file is closed. It starts with a P, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the associated S record
- a user (stop) time
- a system (stop) time
- the actual libref, member type, and member name of the closed file

Output:

```
P,1326479486.706000,1,1,1,1.331915,2.22908,0,WORK DATA,GRADES
```

U (update) record
is an update record, with one record written each time a variable is defined or selected, and each time an index is used during WHERE processing optimization. A U record displays the text of a WHERE expression. It starts with a U, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- the transaction class ID from the G record
- the transaction ID from the associated S record
- a user start time
- a system start time

- the detailed information for the type of U record being written

For variable definition and selection, the variable type is specified with a 1 for a numeric variable, or with a 2 for a character variable. The variable type with the name of the variable are followed by DEF for definition or SEL for selection.

Output:

```
U,1326479486.406000,1,1,1,1.321900,2.22908,2,VAR(2,Student),DEF
U,1326479508.508000,1,1,2,1.612318,2.443513,2,VAR(2,Student),SEL
```

For index selection, the index type is specified with an S for simple, or with a C for complex, followed by the name of the index.

```
U,1326479606.48000,1,1,4,2.403456,3.915630,2,INDEX(S,Test1),SEL
```

For WHERE expression text information, the expression is specified as:

```
U,1326479606.48000,1,1,4,2.403456,3.915630,2,WHERE(0),test1>60
```

E (end) record
is an end record, with one record written per session. It starts with an E, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- a user stop time
- a system stop time

Output:

```
E,1326480210.737000,1,2.533643,4.25788
```

Understanding the Records Written by the ARM_PROC Subsystem

The ARM_PROC subsystem writes records to the SAS logging facility or the ARM log. This subsystem collects SAS procedure and DATA step processing information. The records that are written to the SAS logging facility or the ARM log are:

G (GetID) record
is a transaction ID record, with one record written per transaction. It starts with a G, followed by:

- the SAS datetime value when the record was written
- the application ID from the I record
- a transaction class ID
- a transaction name
- a transaction description
- a description of the values that are provided in subsequent S (start) and P (stop) records

Output:

```
G,1501177361.426000,1,2,PROCEDURE,PROC START/STOP,PROC_NAME,ShortStr,
     PROC_IO,Count64,PROC_MEM,Count64,PROC_LABEL,LongStr
```

S (start) record
is a start record, with one record written immediately before the procedure executes. It starts with an S, followed by:

- the SAS datetime value when the record was written
- an application ID

- a transaction class ID
- a transaction ID
- a user CPU (start) time
- a system CPU (start) time
- the procedure or DATA step name
- the procedure or I/O count
- the amount of memory used
- the label name

Output:

```
S,1501177361.436000,1,2,2,0.350504,0.620892,DATASTEP,0,0,
GLMSTEPONE
```

P (stop) record
is a stop record, with one record written when the procedure terminates. It starts with a P, followed by:

- the SAS datetime value when the record was written
- an application ID
- a transaction class ID
- a transaction ID from the associated S record
- a user CPU (stop) time
- a system CPU (stop) time
- the procedure or DATA step name
- the procedure or I/O count
- the amount of memory used
- the label name

Output:

```
P,1501177361.776000,1,2,2,0.510734,0.741065,0,DATASTEP,8123483,333792,
GLMSTEPONE
```

Example

The following example shows the ARM subsystem ARM_DSIO, which collects SAS data set I/O processing information. The OPENCLOSE item logs the SAS data set open and close transaction.

```
options armsubsys=(ARM_DSIO OPENCLOSE);
```

The following example shows the ARM subsystem ARM_ALL, which specifies that all internal SAS processing transactions are written to the SAS logging facility or the ARM log.

```
options
   armagent=SAS
   armsubsys=arm_all;
```

See Also

System Options

- “ARMAGENT= System Option” on page 103
- “ARMLOC= System Option” on page 104

Part 6

Appendices

Appendix 1

SAS Logging Facility Configuration File

The following SAS logging facility configuration file represents a possible SAS logging facility XML configuration, which contains specifications for the ARM appender. The file is customized using standard XML language syntax, and includes the following definitions or parameters:

- <appender> statement: creates an instance of an appender
- <logger> statement: creates a named logger to receive SAS logging facility events
- <root> statement: by default, a root logger is required and it receives all SAS logging facility messages
- <appender-ref> statement: references an appender, used in a <logger> or <appender> definition
- <param> statement: optional parameter definitions for an appender, can be used for unique appender options or output format specifications
- Class="<value>" statement: defines a type of appender, such as FileAppender or ARMAppender
- %d, %c, %m, %X specifications: optional pattern conversion specifiers that map to the SAS logging facility values; see *SAS Logging: Configuration and Programming Reference* for a complete range and description of format specifiers.

Note: XML is case sensitive.

```
<?xml version="1.0" encoding="UTF-8"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">

  <appender class="FileAppender" name="ARM2LOG">
             <param name="File" value="arm2.log"/>
             <param name="ImmediateFlush" value="true"/>
             <layout>
                 <param name="ConversionPattern"
                          value="%X{ARM2.Record}"/>
              </layout>
  </appender>
  <appender name="ARM4LOG" class="FileAppender">
             <param name="File" value="arm4.log"/>
             <param name="ImmediateFlush" value="true"/>
             <layout>
                 <param name="ConversionPattern"
                       value="%d,
                       %12X{App.Name},
                       %14X{ARM.GroupName},
                       %12X{ARM.TranName},
                       %8X{ARM.TranState},
```

```
                        %8X{ARM.TranStatus},
                        %20X{ARM.TranStart.Time},
                        %20X{ARM.TranStop.Time},
                        %56X{ARM.ParentCorrelator},
                        %56X{ARM.CurrentCorrelator}
                      "/>
            </layout>
  </appender>

  <appender class="ARMAppender" name="ARM">
            <param name="Encoding" value="UTF-8"/>
            <param name="GetTimes" value="true"/>
            <param name="ManageCorrelators" value="true"/>
            <param name="AppName" value="yourSampleApp"/>
            <param name="GroupName" value="SAS"/>
            <appender-ref ref="ARM4LOG"/>
            <appender-ref ref="ARM2LOG"/>
  </appender>

  <appender class="FileAppender" name="LOG">
            <param name="File" value="root.log"/>
            <param name="ImmediateFlush" value="true"/>
            <layout>
                <param name="ConversionPattern"
                     value="%d{yyyyMMdd:HH.mm.ss,SS} %c %m"/>
            <layout>
  </appender>

  <logger name="Perf.ARM" additivity="false">
          <level value="info"/>
          <appender-ref ref="ARM"/>
  </logger>

  <root>
          <level value="info"/>
          <appender-ref ref="LOG"/>
  </root>

</logging:configuration>
```

Glossary

Application Response Measurement
the name of an application programming interface that was developed by an industry partnership and which is used to monitor the availability and performance of software applications. ARM monitors the application tasks that are important to a particular business. Short form: ARM.

ARM
See Application Response Measurement.

ARM agent
a software vendor's implementation of the ARM API. Each ARM agent is a set of executable routines that can be called by applications. The ARM agent runs concurrently with SAS. The SAS application passes transaction information to the agent, which collects the ARM transaction records and writes them to the ARM log.

ARM appender
a standard logging facility appender, which is configured and customized for accessing performance data. The primary role of the ARM appender is to record ARM transaction events, process the events, and route the events to a specified output destination.

ARM log
an external file that contains records of ARM transactions.

ARM macro
a macro that measures the response time of an application. ARM macros invoke ARM API function calls. They permit conditional execution by setting the appropriate macro parameters and macro variables. ARM macros are not part of the SAS macro facility.

ARM performance macros
SAS macros that enable you to identify transactions that you want to log using the SAS logging facility. Insert ARM performance macros in your SAS program at strategic points.

ARM subsystem
a group of internal SAS processing transactions such as PROC and DATA step processing and file input/output processing. You use the ARM system option ARMSUBSYS= to turn on a subsystem or all subsystems.

ARM system options
a group of SAS system options that control various aspects of the SAS ARM interface.

pattern layout
a template that you create to format log messages. The pattern layout identifies the type, order, and format of the data that is generated in a log event and delivered as output.

SAS ARM interface
an interface that can be used to monitor the performance of SAS applications. In the SAS ARM interface, the ARM API is implemented as an ARM agent. In addition, SAS supplies ARM macros, which generate calls to the ARM API function calls, and ARM system options, which enable you to manage the ARM environment and to log internal SAS processing transactions.

transaction
a unit of work that is meaningful for monitoring an application's performance. A transaction can be started and stopped one or more times within a single execution of an application. For example, in a SAS application, a transaction could be a step that updates a customer database. In SAS/MDDB Server software, a transaction might be a query on a subcube. Another type of transaction could be internal SAS processing that you want to monitor, such as how many times a SAS file is opened and closed or how long it takes to process a DATA step.

Index