

SAS[®] 9.3 Logging: Configuration and Programming Reference

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® 9.3 Logging: Configuration and Programming Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Logging: Configuration and Programming Reference

Copyright © 2011, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hardcopy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

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.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contents

What's New in the SAS 9.3 Logging Facility v

PART 1 SAS Logging 1

Chapter 1 • The SAS Logging Facility	3
Accessibility Features of the SAS Logging Facility	4
Overview of the SAS Logging Facility	4
Logging Facility Terminology	5
How the Logging Facility Works	6
Loggers	7
Appenders	11
Logging Thresholds	16
Formatting Messages	17
Message Filtering	17
Using the SAS Logging Facility in the SAS Intelligence Platform	18
Chapter 2 • Enabling the SAS Logging Facility	21
Enabling the Logging Facility for SAS Server Logging	21
Enabling the Logging Facility in SAS Programs	21
Naming a SAS Session	22
Chapter 3 • System Options for Enabling the SAS Logging Facility	23
Dictionary	23

PART 2 XML Configuration Overview and Reference 27

Chapter 4 • Overview of the Logging Configuration File	29
Typographical Conventions	29
Syntax Conventions	29
XML Elements for Configuring SAS Logging	30
Structure of the Logging Configuration File	33
Sample Configuration Files	34
Modifying or Replacing the Logging Configuration File	34
Error Messages for Logging Configuration File Errors	35
Chapter 5 • Logger Reference	37
Dictionary	37
Chapter 6 • Appender Reference	41
Dictionary	41
Chapter 7 • Database Connection Options Reference for DBAppender	89
Dictionary	89
Chapter 8 • Pattern Layouts	95

About Pattern Layouts	95
Dictionary	104
Chapter 9 • Filters	121
Overview of Filters	121
Filter Examples	123
Dictionary	124
PART 3 The Logging Facility for SAS Programs 131	
Chapter 10 • The SAS Logging Facility in the SAS Language	133
Overview of the SAS Logging Facility in the SAS Language	133
Initializing the SAS Logging Facility for SAS Programs	134
Creating and Using Appenders in a SAS Program	135
Creating Loggers in a SAS Program	136
Creating Log Events in a SAS Program	138
Example of Creating Logger and Appender Categories	139
Chapter 11 • Autocall Macro Reference	141
Using Autocall Macros to Log Messages	141
Example of Using Autocall Macros to Log Messages	142
Dictionary	145
Chapter 12 • Function Reference	155
Using the Logging Facility Functions in the DATA Step	155
Logging Example Using Functions	156
Dictionary	159
Chapter 13 • Component Object Reference	165
The Logger and Appender Component Object Interface	165
Dot Notation and DATA Step Component Objects	166
Dictionary	167
PART 4 Appendix 181	
Appendix 1 • Audit Messages for SAS Library Access	183
About Audit Messages for SAS Library Access	183
Sample: XML Logger and Appender	184
Sample Code to Look for Log Files and Build an Audit Report Data Set	185
Glossary	189
Index	193

What's New in the SAS 9.3 Logging Facility

Overview

The logging facility has the following changes and enhancements:

- support for modifying or replacing the logging configuration file without restarting SAS
- ability to restrict modification of the logging configuration using the SAS language
- new error messages for logging configuration file errors
- new appenders to write log events for third-party DBMS, Java classes, and Java Messaging Service (JMS)
- ability to audit access to SAS libraries
- new conversion patterns and new header and footer conversion patterns that SAS supplies
- filter enhancements

Support for Modifying and Replacing the Logging Configuration File

You can now modify or replace the logging configuration file without restarting SAS. For more information, see [“Modifying or Replacing the Logging Configuration File”](#) on page 34.

Restrict Modification of the Logging Configuration

The new logger attribute, IMMUTABILITY, can be set to restrict modification of the logging configuration by users who use the logging facility autocall macros, functions, and component objects in SAS programs. If IMMUTABILITY is set to FALSE, the SAS

language can be used to modify the logger settings for additivity and level. For more information, see [“Loggers” on page 7](#) and [“Logger” on page 37](#).

Logging Configuration File Error Messages

When an appender or logger cannot be created or configured, SAS issues more specific messages. For more information, see [“Error Messages for Logging Configuration File Errors” on page 35](#).

New Appenders

The following appenders have been added:

- DBAppender writes log events to a SAS table or to a table in a third-party DBMS. For more information, see [“DBAppender” on page 46](#).
- JavaAppender sends messages to a custom Java class. For more information, see [“JavaAppender” on page 60](#).
- JMSAppender sends messages to a message queue by using the Java Message Service (JMS) interfaces. For more information, see [“JMSAppender” on page 67](#).

Audit the Access to SAS Libraries

When you use the new logger `Audit.Data.Dataset.Open` and the `%E` conversion character, log messages can include SAS library information such as the `libref`, the engine assigned to the library, the library member and member type, the mode the library was opened for, and the path to the library. For more information, see [“About Audit Messages for SAS Library Access” on page 183](#) and [“E Conversion Character” on page 109](#).

Conversion Pattern Enhancements

The new `uuid` conversion character reports the unique identifier for the log event. For more information, see [“uuid Conversion Character” on page 116](#).

The new severity conversion character translates logging facility levels to Common Base Event (CBE) and Web Services Distributed Management Event Format (WEF) severity codes. For more information, see [“severity Conversion Character” on page 112](#).

The `%d` conversion character can now be specified in `HeaderPattern` and `FooterPattern` layout parameters to capture date information. For more information, see [“Syntax for a Pattern Layout” on page 104](#).

The %E conversion character can be used to add audit data to an audit log. For more information, see [“E Conversion Character” on page 109](#) and [Appendix 1, “Audit Messages for SAS Library Access,” on page 183](#).

You can now specify a default value for the %S conversion character. The default value is used as the value for %S when a specified key cannot be found. For more information, see [“S Conversion Character” on page 113](#).

SAS now provides several conversion patterns that you can specify as values for the HeaderPattern, FooterPattern, and ConversionPattern parameters in appender definitions. You specify the name of a SAS conversion pattern in place of a conversion pattern that you would normally specify. For more information, see [“Conversion Patterns Supplied by SAS” on page 96](#).

Filter Enhancements

The RepeatMatchFilter inhibits logging repeated messages if the immediate prior log message is identical to the current log message for an appender. For more information, see [“Overview of Filters” on page 121](#) and [“RepeatMatchFilter” on page 129](#).

Part 1

SAS Logging

<i>Chapter 1</i>	
The SAS Logging Facility	3
<i>Chapter 2</i>	
Enabling the SAS Logging Facility	21
<i>Chapter 3</i>	
System Options for Enabling the SAS Logging Facility	23

Chapter 1

The SAS Logging Facility

Accessibility Features of the SAS Logging Facility	4
Overview of the SAS Logging Facility	4
What Is the Logging Facility?	4
Who Uses the Logging Facility?	4
Comparing the SAS Logging Facility and the SAS Log	4
Logging Facility Terminology	5
How the Logging Facility Works	6
Setting Up the Logging Process	6
The Logging Process	6
Loggers	7
What Is a Logger?	7
XML Elements for Configuring Loggers	8
Hierarchical Logger Names	9
SAS Server Logger Names	9
Loggers in the SAS Language	11
Appenders	11
Appender Overview	11
XML Elements for Configuring Appenders	12
SAS Appenders for Server Logging	14
Appenders in the SAS Language	15
Referencing Appenders in a Logger	15
Logging Thresholds	16
Formatting Messages	17
Message Filtering	17
Using the SAS Logging Facility in the SAS Intelligence Platform	18
About the Initial Logging Configuration for SAS Servers	18
Viewing SAS Logging Messages and Adjusting Logging Levels in Client Applications	19
Best Practices for SAS Server Logging	19

Accessibility Features of the SAS Logging Facility

For information about accessibility for any of the products mentioned in this book, see the online Help for that product.

If you have questions or concerns about the accessibility of SAS products, send e-mail to accessibility@sas.com.

Overview of the SAS Logging Facility

What Is the Logging Facility?

The SAS 9.3 logging facility is a flexible, 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. The logging facility has the following features:

- Log events are categorized using a hierarchical naming system that enables you to configure logging at a broad or a fine-grained level.
- Log events can be directed to multiple output destinations, including fixed files, rolling files, operating system facilities, client applications, database tables, message queues, and custom Java classes. For each output destination, you can specify the following logging facility components:
 - the categories and levels of log events to report
 - the message layout, including the types of data to be included, the order of the data, and the format of the data
 - filters based on criteria such as diagnostic levels and message content
- Logging levels can be adjusted dynamically without starting and stopping processes.
- Performance-related log events can be generated for processing by the Application Response Measurement (ARM) 4.0 agent.

The logging facility is used by most SAS server processes. You can also use the logging facility within SAS programs.

Who Uses the Logging Facility?

This guide is for both administrators, who configure the SAS logging facility, and for programmers, who can use the logging facility in their SAS programs.

Auditors use this guide to understand the logging process and the information that the logging facility generates.

Comparing the SAS Logging Facility and the SAS Log

The SAS logging facility and the SAS log are two different logging systems within SAS.

Traditionally, the SAS log displays information, warning, and error messages as a result of executing SAS programs or SAS global statements. Regardless of their origin, all messages are destined for a single log.

By contrast, the SAS logging facility is a framework that categorizes and filters log messages in SAS server and SAS programming environments, and writes log messages to various output devices. In the server environment, the logging facility logs messages based on predefined message categories, such as Admin for administrative messages, App for application messages, and Perf for performance messages. Messages for a category can be written to files, consoles, and other system destinations simultaneously. The logging facility also enables messages to be filtered based on the following thresholds: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

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

Logging Facility Terminology

Here are the common terms that this document uses:

appender

a named entity that represents a specific output destination for messages. Destinations include fixed files, rolling files, operating system facilities, client applications, database tables, message queues, and custom Java classes. You can configure appenders by specifying thresholds, filters, log directories and filenames, pattern layouts, and other parameters that control how messages are written to the destination.

filter

a set of character strings or thresholds, or a combination of strings and thresholds that you specify. Log events are compared to the filter to determine whether they should be processed.

level

the diagnostic level that is associated with a log event. The levels, from lowest to highest, are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

log event

an occurrence that is reported by a program for possible inclusion in a log.

logger

a named entity that identifies a message category. Loggers are named using a hierarchical system that enables you to configure logging at a broad or a fine-grained level.

The logging facility includes a set of high-level loggers for SAS servers, including Audit, Admin, App, IOM, and Perf. Some loggers are subdivided into lower-level (child) loggers. For example, the Audit logger has descendant loggers called Audit.Meta and Audit.Authentication, and Audit.Meta has descendant loggers called Audit.Meta.Security and Audit.Meta.Updates. The Root logger is the highest-level logger and does not represent a specific message category.

Loggers inherit settings from their higher-level (ancestor) loggers.

logging configuration

an XML file or a set of SAS program statements that determines how log events are processed. You use the logging configuration to assign thresholds to loggers, to configure appenders, and to specify which categories and levels of log events are to be written to each appender.

If you perform a planned deployment, then the SAS Deployment Wizard provides default logging configuration files for your SAS servers.

message category

a classification for messages that are produced by a SAS sub-system. Message categories for the logging facility are administrative messages, application-specific messages, audit messages, IOM messages, and performance messages.

pattern layout

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

threshold

the lowest event level that is processed. Log events whose levels are below the threshold are ignored.

How the Logging Facility Works

Setting Up the Logging Process

To use the SAS logging facility, you must set up your logging environment:

- Define a logging configuration, which configures appenders and loggers. You can define the configuration by setting up an XML file or by using SAS language elements. If you perform a planned deployment, then logging configuration files are provided for your SAS servers.
- Specify the LOGCONFIGLOC= system option to enable logging, if you are using configuration files. If you perform a planned deployment, then this system option is included in the SAS configuration files for your SAS servers.
- Issue log events in a format that can be processed by the logging facility, if you are developing your own SAS programs.

Note: You can experiment with the logging facility without creating logging configuration files. SAS provides three basic logging configurations that write messages to either a file, the current console, or to the traditional SAS log using the root logger. For more information, see [“LOGCONFIGLOC= System Option” on page 24](#).

The Logging Process

After your logging environment is in place, the SAS logging facility begins processing as follows:

1. A SAS process (for example, a SAS server process) issues a log event. Each event includes the following attributes: a name that indicates the message category, a diagnostic level, and a message that describes the context for the event.

2. The logging facility receives the log event and determines which logger to assign it to, based on the event's name attribute.
3. The log event's level is compared to the threshold that is specified for the logger in the logging configuration. If the event's level is at or above the specified threshold, then processing continues. If the level is below the threshold, then the event is ignored.

If no threshold is specified for the event's logger, then the event inherits the threshold setting of the nearest ancestor logger. For example, if an Audit.Meta.Security event is being processed, then inheritance occurs as follows:

- a. The event's level is compared to the threshold for the Audit.Meta.Security logger.
- b. If no threshold is specified for Audit.Meta.Security, then the threshold for Audit.Meta is applied.
- c. If no threshold is specified for Audit.Meta, then the threshold for Audit is applied.
- d. If no threshold is specified for Audit, then the threshold for Root is applied.

If no thresholds are assigned to the logger or its ancestors, then the event is ignored.

4. The log event is processed by the appenders that are assigned to the logger and any of its ancestors in the logging configuration. For example, an Audit.Meta.Security event is processed by the appenders that are assigned to the following loggers: Audit.Meta.Security, Audit.Meta, Audit, and Root.

Each of these appenders processes the event according to the appender's configuration as specified in the logging configuration. Appender processing is performed as follows:

- a. If the appender configuration includes a threshold, then the event's level is compared to the threshold. If the event's level is at or above the threshold, then processing continues. If the level is below the threshold, then processing stops.
- b. If the appender configuration includes a filter, then the event is compared to the filtering criteria. Processing either continues or stops depending on the results of the comparison.
- c. The event is written to the output destination using the specifications that are defined in the appender configuration. Appender specifications include parameters such as pattern layouts, log directories, log filenames, rolling policies, locales, and encoding.

Loggers

What Is a Logger?

A logger is a named entity that identifies a message category. A logger's attributes consist of a level and one or more appenders that process the log events for the message category. The level indicates the threshold, or lowest event level, that will be processed for this message category.

Loggers are specified in log events to associate the log event with a message category. By categorizing log events, the logger can write messages of the same category to the same destinations. When a log event occurs, the log event message is processed by the

appender that is associated with the logger that is named in the event log if the log event level is the same or higher than the level that is specified for the logger.

Loggers are organized hierarchically and inherit the attributes of their ancestor logger. Hierarchical logger names are separated by a period (.) (for example, Admin.Meta.Security). The root logger is the highest level logger. All loggers inherit the root logger's attributes. The logging configuration file defines several message categories that are immediate descendants of the root logger. These high-level categories, Admin, App, Audit, IOM, and Perf, are used for SAS server logging and can be referenced by log events in SAS programs.

You configure loggers in a logging configuration file for SAS server logging or by using SAS language elements in a DATA step or macro program. If you perform a planned deployment, then the SAS Deployment Wizard provides logging configuration files for your SAS servers. You can dynamically adjust thresholds by using the server management features of SAS Management Console. For more information, see “Administering Logging for SAS Servers” in the *SAS Intelligence Platform: System Administration Guide*.

For more information, see “Logging Thresholds” on page 16 and “Appenders” on page 11.

XML Elements for Configuring Loggers

In a logging configuration file, a logger has the following structure:

```
<logger name="logger-name" additivity="TRUE | FALSE"
  immutability="TRUE | FALSE">
  <level value=threshold/>
  <appender-ref ref="appender-name"/>
</logger>
```

Syntax Description:

name="logger-name"

specifies the name of a message category name. The logger name is specified in a log event to associate a message with a message category.

additivity="TRUE | FALSE"

specifies whether to pass the log event to loggers in the hierarchy.

immutability="TRUE | FALSE"

specifies whether the logger's additivity and level settings are permanent or whether they can be changed by using the SAS language. If the value of IMMUTABILITY is TRUE, no changes can be made to the logger's additivity and level settings. If the value of IMMUTABILITY is FALSE, the logger level and the additivity setting can be changed using the SAS language. IMMUTABILITY is ignored for configuration changes made by administrators using SAS Management Console or the IOMOPERATE procedure.

level value="threshold"

specifies one of the following levels, from lowest to highest: TRACE, DEBUG, INFO, WARN, ERROR, FATAL. You use the threshold to filter log events. If the log event diagnostic level is the same or higher than the threshold that is specified for the log event's logger, the logging facility continues to process the log event. If the log event diagnostic level is lower than the logger's threshold, the log event is ignored.

appender-ref ref="appender-name"

specifies the name of an appender to record messages for this logger's message category.

Hierarchical Logger Names

The logger architecture enables logger names to be multiple levels so that descendant loggers can inherit thresholds and appender references from their parent loggers, therefore omitting the appender reference and threshold in the descendant logger definition. You separate hierarchical logger names with a period (.).

For example, suppose that your logging facility configuration file defines the Admin logger with an appender reference value of **MyRollingFile** and a threshold of **Info**. A second logger definition, Admin.MyPgm, specifies the logger name and a threshold of **Debug**. Because no appender reference is specified in Admin.MyPgm, the appender reference is inherited from its parent, the Admin logger. The appender reference **MyRollingFile** logs messages for Admin log events whose level is INFO or higher, as well as Admin.MyPgm log events whose level is DEBUG or higher.

These loggers might be defined using the following logger elements in the logging configuration file:

```
<logger name="Admin">
  <level value="Info"/>
  <appender-ref ref="MyRollingFile"/>
</logger>

<logger name="Admin.MyPgm">
  <level value="Debug"/>
</logger>

<root>
  <level value="Error"/>
  <appender-ref ref="SystemRollingFile">
</root>
```

If a log event specifies a hierarchical logger name that does not exist, the logging facility checks for a parent logger definition. If the parent logger exists, the log event is processed by the parent logger. If a logger definition does not exist for the parent, the root logger processes the log event.

Consider the example logger definitions in this section. If a log event specifies the logger Admin.Special, the logging facility determines that the logger Admin.Special does not exist. The logging facility then checks for the Admin logger. In this case, the Admin logger exists and the log event is processed by the Admin logger. If the Admin logger was not defined, the root logger would process the log event.

SAS Server Logger Names

Log events for SAS servers use a hierarchical logger name where each name in the hierarchy identifies a category such as an operation, a server, and a server operation. For example, log events that specify the Admin.OLAP.Security logger indicate that the message is an OLAP server security message that is intended for a system administrator or computer operator.

SAS server logger names begin with one of the following logger categories:

Admin

processes log events that are relevant to system administrators or computer operators.

App

processes log events that are related to specific applications. For example, metadata servers, OLAP servers, stored process servers, and workspace servers use loggers that are named *App.class.interface.method* to record method calls that are issued to the server.

Audit

processes log events to be used for auditing. These events include updates to public metadata objects, user access to SAS libraries, accepted and rejected user authentication requests, and administration of users, groups, and access controls.

IOM

processes log events for servers that use the Integrated Object Model (IOM). The IOM interface provides access to Foundation SAS features such as the SAS language, SAS libraries, the server file system, results content, and formatting services. IOM servers include metadata servers, OLAP servers, stored process servers, and workspace servers.

Perf

processes log events that are related to system performance.

The second category in a hierarchical logger name can indicate a type of server or some type of event, such as authentication. In most cases, however, the categories are self-explanatory. The following list gives some examples of server categories for the logging facility.

Logging Facility Server Category	SAS Server
Connect	SAS/CONNECT Server
Meta	SAS Metadata Server
ObjectSpawner	SAS Object Spawner
OLAP	SAS OLAP Server

Here is a list of some of the loggers that the logging facility uses for SAS servers:

Admin.Operations

processes log events that are related to server operations, such as starting, pausing, and stopping an instance of a workspace server.

Admin.Session

processes log events that are related to starting and stopping batch, windowing, and SAS/CONNECT server sessions.

Audit.Authentication

processes log events for server authentication requests.

Audit.Data.Dataset.Open

processes log events that are related to users' access to SAS libraries.

App.Program

processes log events that are related to running a program using the SAS language.

IOM

processes log events that are related to client interactions.

IOM.PE

processes log events that are related to packets that are processed by the BRIDGE and COM protocol engines.

Perf.ARM

processes log events that are related to ARM 4.0 transactions.

SAS issues an error message if a logger cannot be configured or accessed.

Loggers in the SAS Language

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

- %log4sas_logger() autocall macro for macro programming
- log4sas_logger function in a DATA step
- Declare Logger object constructor statement in a DATA step

See the following reference documents for information about defining loggers in the SAS language:

- [“%LOG4SAS_LOGGER Autocall Macro” on page 147](#)
- [“LOG4SAS_LOGGER Function” on page 161](#)
- [“DECLARE Statement, Logger Object” on page 172](#)

If you are writing SAS programs, you can write log events for loggers that are defined in one of the logging configuration files or you can write log events for loggers that you create by using the SAS language.

Loggers that are created by using the SAS language exist for the duration of the SAS session.

Appendix

Appender Overview

An appender is a named entity that is referenced by a logger. An appender specifies the destination for the message, specifies how the message is formatted, specifies attributes for the appender class, and provides additional filtering capabilities.

When a log event occurs, the logging facility processes the message by using the appender that is named in the logger's <appender-ref> element in a logging facility configuration file, or in the APPENDER-REF argument of a logger language element in a SAS program.

SAS has several appender classes for processing messages:

- appenders to log messages to an operating system console
- an IOM server appender to log messages from any IOM server
- file appenders for writing log messages to a file on disk
- appenders to write to Windows, UNIX, and z/OS operating system logs
- an appender to write messages to a message queue

For a complete list and description of the SAS server appenders, see [“SAS Appenders for Server Logging” on page 14](#).

You define appenders in the logging configuration file or in a SAS program by using a SAS function, autocall macro, or DATA step component object. An appender definition requires an appender class and name and the required parameters for the appender class. To customize the message, you specify the message layout within the appender definition. In a logging facility configuration file, you can include additional filtering arguments in the appender definition.

Logger definitions in SAS programs can reference appenders that are defined in a SAS program or any of the SAS server appenders.

For more information, see [“Creating and Using Appenders in a SAS Program ” on page 135](#).

XML Elements for Configuring Appenders

General Appender Syntax

In a logging configuration file, the appender has the following general structure:

```
<appender class="appender-class" name="appender-name">
  [ <param name="parameter-name" value="parameter-value"/>-1
  ... <param name="parameter-name" value="parameter-value"/>-n ]
  [ <layout>
    <param name="Header" value="header-text"/>
    <param name="HeaderPattern" value="conversion-pattern"/>
    <param name="ConversionPattern" value="conversion-pattern"/>
    <param name="Footer" value="footer-text"/>
    <param name="FooterPattern" value="conversion-pattern"/>
    <param name="XMLEscape" value="TRUE | FALSE"/>
  </layout> ]
  [ <filter>
    <filter-definitions>
  </filter> ]
</appender>
```

The brackets ([]) indicate that the element is optional.

Syntax Description:

class="appender-class"

The appender class is a type of appender. The following appender classes can be used in the logging facility:

- ARMAppender
- ConsoleAppender
- DBAppender
- FileAppender
- FilteringAppender
- IOMServerAppender
- JavaAppender
- JMSAppender
- RollingFileAppender

- sLogAppender
- UNXFacilityAppender
- WindowsEventAppender
- ZOSFacilityAppender
- ZOSWtoAppender

See: [Chapter 6, “Appender Reference,” on page 41](#)

name="appender-name"

The appender name is a user-specified name for the appender. An appender is associated with a logger when the appender name is specified as the value of the logger's appender-ref attribute.

<param name="parameter-name" value="parameter-value"/>

Most appenders have required parameters for specifying information that is relevant to the appender. Many parameter names are unique for individual appenders. The name= attribute specifies the name of the parameter, and the value= attribute specifies the value for the parameter.

For example, appenders that write messages to a user-specified file use the File parameter, where the value of the File parameter specifies the location and name of the log file.

See: [Chapter 6, “Appender Reference,” on page 41](#) for each appender's required parameters

<layout>

<param name="ConversionPattern" value="conversion-pattern"/>

<param name="Header" value="literal-string"/>

<param name="HeaderPattern" value="conversion-pattern"/>

<param name="Footer " value="literal-string"/>

<param name="FooterPattern" value="conversion-pattern"/>

<param name="XMLEscape" value="true | false"/>

</layout>

You use the <layout> elements to specify how messages should be formatted in the log. The conversion pattern is a series of conversion characters that represent the types of data to include in the log. For example, use the conversion characters %d %t %m to include the date and time, the thread identifier, and the message, respectively, in the log.

You can use the Header, HeaderPattern, Footer, and FooterPattern parameters to specify the conversion characters that appear at the top and the bottom of the log. You can use the XMLEscape parameter to specify whether certain characters (for example, "<") are converted to their entity representation, which in this case would be "<".

See: [“Formatting Messages” on page 17](#)

<filter>

<filter-definitions>

</filter>

You can use filters to accept or deny messages based on the following:

- a character string in the message
- a range of message thresholds
- a single message threshold
- a combination of character string, single message threshold, or a range of message thresholds

See: [Chapter 9, “Filters,” on page 121](#)

SAS Appenders for Server Logging

The following appenders can be configured as the value of the <appender> "class" attribute in the XML configuration files for SAS servers:

[ARMAppender](#) (p. 41)

ARMAppender processes all Application Response Measurement (ARM) messages that are submitted by an external ARM agent or by the SAS ARM agent.

[ConsoleAppender](#) (p. 45)

ConsoleAppender writes messages to the UNIX and Windows operating system consoles.

[DBAppender](#) (p. 46)

DBAppender writes log events to a SAS table or to a table in a third-party DBMS.

[FileAppender](#) (p. 52)

FileAppender writes messages to the specified file in the specified path.

[FilteringAppender](#) (p. 55)

FilteringAppender applies specified filters to determine whether events should be passed to a referenced appender. You can specify a layout to be applied to the events before they are passed.

[IOMServerAppender](#) (p. 58)

IOMServerAppender commits messages from any IOM server to a volatile run-time cache.

[JavaAppender](#) (p. 60)

JavaAppender sends messages to a custom Java class.

[JMSAppender](#) (p. 67)

JMSAppender sends messages to a message queue by using the Java Message Server (JMS) interface.

[RollingFileAppender](#) (p. 72)

RollingFileAppender writes messages to the specified file in the specified path, and begins writing messages to a new file that has a different name when specified criteria are met.

[sLogAppender](#) (p. 80)

sLogAppender is a reserved appender. You should not define new instances of this appender.

[UNXFacilityAppender](#) (p. 80)

UNXFacilityAppender writes messages to the syslogd logging facility in UNIX operating systems.

[WindowsEventAppender](#) (p. 82)

WindowsEventAppender writes messages to the Windows Event log.

[ZOSFacilityAppender](#) (p. 83)

ZOSFacilityAppender enables multiple instances of SAS in the z/OS operating system to write messages to a common location.

[ZOSWtoAppender](#) (p. 86)

ZOSWtoAppender directs SAS application messages to the z/OS operating system console.

Appenders in the SAS Language

When you specify an appender reference in a logger language element, you can use any of the appenders that are defined for SAS server logging and the appender FileRefAppender.

FileRefAppender is an appender that you create only in the SAS language, and only by using a SAS function, DATA step object, or autocall macro. As the name indicates, FileRefAppender names a fileref that defines a location to store messages. FileRefAppender is the only appender that can be created by using the SAS language.

When you create an appender in a DATA step, the appender is available only for the duration of the DATA step. After the DATA step has run, the appender is no longer available.

For more information, see [“Creating and Using Appenders in a SAS Program”](#) on page 135.

Referencing Appenders in a Logger

After an appender is defined, it can be referenced by a logger. To reference an appender in a logging configuration file, you include the appender name in the logger's <appender-ref> element. In the following logger and appender definitions, the appender WinEvtVwr is referenced by the logger WEVLogger:

```
<appender class="WindowsEventAppender" name="WinEvtVwr">
  <param name="Appname" value="myApp"/>
</appender>

<logger name="WEVLogger">
  <level="error"/>
  <appender-ref ref="WinEvtVwr"/>
</logger>
```

To reference an appender in a logger language element, you specify the appender name as the value of the APPENDER-REF argument:

```
%log4sas_logger(myLogger, appender-ref=(myAppender), level=error);

rc= log4sas_logger("myLogger" "appender-ref=(myAppender) level=error";

declare logger logobj("myLogger");
logobj.appenderref="myAppender";
```

To write the same message in multiple logs, you can specify multiple appender references in a configuration file logger definition:

```
<logger name="MyLoggers">
  <level="error"/>
  <appender-ref ref="WinEvtVwr"/>
  <appender-ref ref="RollingFileAppender"/>
</logger>
```

In a SAS program, you can add multiple appender names separated by a space in the APPENDER-REF argument:

```
%log4sas_logger(myLogger, appender-ref=(myAppender myRollingFile), level=error);
```

Logging Thresholds

The SAS logging facility provides six thresholds: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. Thresholds are used to ignore log events that are lower than a particular level, or to filter messages so that only a single message level is logged.

When a log event occurs, up to three levels of filtering can take place:

1. filtering log events by comparing the log event level to the log event's logger level
2. filtering log events by comparing the log event level to the appender's threshold
3. filtering log events by comparing the log event level to the threshold that is specified in the filter definition, which is a part of the appender configuration

In the first two cases, if the log event level is lower than the logger or appender threshold, the logging facility ignores the log event. Otherwise, processing of the log event continues.

In the third case, the log event level is compared to the filter threshold. If there is a match, the log event can be either accepted or denied. If there is no match, the filtering process continues to the next filter in the filtering policy. For more information, see [Chapter 9, “Filters,” on page 121](#).

The logging levels, from the lowest to the highest, are as follows:

TRACE

produces the most detailed information about your application. This level is primarily used by SAS Technical Support or development.

DEBUG

produces detailed information that you use to debug your application. This level is primarily used by SAS Technical Support or development.

INFO

provides information that highlights the progress of an application.

WARN

provides messages that identify potentially harmful situations.

ERROR

provides messages that indicate that errors have occurred. The application might continue to run.

FATAL

provides messages that indicate that severe errors have occurred. These errors will probably cause the application to end.

Requirement: The level must be enclosed in quotation marks.

An appender can be configured to have a threshold. By default, however, appenders do not have a threshold. When set, all log events that have a level lower than the threshold are ignored by the appender.

Formatting Messages

The format of a message can be customized by specifying a unique pattern layout for each appender class in the SAS logging facility. To create a pattern layout for an appender class, you use conversion characters that represent the types of data to include in the message. You can also control the sequence of the data and the alignment of the data in columns in the message.

Note: Conversion patterns that are used in the SAS logging facility are similar to the conversion patterns that are used in the C language PRINTF statement.

Note: The ARMAppender classes use a set of pattern layouts that are only for the ARM subsystem. For more information, see Chapter 12, “ARM Appender Pattern Layouts for ARM Messages,” in *SAS Interface to Application Response Measurement (ARM): Reference*.

Here is an excerpt of an XML file that contains a pattern layout:

```
<layout>
  <param name="ConversionPattern" value="%d; %-5p; %t; %c; %m"/>
</layout>
```

Each data item to be included in the message is represented by a conversion character. Also, literal text and alignment commands can be specified to enhance the message format. In this example, the data items are the date, the logging level, the thread, the logger, and the message.

Here is an example of a message:

```
2008-06-25-10:24:22,234; WARN; 3; Appender.IOMCallContext; (yn14.sas.c:149);
  Numeric maximum was larger than 8, am setting to 8.
```

For more information, see [Chapter 8, “Pattern Layouts,”](#) on page 95.

Message Filtering

In addition to filtering log events based on thresholds that are assigned to loggers or appender definitions, the logging facility enables you to use filter classes to filter log events based on the following:

- a character string in the message
- a single threshold
- a range of thresholds
- a combination of strings and thresholds

Here is a list of the filter classes:

Filter Class Name	Description
RepeatMatchFilter	filters repeated log messages

Filter Class Name	Description
StringMatchFilter	filters messages based on a character string in the message.
LevelRangeFilter	filters messages based on a range of thresholds.
LevelMatchFilter	filters messages based on a single threshold.
AndFilter	filters messages based on the results of a list of other filters.
DenyAllFilter	denies log events that did not meet the criteria of previous filters in a filter policy.

You can define one or more filters within the <appender> definition in the logging configuration file. Filters are not available in the logging facility language elements for SAS programs.

Filters are processed in the order in which they appear in the <appender> definition, creating a filtering policy for the appender. The filters either accept the filtering criteria and process the log event, deny the filtering criteria and deny the log event, or accept the filtering criteria, and the filtering process checks the next filter in the filtering policy. If the log event has not been denied, and if there are no other filters in the filtering policy, the appender accepts and processes the log event.

For more information, see [Chapter 9, “Filters,”](#) on page 121.

Using the SAS Logging Facility in the SAS Intelligence Platform

About the Initial Logging Configuration for SAS Servers

When you install the SAS Intelligence Platform, the installation process performs the following configuration steps:

- It enables logging for each server by specifying the LOGCONFIGLOC= option in the server's configuration file.
- For each server, it provides a logging configuration file called **logconfig.xml** that is located in the server's configuration directory.
- For each server, it provides the following alternative logging configuration files:
 - **logconfig_trace.xml**, which can be used for troubleshooting.
 - **logconfig_apm.xml**, which can be used with the SAS Enterprise Business Intelligence Audit and Performance Measurement package. This package is available for download from <http://support.sas.com/rnd/emi>.

For more information, see the following topics in the *SAS Intelligence Platform: System Administration Guide*:

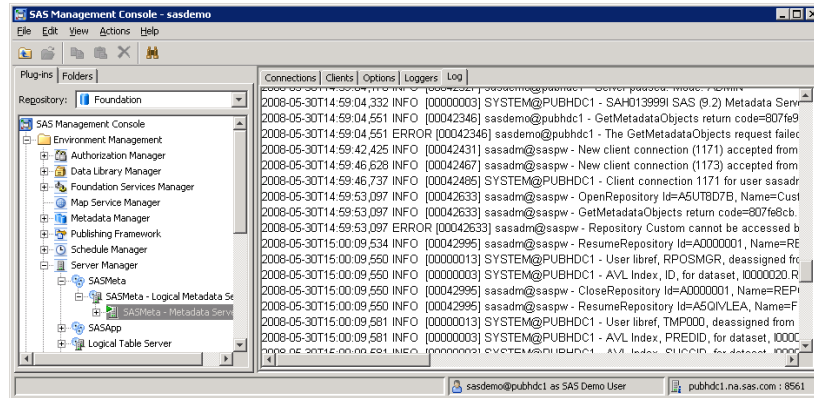
- “Initial Logging Configuration for SAS Servers”

- “Default Locations for Server Logs”

Viewing SAS Logging Messages and Adjusting Logging Levels in Client Applications

The initial logging configurations for some SAS servers include appender definitions that make logging messages available in the following client applications:

- SAS Management Console. From this application, you can view logging messages for metadata servers, object spawners, OLAP servers, pooled workspace servers, and stored process servers. In this example, the metadata server log is displayed:



You can also use the Loggers tab in SAS Management Console to dynamically adjust server logging levels, without the need to restart the server. The following example shows the Loggers tab for the metadata server:

Name ▲	Description	Level
<Root>		Error
Admin		Information
Admin.Operations		<Inherited>
App		Information
App.FilteredList		<Inherited>
App.FilteredList.FilteredList		<Inherited>
App.FilteredList.FilteredList.Close		<Inherited>
App.FilteredList.FilteredList.Filter		<Inherited>
App.FilteredList.FilteredList.Filter.Get		<Inherited>
App.FilteredList.FilteredList.Filter.Set		<Inherited>
App.FilteredList.FilteredList.GetAttrib...		<Inherited>

For more information, see “Using SAS Management Console to Monitor SAS Servers” in the *SAS Intelligence Platform: System Administration Guide*.

- SAS Data Integration Studio. From this application, you can view performance-related events that are associated with a SAS Data Integration Studio job. For more information, see the product Help.

You can also use enterprise systems management products to view server logging messages and dynamically adjust server logging levels. For more information, see the Enterprise Management Integration Web page at <http://support.sas.com/rnd/emi>.

Best Practices for SAS Server Logging

When using the logging facility for SAS servers, follow these best practices:

- Use the initial logging configuration files that are created during installation. These files provide a good starting point for server logging.
- If you need to change a server's logging configuration, back up the initial configuration file before making changes. Make configuration changes incrementally, and evaluate the effect of each change before making additional changes.
- Do not use the TRACE and DEBUG levels unless you are directed to do so by SAS Technical Support, since these logging levels can affect performance. You can use either of these methods to adjust logging levels for SAS Technical Support:
 - Enable the `logconfig_trace.xml` file that is provided for the server.
 - Use the server manager features of SAS Management Console to adjust levels temporarily and avoid having to restart the servers.

For more information, see the following documents:

- “Administering Logging for SAS Servers” in the *SAS Intelligence Platform: System Administration Guide* on <http://support.sas.com>.
- *SAS Interface to Application Response Measurement (ARM): Reference* describes SAS features that are compliant with the ARM 2.0 and ARM 4.0 standards and that enable you to monitor the performance of SAS applications.

Chapter 2

Enabling the SAS Logging Facility

Enabling the Logging Facility for SAS Server Logging	21
Enabling the Logging Facility in SAS Programs	21
Naming a SAS Session	22

Enabling the Logging Facility for SAS Server Logging

You enable the SAS logging facility for SAS servers, by specifying the LOGCONFIGLOC= system option when SAS starts. The LOGCONFIGLOC= system option names the location of the logging configuration file.

You can add the LOGCONFIGLOC= system option to either the SAS configuration file or to the SAS command that you use to start SAS. If you perform a planned deployment, then the SAS Deployment Wizard includes this system option in the configuration for your SAS servers.

This system option can be set only when SAS starts and not during a SAS session. For more information, see [“LOGCONFIGLOC= System Option” on page 24](#).

Enabling the Logging Facility in SAS Programs

The logging facility is enabled for SAS programs at all times. That is, it is not necessary to specify the LOGCONFIGLOC= system option in order for SAS programs to use the logging facility.

If you use the logging facility autocall macros, the MAUTOSOURCE system option must be set and the %LOG4SAS autocall macro must be invoked before any other logging facility autocall macros are invoked. The MAUTOSOURCE system option is set by default. No further action is required unless this option is turned off.

The logging facility functions and DATA step objects have no initialization requirements.

Naming a SAS Session

Logging facility messages can be set to include the name of the SAS session, which might help you to read the log and to diagnose problems.

To identify a SAS session by name, you specify a session name as the value of the LOGAPPLNAME= system option.

To display the value of the LOGAPPLNAME= system option in a message, you must include the S conversion character in the conversion pattern layout, using the key App.Name.

For more information, see [“LOGAPPLNAME= System Option” on page 23](#) and the [“S Conversion Character” on page 113](#).

Chapter 3

System Options for Enabling the SAS Logging Facility

Dictionary	23
LOGAPPLNAME= System Option	23
LOGCONFIGLOC= System Option	24

Dictionary

LOGAPPLNAME= System Option

Specifies a SAS session name for SAS logging.

Valid in:	configuration file, SAS invocation
Category:	Log and procedure output control: SAS log
PROC OPTIONS GROUP=	LOGCONTROL

Syntax

LOGAPPLNAME=*name*

Syntax Description

name

specifies a name for the SAS session. If the name contains a space, enclose the name in either single or double quotation marks.

Details

The name that is specified by the LOGAPPLNAME= system option is used to identify the name of a SAS session in logging facility logs if the S conversion character is specified in the pattern layout.

You can use the &SYSLOGAPPLNAME automatic macro variable to obtain the name of the SAS session in a SAS program.

See Also

- [“S Conversion Character” on page 113](#)

Automatic Macro Variables:

- “SYSLOGAPPLNAME Automatic Macro Variable” in *SAS Macro Language: Reference*

LOGCONFIGLOC= System Option

Specifies the name of the XML configuration file or a basic logging configuration that is used to initialize the SAS logging facility.

Valid in:	configuration file, SAS invocation, spawner invocation
Category:	Environment control: Initialization and operation
PROC OPTIONS GROUP=	EXECMODES
Alias:	LOGCFGLOC=

Syntax**Production Syntax**

LOGCONFIGLOC=*file-specification*

Basic Logging Configuration Syntax

LOGCONFIGLOC=*basic:level, FileAppender, path-and-filename-pattern*

LOGCONFIGLOC=*basic:level, ConsoleAppender*

LOGCONFIGLOC=*basic:level*

Syntax Description*file-specification*

specifies the physical name of the XML configuration file that is used to initialize the SAS logging facility. The physical name is the name that is recognized by your operating system. Enclose the physical name in single or double quotation marks if the name contains spaces.

basic:level,FileAppender,path-and-filename-pattern

specifies to write messages to the specified file. **basic**: indicates to use the root logger. **FileAppender** indicates to use the FileAppender class.

Provide values for these arguments:

level

specifies the logging threshold. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL.

See: “Logging Thresholds” on page 16

path-and-filename-pattern

specifies the path to which the log file is written and the conversion pattern that is used to create the log filename. The conversion pattern can include the %d and %S{key} conversion characters.

See:

“d Conversion Character” on page 108

“S Conversion Character” on page 113

Note: The appender that SAS creates uses DEFAULTHEADER as the HeaderPattern parameter and DEFAULT for the ConversionPattern parameter. For more information, see [“Conversion Patterns Supplied by SAS” on page 96](#).

See:

[“FileAppender” on page 52](#)

[“About Basic Logging Configurations” on page 25](#)

basic:level,ConsoleAppender

specifies to write messages to the current console. **basic:** indicates the root logger. **ConsoleAppender** indicates to use the ConsoleAppender class.

Provide a value for this argument:

level

specifies the logging threshold. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL.

See: [“Logging Thresholds” on page 16](#)

Note: This configuration uses the conversion pattern DEFAULTHEADER for the HeaderPattern parameter and the conversion pattern DEFAULT for the ConversionPattern parameter.

See:

[“ConsoleAppender” on page 45](#)

[“About Basic Logging Configurations” on page 25](#)

basic:level

specifies to use the root logger. When the root logger is specified without an appender, messages are written to the traditional SAS log.

Provide a value for this argument:

level

specifies the logging threshold. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL.

See: [“Logging Thresholds” on page 16](#)

See:

[“Logger” on page 37](#)

[“About Basic Logging Configurations” on page 25](#)

Details

The Basics

If the LOGCONFIGLOC= system option is specified when SAS starts, and if the SYSIN= option or the OBJECTSERVER option is also specified, logging is performed only by the logging facility; the SAS log is not started and the LOGPARM= system option is ignored. The LOG= system option is applied only when the %S{App.Log} conversion character is specified in the logging configuration file.

About Basic Logging Configurations

SAS provides three basic logging configurations for you to experiment with the logging facility:

- basic:level,FileAppender,*path-and-filename-pattern*
- basic:level,ConsoleAppender
- basic:level

By using one of the basic logging configurations, you can experiment with the logging facility without creating a configuration file. SAS creates a configuration file for you based on the basic logging configuration that you specify in the LOGCONFIGLOC= system option. The basic logging configurations enable messages to be written to a file, the current console, and the traditional SAS log.

Example

The following example shows the use of the LOGCONFIGLOC= system option for a user-specified logging configuration:

```
sas -logconfigloc metaserverlog.xml
```

The following example shows the use of the LOGCONFIGLOC= system option for a basic logging configuration:

```
sas -logconfigloc basic:INFO,FileAppender,c:\mylog\logFacility.%S{hostname}.log
```

Part 2

XML Configuration Overview and Reference

<i>Chapter 4</i>	
Overview of the Logging Configuration File	29
<i>Chapter 5</i>	
Logger Reference	37
<i>Chapter 6</i>	
Appender Reference	41
<i>Chapter 7</i>	
Database Connection Options Reference for DBAppender	89
<i>Chapter 8</i>	
Pattern Layouts	95
<i>Chapter 9</i>	
Filters	121

Chapter 4

Overview of the Logging Configuration File

Typographical Conventions	29
Syntax Conventions	29
XML Elements for Configuring SAS Logging	30
Structure of the Logging Configuration File	33
Sample Configuration Files	34
Modifying or Replacing the Logging Configuration File	34
Error Messages for Logging Configuration File Errors	35

Typographical Conventions

Type styles have special meaning for some components of XML syntax in the logging configuration file. The following list explains the style conventions for the syntax:

italics

identifies arguments or values that you supply. Items in italics can represent user-supplied values that are either one of the following:

- nonliteral values that are assigned to an argument (for example, value="*column-parameter*")
- nonliteral arguments (for example, *<filter-definitions>*)

case sensitivity

All text is case sensitive in the logging configuration file. Type element and attribute names, as well as literal values, as they are shown in the syntax.

Syntax Conventions

In traditional SAS syntax, angle brackets (<>) are used to denote optional syntax. In the logging configuration file syntax, square brackets ([]) are used to denote optional syntax. The logging configuration file syntax uses the following symbols:

<>

The left angle bracket begins an XML element. The right angle bracket ends an XML element.

- /
A slash before an element name ends the element definition.
- />
A slash followed by a right angle bracket ends the definition for the <param>, <level>, and <appender-ref> subelements.
- []
Square brackets identify optional elements or arguments. Any XML element or attribute that is not enclosed in square brackets is required.
- |
A vertical bar indicates that you can choose one value from a group of values. Values separated by bars are mutually exclusive.
- 1-n...
For repeated elements, the -1 after an element indicates the first element. The ellipsis before an element and the -n after the same element indicates that the element can be repeated any number of times.
- " "
Double quotation marks identify an attribute value.

XML Elements for Configuring SAS Logging

You use <?xml?> and <logging:configuration> elements as the first XML elements in the configuration file to specify XML attributes and the SAS logging message category.

The <appender>, <logger>, and <root> elements define the loggers and appenders.

The <param>, <layout>, <level>, <filter>, and <appender-ref> elements are child elements that customize appender and logger definitions.

The following table summarizes the XML DTD for SAS logging:

Table 4.1 SAS Logging XML Configuration Elements

Element Name and Description	Element Characteristics
<?xml ?> is the first XML element in the configuration file	<ul style="list-style-type: none"> • Number of instances: one • Required attributes: <ul style="list-style-type: none"> • version="1.0" • Optional attributes: <ul style="list-style-type: none"> • encoding= "<i>encoding-specification</i> " specifies a language encoding. For more information, see the <i>SAS National Language Support (NLS): Reference Guide</i>.

Element Name and Description	Element Characteristics
<p><logging:configuration> defines the logging message category and the message category default values </logging:configuration></p>	<ul style="list-style-type: none"> • Number of instances: one • Required attributes: <ul style="list-style-type: none"> • xmlns:logging="http://support.sas.com/xml/logging/1.0/" • Optional attributes: <ul style="list-style-type: none"> • threshold="<i>level</i>" specifies the default message threshold for filtering log messages; the default is null, two double quotation marks with no space between them (""). For a list of levels, see “Logging Thresholds” on page 16. • debug="TRUE FALSE" specifies whether to run in debug mode; the default is "FALSE". Setting a value of TRUE is the same as setting the level to a value of DEBUG for the logger that is specified in the log event. • modify="TRUE FALSE" specifies whether to modify the configuration file; the default is "FALSE". Set modify="TRUE" to add a new appender, connect a logger to a new or existing appender, remove an appender, or change a logger level. Set modify="FALSE" if you are not modifying the logging configuration file or if you are replacing the configuration file. <p>You can use the IOMOPERATE procedure to modify the configuration file. For more information, see “IOMOPERATE Procedure” in <i>SAS Intelligence Platform: Application Server Administration Guide</i> on http://support.sas.com.</p> • Optional child elements: number of instances <ul style="list-style-type: none"> • <appender>: zero or more • <logger>: zero or more • <root>: zero or one
<p><appender> defines a destination for log messages, filters to limit log messages, and the format of the log message </appender></p>	<ul style="list-style-type: none"> • Number of instances: zero or multiple • Required attributes: <ul style="list-style-type: none"> • name="<i>appender-name</i>" is a user-defined name for the appender. • class="<i>class-name</i>" specifies the type of appender. For more information, see “SAS Appenders for Server Logging” on page 14. • remove="TRUE FALSE" specifies whether to remove the appender from the logging configuration file; the default is "FALSE". A value of "TRUE" removes the appender from the logging configuration file. A value of "FALSE" does not remove the appender from the logging configuration file. <p>You can use the IOMOPERATE procedure to modify the configuration file. For more information, see “IOMOPERATE Procedure” in <i>SAS Intelligence Platform: Application Server Administration Guide</i> on http://support.sas.com.</p> • Optional child elements: number of instances <ul style="list-style-type: none"> • <param>: zero or multiple • <layout>: zero or one • <filter>: zero or multiple • <appender-ref>: zero or multiple • <rollingPolicy>: zero or one; required when the appender class is RollingFileAppender. Otherwise, do not include this element.

Element Name and Description	Element Characteristics
<p><logger></p> <p>names a message category, references an appender, and defines a message threshold</p> <p></logger></p>	<ul style="list-style-type: none"> • Number of instances: zero or multiple • Required attribute: <ul style="list-style-type: none"> • name="logger-name" is the name of the message category. • Optional attribute: <ul style="list-style-type: none"> • additivity="TRUE FALSE" specifies whether to pass the log event to loggers in a hierarchy. • immutability="TRUE FALSE" specifies whether settings for additivity and level can be changed by using the SAS language. • Optional child elements: number of instances <ul style="list-style-type: none"> • <level>: zero or multiple • <appender-ref>: zero or multiple
<p><root></p> <p>a fixed SAS logger and the highest logger in any logging hierarchy</p> <p></root></p>	<ul style="list-style-type: none"> • Number of instances: one <ul style="list-style-type: none"> <i>Note:</i> A root logger is not required for a logging configuration file that is used to modify a logging configuration. • Attributes: none • optional child elements: number of instances <ul style="list-style-type: none"> • <level>: zero or one • <appender-ref>: zero or multiple
<p><param /></p> <p>defines a logger, appender, or a filter parameter and a parameter value</p>	<ul style="list-style-type: none"> • Number of instances: zero or multiple • Required attributes: <ul style="list-style-type: none"> • name="attribute-name" specifies the literal name of an attribute. • value="attribute-value" specifies either a literal or user-supplied value.
<p><layout></p> <p>specifies conversion-pattern layouts</p> <p></layout></p>	<ul style="list-style-type: none"> • Number of instances: zero or one • Optional child element: number of instances <ul style="list-style-type: none"> • <param>: zero or multiple
<p><level></p> <p>defines the message threshold that a logger accepts</p>	<ul style="list-style-type: none"> • Number of instances: zero or one • Required attribute: <ul style="list-style-type: none"> • value="level" specifies a message threshold for filtering log events; by default, a level is inherited from the parent logger. For more information, see “Logging Thresholds” on page 16.
<p><filter></p> <p>specifies message-filtering values</p> <p></filter></p>	<ul style="list-style-type: none"> • Number of instances: zero or one • Required attribute: <ul style="list-style-type: none"> • class="filter-class" specifies the name of the filter. For more information, see “Message Filtering” on page 17. • Optional child element: instances <ul style="list-style-type: none"> • <param>: zero or multiple

Element Name and Description	Element Characteristics
<appender-ref> names an appender for processing a log event </appender-ref>	<ul style="list-style-type: none"> • Number of instances: zero or multiple • Required attribute: <ul style="list-style-type: none"> • <code>ref="appender-name"</code> specifies a user-defined appender name

Structure of the Logging Configuration File

The layout of a logging facility XML configuration file must contain, at minimum, the <?xml?> element, the <logging> element, and a <root> logger. The first two elements, in order, are these elements:

1. <?xml>
2. <logging:configuration>

After the <logging:configuration> element, these elements can appear in any order:

- <appender>
- <logger>
- <root>

See “XML Elements for Configuring SAS Logging” on page 30 for information about the number of instances for each element.

Here is an example configuration file that shows the structure of the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
  <logging:configuration xmlns:logging="http://support.sas.com/xml/logging/1.0/">
    <appender class="RollingFileAppender" name="TimeBasedRollingFile">
      <param name="Append" value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <param name="Unique" value="true"/>
      <filter class="StringMatchFilter">
        <param name="LevelToMatch" value="error"/>
        <param name="AcceptOnMatch" value="true"/>
      </filter >
      <rollingPolicy class="TimeBasedRollingPolicy">
        <param name="FileNamePattern" value="c:\sas\logs\server\workspace_%d.log"/>
      </rollingPolicy>
      <layout>
        <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
      </layout>
    </appender>
    <appender class="FileAppender" name="rootAppender">
      <param name="Append" value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <param name="File" value="c:\logs\root\root1.log"/>
    </appender>
    <logger name="log4WServer">
      <level value="info"/>
    </logger>
  </logging:configuration>
```

```

        <appender-ref ref="TimeBasedRollingFile"/>
    </logger>
</root>
    <level value="error">
        <appender-ref ref="rootAppender"/>
    </root>
</logging:configuration>

```

Sample Configuration Files

SAS supplies sample logging facility configuration files in SAS Help and Documentation. To access the sample 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 down to the logging facility configuration examples.

Modifying or Replacing the Logging Configuration File

You can use the SET LOG CFG command of the IOMOPERATE procedure to replace or modify the logging configuration.

You can modify the logging configuration to do the following:

- configure a logger with a new or existing appender
- change a logger level
- add a new appender
- remove an appender

Note: SAS ignores the IMMUTABILITY settings of a logger when an administrator modifies a configuration file using the IOMOPERATE procedure or SAS Management Console.

To modify the logging configuration, you create a logging configuration file only with the changes that you would like to make. A complete configuration file is not required. The configuration file must contain the `<?xml>`, `<logging:configuration>`, and `</logging:configuration>` elements. Any time you modify the logging configuration, the `<logging:configuration>` element must have the **modify** attribute set to "TRUE". If you are removing an appender, the **remove** attribute on the appender element must be set to "TRUE". Here is an example of a logging configuration file that is used to associate an existing appender with a logger:

```

<?xml version="1.0"?>
<logging:configuration
  xmlns:logging="http://www.sas.com/xml/logging/1.0/" modify="true">
  <logger name="App.Program">
    <appender-ref ref="newcons"/>

```

```
</logger>  
</logging:configuration>
```

To replace the logging configuration, you create a new logging configuration file. The `<logging:configuration>` element must have the `modify` attribute set to "FALSE" or the `modify` attribute must not be present in the `<logging:configuration>` element.

When you modify or replace the logging configuration, the IOMOPERATE procedure might look similar to this:

```
proc iomoperate uri="iom://localhost:15976;Bridge;TRUSTEDSAS" type=METADATA;  
  set log cfg="newcfg.xml";  
quit;
```

For more information, see "IOMOPERATE Procedure" in *SAS Intelligence Platform: Application Server Administration Guide* on <http://support.sas.com>.

Error Messages for Logging Configuration File Errors

If an error occurs in a logging configuration file, SAS issues one of the following error messages:

- Failed to create Appender.
- Failed to configure Appender.
- Failed to add Appender
- Failed to configure Logger.
- Failed to get Logger.
- Failed to allocate a buffer for the configuration path.

Chapter 5

Logger Reference

Dictionary	37
Logger	37

Dictionary

Logger

A logger names a specific message category and associates the message category with a message level and one or more appenders that process the log message.

See: [“Loggers” on page 7](#)

Syntax

XML Configuration

```
<logger name="logger-name" additivity="TRUE | FALSE"
    immutability="TRUE | FALSE">
    <level value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"/>
    <appender-ref ref="appender-name"/>
</logger>
```

Syntax Description

name="logger-name"

specifies the name of a message category. The value of *logger-name* is case sensitive and can be a single name or a hierarchical name. Use a period to separate hierarchical names. Quotation marks are required.

Default: None

Requirement: Yes

additivity="TRUE | FALSE"

specifies whether to pass the log event to loggers in the hierarchy:

TRUE

specifies to pass the log event to loggers in the hierarchy.

FALSE

specifies not to pass the log event to loggers in the hierarchy.

Default: TRUE

Requirement: No

immutability="TRUE | FALSE"

specifies whether the logger's additivity and level settings are permanent or whether they can be changed by using the SAS language. Only level and additivity changes can be made using the SAS language.

TRUE

specifies that no changes can be made to the logger's additivity and level settings..

FALSE

logger level and additivity setting can be changed using the SAS language.

Default: FALSE

Requirement: No

Interaction: IMMUTABILITY is ignored for configuration changes made by administrators using SAS Management Console or the IOMOPERATE procedure.

See:

["%LOG4SAS_LOGGER Autocall Macro" on page 147](#)

["LOG4SAS_LOGGER Function" on page 161](#)

["DECLARE Statement, Logger Object" on page 172](#)

level value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"

specifies the lowest event level that is processed by this logger. Log events that have messages that are below the specified level are ignored. The valid level values are listed here from lowest to highest. If a level is not specified, SAS uses the level of the next highest parent logger that defines a level. Quotation marks are required.

Default: None

Requirement: No

See: ["Logging Thresholds" on page 16](#)

appender-ref ref="appender-name"

specifies the name of an appender whose destination receives messages for log events that are specified for this logger. The value of *appender-name* must be defined in the XML configuration file. You can define multiple appenders for a logger.

Default: None

Requirement: No

Details

The definition of an appender can appear anywhere in a logging configuration file. In a SAS program, an appender that is specified as an appender reference in a logger must be defined before the logger is defined.

Examples

Example 1: Example 1: Define a Logger to Log Error Messages for an Application in Production

This example creates a logger to record error log events for an application that is in production. The appender `ApplProduction_Appender` must also be defined in the XML configuration file.

```
<logger name="ApplProduction_Logger">
  <level value="error"/>
  <appender-ref ref="ApplProduction_Appender"/>
</logger>
```

Example 2: Example 2: Define Loggers That Inherit the Level

In this configuration example, `IOMSRv` is the parent logger for `IOMSRv.Workspace` and `IOMSRv.Metadata`. The `IOMSRv.Workspace` logger and the `IOMSRv.Metadata` logger do not define a level. Therefore, they inherit the level of the closest parent, which is `IOMSRv`. `IOMSRv` defines a level of error. Log events for the `IOMSRv.Workspace` and the `IOMSRv.Metadata` message categories use the level of error and write error and fatal messages to their respective appender destinations.

```
<logger name="IOMSRv">
  <level value="error"/>
</logger>

<logger name="IOMSRv.Workspace">
  <appender-ref ref="WorkspaceLog"/>
</logger>

<logger name="IOMSRv.Metadata">
  <appender-ref ref="MetadataLog"/>
</logger>
```


Chapter 6

Appender Reference

Dictionary	41
ARMAppender	41
ConsoleAppender	45
DBAppender	46
FileAppender	52
FilteringAppender	55
IOMServerAppender	58
JavaAppender	60
JMSAppender	67
RollingFileAppender	72
sLogAppender	80
UNXFacilityAppender	80
WindowsEventAppender	82
ZOSFacilityAppender	83
ZOSWtoAppender	86

Dictionary

ARMAppender

Logs performance data based on ARM 2.0 and ARM 4.0 standards. It supports default transaction correlation and converts ARM transaction events that were created before SAS 9.2 into SAS log events.

Valid in: XML Configuration

Note: ARMAppender syntax is case sensitive.

See: This document covers only the syntax of ARMAppender. For information about using ARM in SAS, including details about using ARMAppender, see *SAS Interface to Application Response Measurement (ARM): Reference*

Syntax

```

<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-appender-name">
  <param name="Agent" value="ARM-agent-library"/>
  <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>

```

Syntax Description

appender class="ARMAppender" name="ARM-appender-name"
 specifies an appender name for the ARM appender.

Default: None

Restriction: Only one instance of an ARMAppender can exist per process.

name="Agent" value="ARM-agent-library"

specifies the name of the library that contains the external ARM 4.0 agent library that receives the events, in addition to the referenced appenders. See your vendor documentation for the correct library name. Two values 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".

Note: If you intend to have a null value for the Agent parameter, you do not need to specify the parameter.

value="library-name"

specifies the name of the library that contains the external ARM 4.0 agent library that receives the events.

Notes:

Output is always sent to all referenced appenders as well as to the external agent, if specified, and to the ARM log.

This parameter is not required.

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

Note: This parameter is not required.

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.

Note: This parameter is not required.

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 is set for the host, or UTF-8 as required by the ARM 4.0 standards.

Note: This parameter is not required.

name="File" value="path-and-filename"

specifies the path and filename of the file to which ARM messages are written.

Default: None

Note: This parameter is required.

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

Note: This parameter is not required.

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

Note: This parameter is not required.

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: The current user ID if available, otherwise NULL

Note: This parameter is not required.

Details

ARMAppender is configured and customized for accessing performance data. The primary role of ARMAppender is to record ARM transaction events, process the events, and route the events to an appropriate output destination. These events, when processed by ARMAppender, are formatted in the appropriate ARM 4.0 format, using the fixed portion of the message and the values that were recorded in the diagnostic context.

The existing ARM 2.0 implementations are changed to logger requests that contain the appropriate performance event attribute settings.

For more information about ARM and ARMAppender, see *SAS Interface to Application Response Measurement (ARM): Reference*

Example

The following example is a SAS logging facility configuration file that includes ARMAppender. In the appender FileAppender class definition, the conversion pattern is separated by the symbol |. The log message contains any character that you enter between conversion patterns.

```
<?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"/>
  </appender>
</logging:configuration>
```

```

    <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>

```

ConsoleAppender

Writes messages to the UNIX and Windows operating system consoles.

Valid in: XML configuration

Syntax

```

<appender class="ConsoleAppender" name="appender-name">
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>

```

Syntax Description

<appender class="ConsoleAppender" name="appender-name">
 specifies the user-assigned name for this instance of ConsoleAppender.

Default: None

Requirement: The appender class and name are required to specify a console appender.

<param name="ConversionPattern" value="conversion-pattern"/>
 specifies a series of conversion characters that represent the types of data to include in the log. For example, use the conversion characters %d %t %m to include the date, the thread, and the message, respectively, in the log.

See: [Chapter 8, "Pattern Layouts,"](#) on page 95

Details

ConsoleAppender is a logging facility appender that supports event logging on UNIX and Windows operating systems. ConsoleAppender writes messages to the current console.

Example

The following example is a typical XML configuration file that specifies ConsoleAppender.

```
<?xml version="1.0" encoding="UTF-8"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">
  <appender class="ConsoleAppender" name="console">
    <layout>
      <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
    </layout>
  </appender>
  <!-- Root logger -->
  <root>
    <level value="Error"/>
    <appender-ref ref="console"/>
  </root>
</logging:configuration>
```

DBAppender

Writes log events to a SAS table or to a table in a third-party DBMS.

Valid in: XML Configuration

Syntax

```
<appender class="DBAppender" name="appender-name">
  <param name="ConnectionString" value="connection-string"/>
  <param name="Locale" value="locale"/>
  <param name="MaxBufferedEvents" value="integer"/>
  <param name="SelectStatement" value="SQL-statement"/>
  <param name="TableName" value="table-name"/>
  <param name="Column" value="column-parameter-1"/>
  [<param name="Column" value="column-parameter-n"/>]
</appender>
```

Syntax Description

class="DBAppender" name="appender-name"

specifies the user-assigned name for this instance of DBAppender.

Default: None

Requirement: This parameter is required.

name="ConnectionString" value="connection-string"

specifies options for connecting to the data store to which events are to be written. Within the connection string, place a semicolon between each option. The options vary depending on the type of data store you are using. For details, see:

- “DBAppender Connection Options for DB2” on page 89
- “DBAppender Connection Options for ODBC” on page 90
- “DBAppender Connection Options for Oracle” on page 92
- “DBAppender Connection Options for SAS Tables” on page 93
- “DBAppender Connection Options for Teradata” on page 93

Requirement: This parameter is required.

name="Locale" value="locale"

specifies the locale that is used to write messages to the file.

Default: The locale setting that is in effect for the SAS session. For example, the LOCALE system option might be specified in the configuration file for a SAS server or in the configuration file for Base SAS.

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the locale that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="MaxBufferedEvents" value="integer"

specifies the maximum number of log events to buffer before sending them to the data store. When MaxBufferedEvents is set to 0, messages are processed synchronously, one row at a time.

Default: 1

Requirement: This parameter is not required.

name="SelectStatement" value="SQL-statement"

specifies an SQL statement that identifies the name of the table, and the names of specific columns within the table, where event information is to be written. The table must already exist. If you want to write information to all of the columns in the table, use TableName instead of SelectStatement.

Requirement: SelectStatement is required if TableName is not specified.

Interaction: The TableName and SelectStatement parameters are mutually exclusive. If both of these parameters are specified, the SelectStatement parameter takes precedence.

Example:

```
<param name="SelectStatement" value="select date,level,message from dblog;"/>
```

name="TableName" value="table-name"

specifies the name of the table to which event information is to be written. The table must already exist. If you want to write to just a subset of the columns in the table, use SelectStatement instead of TableName.

Requirement: TableName is required if SelectStatement is not specified.

Interaction: The TableName and SelectStatement parameters are mutually exclusive. If both of these parameters are specified, the SelectStatement parameter takes precedence.

name="Column" value="column parameter"

specifies a conversion character indicating the type of information that is to be written to a particular column in the table. The following conversion characters are valid: c, d, F, L, m, p, r, S, sn, t, u, x, and X.

To write literal text to a column, specify @ (the at sign) followed by the literal text.

Specify the Column parameters in exactly the same order that the columns occur in the table.

Restrictions:

Limit each column parameter to a single conversion character. Combining multiple conversion characters in a single column could cause alignment problems in the output.

DBAppender does not support the following:

- the n (newline) conversion character
- format modifiers, which control text justification and field widths for data items in a log event

DBAppender has limited support for the d (date/time) conversion character. To insert a date, specify the underlying column to be either a DATE or any character type of length 10. To insert a timestamp, specify the underlying column to be either a TIMESTAMP or any character type of length 24.

If you use the m (message) conversion character, position it in the final column of the table. Because DBAppender does not use format modifiers to control column widths, a lengthy message in an interior column position could cause text overflow problems.

Requirement: At least one Column parameter is required.

See: Conversion character details in the “Pattern Layouts” chapter: [c on page 107](#), [d on page 108](#), [F on page 110](#), [L on page 110](#), [m on page 110](#), [p on page 111](#), [r on page 112](#), [S on page 113](#), [sn on page 115](#), [t on page 115](#), [u on page 115](#), [x on page 116](#), [X on page 116](#)

Example:

```
<param name="column" value="d"/>
  <param name="Column" value="p"/>
  <param name="Column" value="u"/>
  <param name="Column" value="@A literal string value to insert"/>
  <param name="Column" value="m"/>
```

Details

Before using DBAppender, you must first create the table to which the log events are to be written. The Column parameters must be specified in exactly the same order that the corresponding columns occur in the table.

DBAppender processes only a subset of loggers and ignores events for all other loggers. For each ignored event, a new event is created that contains the original event and a message stating that the event was ignored. These new events, which are written to the Logging.Appender.DB logger at the DEBUG level, can be routed to another appender for capture.

DBAppender processes the following subset of loggers:

- Admin and all of its child loggers
- App.Initialize and all of its child loggers
- App.Program and all of its child loggers
- Audit.Data and all of its child loggers

- Audit.Meta and all of its child loggers
- Perf.ARM.*application-name*.APPL
- Perf.ARM.*application-name*.DSIO
- Perf.ARM.*application-name*.PROC
- All child loggers of Perf.ARM.IOM (but not Perf.ARM.IOM itself)
- Perf.ARM.OLAP_SERVER
- Perf.ARM.User and all of its child loggers

For the Perf.ARM loggers, you can use any of the following methods to specify *application-name*:

- specify the LOGAPPLNAME= system option on the command line. See “LOGAPPLNAME= System Option” on page 23.
- specify the AppName parameter in the configuration for the ARMAppender. See “ARMAppender” on page 41.
- specify the value when coding one of the Application Response Measurement (ARM) macros. See the *SAS Interface to Application Response Measurement (ARM): Reference*.

If you do not specify a value for *application-name*, the default value **SAS** is used.

Examples

Example 1: Writing Events to a DB2 Table

In this example, log events are written to a DB2 table called LOG. The table includes the following columns: sequence number, date, logger name, and message.

Step 1: Use the following SQL statement to create the table:

```
create table LOG ( seqno VARCHAR(10), date VARCHAR(24), logger VARCHAR(100),
  msg VARCHAR(500) );
```

Step 2: Use the DB2 command-line processor to create a DSN called LOGDSN.

Step 3: In the logging configuration file, use the following instance of DBAppender to connect to the database and write log events to the table:

```
<appender name="dblog" class="DBAppender">
  <param name="ConnectionString" value="DRIVER=DB2;UID=User1;PWD=*****;
    CONOPTS=(DSN=LOGDSN);CATALOG=X;" />
  <param name="MaxBufferedEvents" value="300" />
  <param name="TableName" value="LOG" />
<!--
The column parameters must be specified in exactly the same order that the
columns occur in the table.
-->
  <param name="Column" value="sn" />
  <param name="Column" value="d" />
  <param name="Column" value="c" />
  <param name="Column" value="m" />
</appender>
```

Example 2: Writing Events to an ODBC-Compliant Database

In this example, log events are written to a table called LOG on a Microsoft SQL Server. The table includes the following columns: sequence number, date, logger name, and message.

Step 1: Use the following SQL statement to create the table:

```
create table LOG ( seqno VARCHAR(10), date VARCHAR(24), logger VARCHAR(100),
  msg VARCHAR(500) );
```

Step 2: Use Windows Data Source Administrator to create an ODBC data source for Microsoft SQL Server. Specify a DSN called SQLSERVERDSN.

Step 3: In the logging configuration file, use the following instance of DBAppender to connect to the server and write log events to the table:

```
<appender name="dblog" class="DBAppender">
  <param name="ConnectionString" value="DRIVER=ODBC;UID=User1;PWD=*****;
    CONOPTS=(DSN=SQLSERVERDSN);" />
  <param name="MaxBufferedEvents" value="300" />
  <param name="TableName" value="LOG" />
<!--
The column parameters must be specified in exactly the same order that the
columns occur in the table.
-->
  <param name="Column" value="sn" />
  <param name="Column" value="d" />
  <param name="Column" value="c" />
  <param name="Column" value="m" />
</appender>
```

Example 3: Writing Events to an Oracle Table

In this example, log events are written to an Oracle table called LOG. The table includes the following columns: sequence number, date, logger name, and message.

Step 1: To create the table, the SAS/ACCESS Engine for Oracle can be used as follows:

```
libname x oracle user=User1 password=***** path='mypath';
data x.LOG;
  length seqno $10;
  length date $24;
  length logger $100;
  length msg $500;
run;
```

Step 2: In the logging configuration file, use the following instance of DBAppender to connect to the Oracle database and write log events to the table:

```
<appender name="dblog" class="DBAppender">
  <param name="ConnectionString" value="DRIVER=oracle;UID=User1;
    PWD=*****;PATH=mypath;CATALOG=oracle_log" />
  <param name="MaxBufferedEvents" value="300" />
  <param name="TableName" value="LOG" />
<!--
The column parameters must be specified in exactly the same order that the
columns occur in the table.
-->
  <param name="Column" value="sn" />
  <param name="Column" value="d" />
  <param name="Column" value="c" />
```

```

    <param name="Column" value="m" />
</appender>

```

Example 4: Writing Events to a SAS Table

In this example, log events are written to a SAS table called LOG. The table includes the following columns: sequence number, date, logger name, and message.

Step 1: Use the following SAS statements to create the table:

```

libname x 'c:\temp';
data x.LOG;
    length seqno $10;
    length date $24;
    length logger $100;
    length msg $500;
run;

```

Step 2: In the logging configuration file, use the following instance of DBAppender to connect to the table and write log events:

```

<appender name="dblog" class="DBAppender">
    <param name="ConnectionString" value="DRIVER=base;CATALOG=base;
        schema=(name=mywork;primarypath='C:\temp');" />
    <param name="MaxBufferedEvents" value="300" />
    <param name="TableName" value="LOG" />
<!--
The column parameters must be specified in exactly the same order that the
columns occur in the table.
-->
    <param name="column" value="sn" />
    <param name="column" value="d" />
    <param name="column" value="c" />
    <param name="column" value="m" />
</appender>

```

Example 5: Writing Events to a Teradata Table

In this example, log events are written to a Teradata table called LOG. The table includes the following columns: sequence number, date, logger name, and message.

Step 1: Use the following SQL statement to create the table:

```

create table LOG ( seqno VARCHAR(10), date VARCHAR(24), logger VARCHAR(100),
    msg VARCHAR(500) );

```

Step 2: In the logging configuration file, use the following instance of DBAppender to connect to the database and write log events to the table:

```

<appender name="dblog" class="DBAppender">
    <param name="ConnectionString" value="DRIVER=TERADATA;UID=User1;
        PWD=*****;server=myserver;database=mydatabase;CATALOG=X;" />
    <param name="MaxBufferedEvents" value="300" />
    <param name="TableName" value="LOG" />
<!--
The column parameters must be specified in exactly the same order that the
columns occur in the table.
-->
    <param name="Column" value="sn" />
    <param name="Column" value="d" />
    <param name="Column" value="c" />

```

```

    <param name="Column" value="m" />
</appender>

```

FileAppender

Writes messages to the specified file in the specified path.

Valid in: XML configuration

Syntax

```

<appender class="FileAppender" name="appender-name">
  <param name="Append" value="TRUE | FALSE"/>
  <param name="Encoding" value="encoding-value"/>
  <param name="File" value="path-and-filename"/>
  <param name="FileNamePattern" value="path-and-filename-pattern"/>
  <param name="ImmediateFlush" value="TRUE | FALSE"/>
  <param name="Locale" value="locale"/>
  <param name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"/>
  <param name="Unique" value="TRUE | FALSE"/>
  <filter>
    <filter-definitions>
  </filter>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>

```

Syntax Description

class="FileAppender" name="appender-name"

specifies the user-assigned name for this instance of FileAppender.

Default: None

Requirement: This element is required.

name="Append" value="TRUE | FALSE"

controls how messages are written to the log file if the file already exists when logging begins. Specify one of the following values:

TRUE

appends new messages to the end of the existing file.

FALSE

erases the contents of the existing file and overwrites them with new messages.

Default: TRUE

Requirement: This parameter is not required.

Interaction: If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. For details, see [name=" Unique" on page 54](#).

name="Encoding" value="encoding-value"

specifies the encoding that is used to write messages to the file.

Default: The encoding setting that is in effect for the SAS session. For example, the ENCODING system option might be specified in the configuration file for a SAS server or for Base SAS. If the ENCODING system option is not specified for the SAS session, then the defaults that are described in the *SAS National Language Support (NLS): Reference Guide*

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the encoding that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="File" value="path-and-filename"

specifies the path and filename of the file to which messages are written.

Default: None

Requirement: This parameter is required if the FileNamePattern parameter is not specified.

Interaction: If both the File parameter and the FileNamePattern parameter are specified, then the File parameter takes precedence.

name="FileNamePattern" value="path-and-filename-pattern"

specifies the path to which the log file is written and the conversion pattern that is used to create the log filename. The conversion pattern can include the following characters:

%d

indicates where the current date appears. You can specify a date format or a date and time pattern in braces after %d if you want the date to appear in a format other than *yyyy-mm-dd*, or if you want to include additional information such as the hour.

See: “d Conversion Character” on page 108

%S{key}

indicates where system information (such as the host name, operating system, system description, or process ID) appears. You must specify a *key* to indicate the type of system information that appears.

See: “S Conversion Character” on page 113

For example, specify `c:\logs\MetadataServer_%d_%S{host_name}.log` if you want the log files to be written to the path `c:\logs\` and the filename to include the current date and the name of the metadata server host machine.

Default: None

Requirement: This parameter is required if the File parameter is not specified.

Interaction: If both the File parameter and the FileNamePattern parameter are specified, then the File parameter takes precedence.

name="ImmediateFlush" value="TRUE | FALSE"

determines whether messages are written to the file immediately or held in a buffer. Specify one of the following values:

TRUE

writes messages to the file immediately as they are received.

FALSE

holds messages in a buffer and writes them to the file when the buffer is full. The buffer size is 16 KB.

Default: FALSE

Requirement: This parameter is not required.

name="Locale" value="locale"

specifies the locale that is used to write messages to the file.

Default: The locale setting that is in effect for the SAS session. For example, the LOCALE system option might be specified in the configuration file for a SAS server or in the configuration file for Base SAS.

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the locale that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"

specifies the lowest event level that this appender processes. Events that are below the specified level are ignored. The valid values are listed here from lowest to highest.

Default: None

Requirement: No

See: [“Logging Thresholds” on page 16](#)

name="Unique" value="TRUE | FALSE"

creates a new file, with an underscore and a unique number appended to the filename, if the log file already exists when logging begins. Numbers are assigned sequentially from 0 to 32766.

For example, suppose `Events.log` is specified in *path-and-filename*. If the files `Events.log` and `Events.log_0` already exist, then the next log file that is created is named `Events.log_1`.

Default: FALSE

Requirement: This parameter is not required.

Interactions:

If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. If the log file already exists when logging begins, messages are logged as follows:

If Unique is set to TRUE and Append is set to either TRUE or FALSE, then messages are written to a new file with a unique number appended to the filename.

If Unique is set to FALSE and Append is set to TRUE, then messages are appended to the end of the existing file.

If Unique is set to FALSE and Append is set to FALSE, then the contents of the existing file are erased and overwritten with new messages.

filter-definitions

specifies the names and associated parameters of filters that limit the messages that are logged by this appender.

Default: None

Requirement: No

See: [Chapter 9, “Filters,” on page 121](#)

name="ConversionPattern" value="conversion-pattern"

specifies how the log message is written to the log.

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

Requirement: No

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

Details

FileAppender writes messages to the specified file in the specified path. When you create an instance of FileAppender, you can specify the following:

- how messages are written if the file already exists when logging begins. Messages can be appended to the end of the existing file, they can overwrite the existing file contents, or they can be written to a new file that has a unique name.
- whether to write messages immediately upon receipt or to hold them in a buffer.
- the minimum (threshold) event level to be logged.
- the locale and encoding to be used when writing to the file.
- a conversion pattern to be used for creating the filename.

The following best practices apply to FileAppender:

- Use of the Unique parameter is recommended to avoid overwriting log files. However, if numerous files are created that have the same root filename and different numerical suffixes, then the system must perform multiple comparisons to determine a unique number. To conserve system resources, consider specifying a *path-and-filename-pattern* that includes a unique identifier such as process ID (`%S{pid}`).

Example: Appending Messages to a File

The following instance of FileAppender writes messages to a file called `Events.log`. If the file already exists when logging begins, messages are appended to the end of the file.

```
<appender class="FileAppender" name="File">
  <param name="File" value="c:\logs\Events.log"/>
  <param name="Append" value="true"/>
  <param name="ImmediateFlush" value="true"/>
  <layout>
    <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
  </layout>
</appender>
```

FilteringAppender

Filters events based on thresholds and string values to determine whether the events should be passed to a referenced appender. You can specify a layout to be applied to the events before they are passed.

Valid in: XML configuration

Syntax

```
<appender class="FilteringAppender" name="appender-name">
  <appender-ref ref="referenced-appender-name"/>
  <filter>
    <filter-definitions>
  </filter>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
  <param name="Locale" value="locale"/>
  <param name="PropagateLayout" value="TRUE | FALSE"/>
  <param name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"/>
</appender>
```

Syntax Description

class="FilteringAppender" name="appender-name"

specifies the user-assigned name for this instance of FilteringAppender.

Default: None

Requirement: These element attributes are required.

ref="referenced-appender-name"

specifies the appender that events are to be passed to.

Requirement: This element attribute is required.

filter-definitions

specifies the names and associated parameters of filters that limit the messages that are passed to the referenced appender.

Default: None

Requirement: Filters are not required.

See: [Chapter 9, “Filters,”](#) on page 121

name="ConversionPattern" value="conversion-pattern"

specifies formatting that is to be applied to the event before it is passed to the referenced appender. The resulting string becomes the %m portion of the event in the layout of the referenced appender.

Default: None. If a conversion pattern is not specified, then the log message is formatted only by the layout that is specified in the referenced appender.

Requirement: This parameter is not required.

See: [Chapter 8, “Pattern Layouts,”](#) on page 95

name="Locale" value="locale"

specifies the locale that is used when the specified layout is applied to the event.

Default: The locale setting that is in effect for the SAS session. For example, the LOCALE system option might be specified in the configuration file for a SAS server or in the configuration file for Base SAS.

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the locale that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="PropagateLayout" value="TRUE | FALSE"

specifies whether the layout that is specified in the conversion pattern is to be applied to events before they are passed to the referenced appender. Specify one of the following values:

TRUE

applies the specified layout to events before they are passed to the referenced appender. The resulting string becomes the %m portion of the event in the layout of the referenced appender.

FALSE

passes events to the referenced appender without applying the specified layout. Messages are formatted only by the layout that is specified in the referenced appender.

Default: TRUE

Requirement: This parameter is not required.

name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"

specifies the lowest event level that this appender processes. Events that are below the specified level are ignored. The valid values are listed here from lowest to highest.

Default: None

Requirement: This parameter is not required.

See: [“Logging Thresholds” on page 16](#)

Details

FilteringAppender enables you to do one or both of the following:

- filter events based on thresholds and string values to determine whether the events should be passed to a referenced appender.
- apply a layout to events before they are passed to the referenced appender. The resulting string becomes the %m portion of the event in the layout of the referenced appender.

Since FilteringAppender is an intermediate appender rather than a logging destination, it must be configured with an appender reference.

The primary use of FilteringAppender is to specify different layouts for different categories of events that are to appear together in the same log. Specify a separate instance of FilteringAppender for each event category that requires a different layout. After the layout is applied, the resulting string becomes the %m portion of the event in the layout of the referenced appender. You can specify filters to limit the events that are passed.

If you do not specify a layout, or if you set the PropagateLayout parameter to FALSE, then events are formatted only by the layout of the referenced appender.

Example

The following logging configuration file writes two different categories of events to the same log file:

- Events from the App.Program logger. These events are written directly to the log file.

- Events from loggers other than App.Program, if they contain the word “state.” For these events, a layout is applied that includes the event's level and logger followed by the message. The resulting string becomes the %m portion of the event in the log file's layout.

```
<?xml version="1.0" encoding="UTF-8"?>
<logging:configuration xmlns:logging="http://support.sas.com/xml/logging/1.0">
  <!-- Write just the message portion of the event to the log file. -->
  <appender name="file" class="FileAppender">
    <param name="Append" value="false" />
    <param name="FileNamePattern" value="logfile.%S{pid}.log" />
    <layout>
      <param name="ConversionPattern" value="%m" />
    </layout>
  </appender>
  <!--
    Include only the events that contain the word "state," and
    prepend the level and the logger name of the event to the
    message.
  -->
  <appender name="filter" class="FilteringAppender">
    <appender-ref ref="file" />
    <filter class="StringMatchFilter">
      <param name="StringToMatch" value="state" />
      <param name="AcceptOnMatch" value="true" />
    </filter>
    <filter class="DenyAllFilter" />
    <layout>
      <param name="ConversionPattern" value="%c - %p - %m" />
    </layout>
  </appender>
  <!-- Send App.Program messages directly to the log file -->
  <logger name="App.Program" additivity="false">
    <appender-ref ref="file" />
    <level value="INFO" />
  </logger>
  <!--
    Send all other events to the filter so that a different layout
    can be applied.
  -->
  <root>
    <appender-ref ref="filter" />
    <level value="INFO" />
  </root>
</logging:configuration>
```

IOMServerAppender

Writes log messages from any IOM server to a volatile runtime cache.

Valid in: XML configuration

Syntax

```
<appender class="IOMServer" name="appender-name">
  <param name="MaxEntries" value="maximum-number-of-entries"/>
  <param name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"/>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>
```

Syntax Description

name="MaxEntries" value="maximum-number-of-entries"

an integer that specifies the maximum number of messages that are stored in the cache. When the maximum number is reached, the oldest messages are deleted as new messages are added.

Default: 10000

Range: 0 to 1000000

Requirement: This parameter is not required.

name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"

specifies the lowest event level that this appender processes. Events that are below the specified level are ignored. The valid values are listed here from lowest to highest.

Default: None

Requirement: This parameter is not required.

See: [“Logging Thresholds” on page 16](#)

name="ConversionPattern" value="conversion-pattern"

specifies how the log message is written to the log.

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

Requirement: This parameter is not required.

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

Details

IOM Server Appender writes log messages from any IOM server (for example, a SAS Metadata Server, a SAS OLAP Server, or a SAS Stored Process Server) to a volatile runtime cache. The contents of the cache are available for display on the **Log** tab of SAS Management Console. For more information, see “Use the Log Tab in Server Manager” in the *SAS Intelligence Platform: System Administration Guide*.

If you perform a planned deployment, then IOMServerAppender definitions are included in the logging configurations for most of your SAS servers. Follow these best practices when modifying these definitions:

- You can adjust the MaxEntries value to capture a larger or smaller number of messages for display.
- Do not change the message layout. Changing the message layout could cause messages to be captured incorrectly.

Note: A location for temporary files must be defined on the host operating system. If a location has not been defined, then the process that is being logged fails with the

following message: **Error creating IOMServerAppender index cache. The location required for storing temporary utility files does not exist.** If a location for temporary files is not already defined, use one of the following procedures to define it:

- On Windows systems, define the TEMP environment variable.
- On UNIX systems, create the directory `/tmp`.
- On z/OS systems, create the directory `/tmp` if you are using a UNIX file system (UFS); or submit the following TSO command:

```
ALLOC UNIT(SYSDA) BLOCK(8192) SPACE(1280,1280)
```

Example

The following instance of `IOMServerAppender` writes a maximum of 10,000 messages to a runtime cache. When the cache contains 10,000 messages, the oldest messages are deleted as new messages are added.

```
<appender class="IOMServerAppender" name="IOMServer">
  <param name="MaxEntries" value="10000"/>
  <layout>
    <param name="ConversionPattern" value="%d %-5p [%t] %X{Client.ID}:%u - %m"/>
  </layout>
</appender>
```

JavaAppender

Sends messages to a custom Java class.

Valid in: XML configuration

Syntax

```
<appender class="JavaAppender" name="appender-name">
  <param name="Class" value="class"/>
  <param name="ClassPath" value="class-path"/>
  <param name="MaxBufferedEvents" value="maximum-buffered-events"/>
  <param name="SASEncodedPassword" value="SAS-encoded-password"/>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>
```

Syntax Description

class="JavaAppender" name="appender-name"

specifies the user-assigned name for this instance of `JavaAppender`.

Default: None

Requirement: These element attributes are required.

name="Class" value="class"

specifies the custom Java class to which events are to be sent. The class that you specify must support the following methods:

- A no-argument constructor for creating the object.
- An **append** method for handling each event:

```
void append( int level, String logger, String msg )
```

The class can also support the following methods:

void setOption(String name, Object value)	specifies the handling for any parameters in the configuration file that are not recognized by JavaAppender
void activate()	is called after all of the options have been set and before any calls to the append method
void destroy	is called when the appender is destroyed

Default: com.sas.jms.logging.JMSAppender. If you use the default class, then JavaAppender has the same functionality and uses the same parameters as JMSAppender. For details, see “[JMSAppender](#)” on page 67.

Requirement: This parameter is required if you want to use a class other than com.sas.jms.logging.JMSAppender.

name="ClassPath" value="class-path"

specifies the path for classes that are included in this appender configuration.

Default: None

Requirement: This parameter is not required.

Interaction: When JavaAppender searches for classes to load, the CLASSPATH environment variable takes precedence over the ClassPath parameter, and the ClassPath parameter takes precedence over the JARs that are provided by SAS.

name="MaxBufferedEvents" value="maximum-buffered-events"

specifies the maximum number of events that can be waiting to be sent to the Java class. JavaAppender stores events in an internal list that is bounded by the number specified in this parameter. A worker thread removes events from the list and sends them to the Java class. In the meantime, SAS can continue processing without waiting for the Java class. When the list contains the specified number of events, the appender blocks further events until the list can accommodate them.

TIP A high number favors performance over reliability. A catastrophic error could cause events in the list to be lost. A high number also increases the memory usage of the appender.

TIP A low number favors reliability and memory usage over performance.

Default: Infinite

Requirement: This parameter is not required.

name="SASEncodedPassword" value="SAS-encoded-password"

specifies a plain-text password or a password that has been encoded by using sas001, sas002, or sas003 encoding. If the password is encoded, JavaAppender converts it to plain text before sending it to the Java class.

TIP For information about how to obtain the encoded password, see Chapter 40, “PWENCODE Procedure” in *Base SAS Procedures Guide*.

Default: None.

Requirement: This parameter is not required.

name="ConversionPattern" value="conversion-pattern"

specifies how the log message is written to the Java class.

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

Requirement: This parameter is not required.

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

Details

If additional parameters are needed by your custom Java class, you can include them in the appender configuration. Any parameters that JavaAppender does not recognize are passed to the custom Java class.

Examples

Example 1: Displaying a Supplementary Log Window for Trace Events

This example uses a custom Java class to display trace events in a separate log window in the SAS windowing environment, while preserving the contents of the main Log window.

Step 1: Create a custom Java class called TraceWindow.

Note that the main class cannot inherit directly from JFrame. To prevent a HeadlessException, you must set the java.awt.headless system property before the JFrame constructor runs.

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.Font;

public class TraceWindow
{
    JFrame window;
    JTextArea console;
    String separator;

    public TraceWindow() {
        System.setProperty( "java.awt.headless", "false" );
        separator = System.getProperty("line.separator");
    }

    public void activate() {
        window = new JFrame( "Trace window" );
        console = new JTextArea();
        console.setFont( new Font("Courier New", Font.PLAIN, 12) );
        console.setEditable( false );
        JScrollPane scroll = new JScrollPane( console );
        window.getContentPane().add(scroll);
        window.setSize( 600, 600 );
        window.setLocation( 200, 200 );
        window.setVisible( true );
    }
}
```

```

public void terminate() {
    window.dispose();
}

public void append( int level, String logger, String msg ){
    window.setVisible( true );
    console.append( msg );
    console.append( separator );
    console.setCaretPosition( console.getText().length() );
}
}

```

Step 2: Create the following logging configuration file. The file specifies JavaAppender with the custom Java class TraceWindow, which is displayed in the preceding code. Because of the specified threshold, only trace events are sent to the custom class.

```

<?xml version="1.0"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/"
    threshold="trace">
  <appender name="java" class="JavaAppender">
    <param name="Class" value="TraceWindow"/>
    <layout>
      <param name="ConversionPattern" value="[%F:%L] %c %6p - %m"/>
    </layout>
  </appender>
</root>
  <level value="info"/>
  <appender-ref ref="java"/>
</root>
</logging:configuration>

```

Example 2: Sending Events to a Database Using a JDBC Driver

This example uses a custom Java class to connect to a MySQL server via JDBC and write events to a table.

Step 1: Use the following MySQL code to create the database and the table:

```

CREATE DATABASE log;
CREATE TABLE log ( level varchar(10), logger varchar(50), msg varchar(500) );

```

Step 2: Create the following Java class, called LogToJDBC. This class uses JDBC to connect to the MySQL server and writes events to the table.

```

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;

class LogToJDBC {
    String          driver;
    String          url;
    String          tablename;
    String          username;
    String          password;
    long           rowsetSize;
    long           eventCount;

    Connection     connection;
    PreparedStatement stmt;
}

```

```

public LogToJDBC() {
    /*-----+
    | Nothing to do in the constructor. |
    +-----*/
}

/*-----+
| Appender API |
+-----*/
public void setOption( String name, Object value ) {
    if( name.equalsIgnoreCase( "driver" ) ) {
        driver = value.toString();
    } else if( name.equalsIgnoreCase( "url" ) ) {
        url = value.toString();
    } else if( name.equalsIgnoreCase( "tablename" ) ) {
        tablename= value.toString();
    } else if( name.equalsIgnoreCase( "username" ) ) {
        username = value.toString();
    } else if( name.equalsIgnoreCase( "password" ) ) {
        password = value.toString();
    } else if( name.equalsIgnoreCase( "rowsetsize" ) ) {
        rowsetsize = Long.valueOf(value.toString());
    }
}

public void activate() throws Exception {
    /*-----+
    | Set up all of our JDBC objects and set any properties we |
    | cached from setOption. |
    +-----*/
    Class.forName( driver );
    connection = DriverManager.getConnection( url, username,
                                             password );
    connection.setAutoCommit( false );
    stmt = connection.prepareStatement( "insert into " + tablename +
        " (level, logger, msg ) values (?, ?, ?)" );
}

public void append( int level, String logger, String msg ) throws Exception {
    /*-----+
    | Set the columns of our statement and if we have batched |
    | enough rows then commit them to the data base. |
    +-----*/
    stmt.setString( 1, new Integer(level).toString() );
    stmt.setString( 2, logger );
    stmt.setString( 3, msg );
    stmt.addBatch();
    eventCount++;
    if( eventCount >= rowsetsize )
    {
        stmt.executeBatch();
        connection.commit();
        eventCount = 0L;
    }
}

```

```

public void destroy() throws Exception {
    /*-----+
    | Send any batched events to the server.          |
    +-----*/
    if( eventCount > 0 )
    {
        stmt.executeBatch();
        connection.commit();
        eventCount = 0L;
    }
    /*-----+
    | Clean up our objects if we have any.          |
    +-----*/
    if( stmt != null )
        stmt.close();
    if( connection != null )
        connection.close();
    }
}

```

Step 3: Create the following logging configuration file. This file specifies JavaAppender with the custom Java class LogToJDBC, which is displayed in the preceding code. The Driver, URL, TableName, UserName, Password, and RowSetSize parameters are passed to LogToJDBC, which uses them to connect to the server and to write to the table.

```

<?xml version="1.0" encoding="UTF-8" ?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">
  <appender name="java" class="JavaAppender">
    <param name="classpath"
      value="mysql-connector-java-commercial-5.0.8-bin.jar"/>
    <param name="class" value="LogToJDBC"/>
    <param name="Driver" value="com.mysql.jdbc.Driver"/>
    <param name="URL" value="jdbc:mysql://mysql.example.com:3306/log"/>
    <param name="TableName" value="log"/>
    <param name="UserName" value="myusername"/>
    <param name="Password" value="mypassword"/>
    <param name="RowSetSize" value="1000" />
    <param name="MaxBufferedEvents" value="2000" />
    <layout>
      <param name="conversionpattern" value="%sn %d %c %p %m"/>
    </layout>
  </appender>
</root>
  <level value="info" />
  <appender-ref ref="java" />
</root>
</logging:configuration>

```

Example 3: Sending Events to log4j Appenders

This example uses a custom Java class to invoke a log4j configuration. The example uses org.apache.log4j.net.SocketServer to listen for events, which are then sent to the log4j ConsoleAppender. By following these steps, you can send events to any log4j appender.

Step 1: Create a custom Java class called `SendToLog4j` that sends events from SAS to `log4j`:

```
import org.apache.log4j.*;

public class SendToLog4j
{
    public SendToLog4j(){
        PropertyConfigurator.configure(System.getProperty("log4j.configuration"));
    }

    public void append( int level, String logger, String msg ){
        Logger l = Logger.getLogger( logger );
        switch( level ) {
            case 0:
            case 1:
            case 2:
                l.log( Level.TRACE, msg );
                break;
            case 3:
                l.log( Level.DEBUG, msg );
                break;
            default:
            case 4:
                l.log( Level.INFO, msg );
                break;
            case 5:
                l.log( Level.WARN, msg );
                break;
            case 6:
                l.log( Level.ERROR, msg );
                break;
            case 7:
                l.log( Level.FATAL, msg );
                break;
        }
    }
}
```

Step 2: Create a SAS logging configuration file called `socket.xml`. In the file, specify `JavaAppender` with the custom Java class `SendToLog4j`.

```
<?xml version="1.0"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/"
    threshold="trace">
    <appender name="java" class="JavaAppender">
        <param name="class" value="SendToLog4j"/>
        <layout>
            <param name="ConversionPattern" value="%d - %S{hostname} - %S{pid}
                - %c - %m"/>
        </layout>
    </appender>
</root>
    <level value="info"/>
    <appender-ref ref="java"/>
</root>
</logging:configuration>
```

Step 3: Create a client-side log4j configuration file called `client.properties`. In the file, specify the use of `SocketAppender` to listen for events.

```
log4j.rootLogger=DEBUG,A2
log4j.appender.A2=org.apache.log4j.net.SocketAppender
log4j.appender.A2.Port=55555
log4j.appender.A2.RemoteHost=localhost
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

Step 4: Create a server-side log4j configuration file called `server.properties`. In the file, specify the use of `ConsoleAppender`. (Any log4j appender could be specified.)

```
log4j.rootLogger=DEBUG,A2
log4j.appender.A2=org.apache.log4j.net.SocketAppender
log4j.appender.A2.Port=55555
log4j.appender.A2.RemoteHost=localhost
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

Step 5: Use the following command to start the socket server. The command specifies the `server.properties` log4j configuration file.

```
java -classpath log4j.jar org.apache.log4j.net.SocketServer 55555
server.properties
```

Step 6: Specify the following options when you start SAS. The first option specifies the `socket.xml` logging configuration file, and the second option passes the `client.properties` log4j configuration file to the SAS Java environment.

```
-logconfigloc socket.xml
-jreoptions '(-Dlog4j.configuration=client.properties)'
```

JMSAppender

Send messages to a message queue by using the Java Message Service (JMS) interfaces.

Valid in: XML configuration

Syntax

```
<appender class="JMSAppender" name="appender-name">
  <param name="ClassPath" value="class-path"/>
  <param name="MaxBufferedEvents" value="maximum-buffered-events"/>
  <param name="Persistent" value="TRUE | FALSE"/>
  <param name="Factory" value="factory"/>
  <param name="Queue | Destination" value="queue-name"/>
  <param name="UserName" value="user-name"/>
  <param name="SASEncodedPassword" value="SAS-encoded-password"/>
  <param name="Prioritize" value="TRUE | FALSE"/>
  <param name="Priority" value="priority"/>
  <param name="TimeToLive" value="time-to-live"/>
</appender>
<layout>
  <param name="ConversionPattern" value="conversion-pattern"/>
</layout>
```

```
</appender>
```

Syntax Description

class="JMSAppender" name="appender-name"

specifies the user-assigned name for this instance of JMSAppender.

Default: None

Requirement: These element attributes are required.

name="ClassPath" value="class-path"

specifies the path for classes that are included in this appender configuration.

Default: None

Requirement: This parameter is not required.

Interaction: When JMSAppender searches for classes to load, the CLASSPATH environment variable takes precedence over the ClassPath parameter, and the ClassPath parameter takes precedence over the JARs that are provided by SAS.

name="MaxBufferedEvents" value="maximum-buffered-events"

specifies the maximum number of events that can be waiting to be sent to the message queue. JMSAppender stores events in an internal list that is bounded by the number that is specified in this parameter. A worker thread removes events from the list and sends them to the message queue. In the meantime, SAS can continue processing without waiting for the JMS provider. When the list contains the specified number of events, the appender blocks further events until the list can accommodate them.

TIP A high number favors performance over reliability. A catastrophic error could cause events in the list to be lost. A high number also increases the memory usage of the appender.

TIP A low number favors reliability and memory usage over performance.

Default: Infinite

Requirement: This parameter is not required.

name="Persistent" value="TRUE | FALSE"

specifies whether the persistent delivery mode is to be used to send messages to the queue.

TRUE

specifies the use of the persistent delivery mode to send messages to the queue. This mode instructs the JMS provider to store messages after they are sent to ensure that they are not lost in transit if a provider failure occurs.

FALSE

specifies the use of the nonpersistent delivery mode to send messages to the queue. If you specify this mode, the JMS provider is not required to store messages or otherwise guarantee that they will be preserved if the provider fails.

TIP The nonpersistent delivery mode can improve performance and reduce storage overhead on the message queue server. However, you should use this mode only if it is acceptable for messages to be missed.

Default: TRUE

Requirement: This parameter is not required.

name="Factory" value="factory"

specifies the name of a connection factory that is administered through the Java Naming and Directory Interface (JNDI) and that is to be used to connect to the message queue.

Default: None

Requirement: This parameter is required.

name="Queue" value="queue-name"

specifies the name of the JNDI-administered queue to which messages are to be sent.

Alias: name="Destination"

Default: None

Requirement: This parameter is required.

name="Username" value="user-name"

specifies the user name that is to be used to connect to the message queue.

Default: If a user name is not specified, then the user name of the process owner is used to connect to the message queue.

Requirement: This parameter is not required.

name="SASEncodedPassword" value="SAS-encoded-password"

specifies the password that is to be used with *user-name* to connect to the message queue. You can specify a plain-text password or a password that has been encoded by using *sas001*, *sas002*, or *sas003* encoding. If the password is encoded, JMSAppender converts it to plain text before connecting to the queue.

TIP For information about how to obtain the encoded password, see Chapter 40, “PWENCODE Procedure” in *Base SAS Procedures Guide*.

Default: None.

Requirement: This parameter is not required.

name="Prioritize" value="TRUE | FALSE"

specifies whether messages are to be prioritized based on the level of the event.

TRUE

sets the message priority based on the level of the event, as follows:

Level	Assigned Message Priority
Trace	2
Debug	3
Info	4
Warn	5
Error	6
Fatal	7

Some message queuing systems, such as IBM WebSphere MQ, might return events with a higher priority to a caller before events with a lower priority, regardless of when the events were added to the queue.

FALSE

sets the message priority to **4** for all events.

Default: FALSE

Requirement: This parameter is not required.

Interaction: If both the Prioritize parameter and the Priority parameter are specified, then the Priority parameter takes precedence.

name="Priority" value="priority"

specifies an integer that is to be assigned as the message priority for each message that is written to the queue.

Default: None. If this parameter is not specified, then the Prioritize parameter determines how priorities are assigned.

Range: 0 – 9

Requirement: This parameter is not required.

Interaction: If both the Prioritize parameter and the Priority parameter are specified, then the Priority parameter takes precedence.

name="TimeToLive" value="time-to-live"

specifies the message time-to-live in milliseconds. When a message is sent, JMS computes the message's expiration time by adding the time-to-live value to the current Greenwich Mean Time. If time-to-live is specified as zero, the message does not expire.

Default: 0

Requirement: This parameter is not required.

name="ConversionPattern" value="conversion-pattern"

specifies how the log message is written to the queue.

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

Requirement: This parameter is not required.

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

Details

If additional JNDI parameters (for example, `java.naming.factory.initial` or `java.naming.provider.url`) are needed by your messaging provider, you can include them in the appender configuration. Any parameters that JMSAppender does not recognize are passed to the `javax.naming.InitialContext` constructor in the JNDI initial context as part of the `HashMap` parameter.

Note: You can also use the `JREOPTIONS` system option to pass JNDI parameters directly to SAS. See:

- “JREOPTIONS System Option: Windows” in *SAS Companion for Windows*
- “JREOPTIONS System Option: UNIX” in *SAS Companion for UNIX Environments*
- “JREOPTIONS= System Option: z/OS” in *SAS Companion for z/OS*

Depending on the provider, you might be able to use other methods to specify these parameters.

TIP By default, JMSAppender uses the Java class `com.sas.logging.appenders.JMSAppender`. If you want events to be handled by a custom Java class, use `JavaAppender`.

TIP When the Java Runtime Environment attempts to append messages to a message queue, additional events might be generated. These events are captured by the `App.tkjni` logger and are ignored by JMSAppender. If you want to log these events, specify a different appender (for example, `FileAppender`) for the `App.tkjni` logger in your logging configuration.

See also:

- <http://java.sun.com/products/jms> for details about the Java Message Service.
- *Application Messaging with SAS* for details about SAS language interfaces to JMS.

Examples

Example 1: Sending Events to IBM WebSphere MQ

The following instance of JMSAppender writes messages to a queue called TEST.Q that is managed by IBM WebSphere MQ. The JNDI objects are located in the folder **C:\JNDI** on the local machine.

```
<appender name="java" class="JMSAppender">
  <!-- Common properties for all Java Appender configurations -->

  <!-- Properties specific to the Default JMSAppender implementation -->
  <param name="persistent" value="true"/>

  <!-- This is an example configuration for WebSphere MQ. -->
  <!-- This example assumes that the Websphere MQ classes -->
  <!-- are in your classpath environment variable. -->
  <param name="java.naming.factory.initial"
    value="com.sun.jndi.fscontext.RefFSContextFactory"/>
  <param name="java.naming.provider.url" value="file:/c:/JNDI"/>
  <param name="factory" value="TEST.FACTORY"/>
  <param name="destination" value="TEST.Q"/>

  <layout>
    <param name="ConversionPattern" value="%d - %c -%m"/>
  </layout>
</appender>
```

Example 2: Sending Events to Apache ActiveMQ

The following instance of JMSAppender writes messages to a queue called TEST.Q that is managed by Apache ActiveMQ:

```
<appender class="JMSAppender" name="java">
  <!-- Common properties for all Java Appender configurations -->
  <param name="classpath" value="C:\Program
    Files\apache-activemq-5.2.0\activemq-all-5.2.0.jar"/>

  <!-- Properties specific to the Default JMSAppender implementation -->
  <param name="persistent" value="true"/>

  <!-- Active MQ example configuration -->
  <param name="java.naming.factory.initial"
    value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/>
  <param name="java.naming.provider.url" value="tcp://localhost:61616"/>
  <param name="factory" value="ConnectionFactory"/>
  <param name="destination" value="TEST.Q"/>
  <!-- An additional property is required only for Active MQ to map
    the queue name into the JNDI namespace. Use the form
    <param name="queue.[jndiName]" value="[physicalName]"/>.
    See http://activemq.apache.org/jndi-support.html. -->
```

```

<param name="queue.TEST.Q" value="TEST.Q"/>

<layout>
  <param name="ConversionPattern" value="%d - %c - %m"/>
</layout>
</appender>

```

RollingFileAppender

Writes messages to the specified file in the specified path, and begins writing messages to a new file that has a different name when specified criteria are met.

Valid in: XML configuration

Syntax

```

<appender class="RollingFileAppender" name="appender-name">
  <param name="Append" value="TRUE | FALSE"/>
  <param name="Encoding" value="encoding-value"/>
  <param name="File" value="path-and-filename"/>
  <param name="ImmediateFlush" value="TRUE | FALSE"/>
  <param name="Locale" value="locale"/>
  <param name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"/>
  <param name="Unique" value="TRUE | FALSE"/>
  <filter>
    <filter-definitions>
  </filter>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
  <rollingPolicy class="FixedWindowRollingPolicy | TimeBasedRollingPolicy">
    <rollingPolicy-parameters>
  </rollingPolicy>
  <triggeringPolicy class="SizeBasedTriggeringPolicy | FilterBasedTriggeringPolicy">
    <triggeringPolicy-parameters>
  </triggeringPolicy>
</appender>

```

Syntax Description

class="RollingFileAppender" name="appender-name"

specifies the user-assigned name for this instance of RollingFileAppender.

Default: None

Requirement: These attributes are required.

name="Append" value="TRUE | FALSE"

controls how messages are committed to the log file if the file already exists when logging begins. Specify one of the following values:

TRUE

appends new messages to the end of the existing file.

FALSE

erases the contents of the existing file and overwrites them with new messages.

Default: TRUE

Requirement: This parameter is not required.

Interaction: If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. For details, see [the Unique parameter on page 74](#).

name="Encoding" value="encoding-value"

specifies the encoding that is used to write messages to the file.

Default: The encoding setting that is in effect for the SAS session. For example, the ENCODING system option might be specified in the configuration file for a SAS server or for Base SAS. If the ENCODING system option is not specified for the SAS session, then the defaults that are described in the *SAS National Language Support (NLS): Reference Guide* are used.

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the encoding that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="File" value="path-and-filename"

specifies the path and filename of the file to which messages are written.

Default: None

Requirement: This parameter is not required.

Interaction: This filename overwrites any value that you specify for *path-and-filename-pattern* in a RollingPolicy or TriggeringPolicy configuration.

name="ImmediateFlush" value="TRUE | FALSE"

determines whether messages are written to the file immediately or held in a buffer. Specify one of the following values:

TRUE

writes messages to the file immediately as they are received.

FALSE

holds messages in a buffer and writes them to the file when the buffer is full. The default buffer size is 16 KB.

Default: FALSE

Requirement: This parameter is not required.

name="Locale" value="locale"

specifies the locale that is used to write messages to the file.

Default: The locale setting that is in effect for the SAS session. For example, the LOCALE system option might be specified in the configuration file for a SAS server or for Base SAS.

For logging processes that run outside a SAS session (for example, logging for the SAS Object Spawner), the default is the locale that is specified in the operating system settings.

Requirement: This parameter is not required.

See: *SAS National Language Support (NLS): Reference Guide*

name="Threshold" value="TRACE | DEBUG | INFO | WARN | ERROR | FATAL"

specifies the lowest event level that this appender processes. Events that are below the specified level are ignored. The valid values are listed here from lowest to highest.

Default: None

Requirement: This parameter is not required.

See: [“Logging Thresholds” on page 16](#)

name="Unique" value="TRUE | FALSE"

creates a new file, with an underscore and a unique number appended to the filename, if the log file already exists when logging begins. Numbers are assigned sequentially from 0 to 32766.

For example, suppose `Events_%d.log` is specified in *path-and-filename-pattern* for `TimeBasedRollingPolicy`. If the current date is August 3, 2008, and a file already exists that has the name `Events_20080803.log`, then the next log file that is created is named `Events_20080803_0.log`. If a file already exists that has the name `Events_20080803_0.log`, then the next log file that is created is named `Events_20080803_1.log`.

Default: FALSE

Requirement: This parameter is not required.

Interactions:

If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. If the log file already exists when logging begins, and if Unique is set to TRUE and Append is set to either TRUE or FALSE, then messages are written to a new file with a unique number appended to the filename.

If Unique is set to TRUE and `FixedWindowRollingPolicy` is specified, then a complete set of unique files is created when logging begins. For details, see the Interaction for [FixedWindowRollingPolicy on page 75](#)

filter-definitions

specifies the names and associated parameters of filters that limit the messages that are logged by this appender.

Default: None

Requirement: *filter-definitions* are not required.

See: [Chapter 9, “Filters,” on page 121](#)

name="ConversionPattern" value="conversion-pattern"

specifies how the log message is written to the log.

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

Requirement: This parameter is not required.

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

rollingPolicy class="FixedWindowRollingPolicy | TimeBasedRollingPolicy"

specifies the policy that controls the creation of new log files and filenames when messages roll over to a new file. SAS provides the following instances of `rollingPolicy`:

- [FixedWindowRollingPolicy on page 75](#)

- [TimeBasedRollingPolicy](#) on page 76

FixedWindowRollingPolicy

specifies a fixed set of filenames that include sequentially assigned index numbers. To specify when log files roll over, specify either **SizeBasedTriggeringPolicy** or **FilterBasedTriggeringPolicy** in the `TriggeringPolicy` parameter. When the specified criteria are met, the log files are rolled over as follows:

- The appender renames each existing log file by incrementing the index number in the filename by 1. For example, **Events03.log** is renamed to **Events04.log**, **Events02.log** is renamed to **Events03.log**, and **Events01.log** is renamed to **Events02.log**.
- If a file already exists that has a filename that includes *maximum-index*, then the messages in that file are overwritten. For example, if **Events04.log** already exists when rollover occurs, and if 4 is specified in *maximum-index*, then the contents of **Events04.log** are replaced with the contents of **Events03.log**.
- The appender creates a new file with *minimum-index* in the filename (for example, **Events01.log**), and subsequent messages are written to that file.

Use the following syntax to specify `FixedWindowRollingPolicy`:

```
<rollingPolicy class="FixedWindowRollingPolicy">
  <param name="FileNamePattern" value="path-and-filename-pattern"/>
  <param name="maxIndex" value="maximum-index"/>
  <param name="minIndex" value="minimum-index"/>
</rollingPolicy>
```

`name="FileNamePattern" value="path-and-filename-pattern"`

specifies the path to which the log file is written and the conversion pattern that is used to create the log filename when messages roll over to a new file. The conversion pattern can include the following characters:

`%i`

indicates where the index number is to appear. The index number is incremented by 1 each time a new file is created.

`%S{key}`

indicates where system information (such as the host name, operating system, or system description) appears. You must specify a *key* to indicate the type of system information that appears.

See: [“S Conversion Character” on page 113](#)

Default: None

Requirement: This parameter is not required.

Interaction: If both the `File` parameter and the `FileNamePattern` parameter are specified, then the `File` parameter takes precedence.

Example: Specify `c:\logs\MetadataServer_S{host_name}_%i.log` if you want the log files to be written to the path `c:\logs\` and if you want the files to be named `MetadataServer_host-name_01.log`, `MetadataServer_host-name_02.log`, and so on.

`name="maxIndex" value="maximum-index"`

specifies an integer that is the highest number to be used as an index in the log filename. For example, if you set *minimum-index* to 1 and *maximum-index* to 10, the appender creates a maximum of ten log files. When the maximum has been reached, the appender overwrites the most recently created file.

Default: 7

Range: 1–14

Requirement: This parameter is not required.

Interaction: If *maximum-index* is equal to *minimum-index*, then only one file is created.

`name="minIndex" value="minimum-index"`

specifies an integer that is the beginning number to be used as an index in the log filename. For example, if you set *minimum-index* to 3, the name of the first log file that is created will contain the characters **03** in the position that is specified by `%i` in the filename pattern.

Default: 1

Range: 1–14

Requirement: This parameter is not required.

Interactions:

If `Unique` is set to `TRUE` and `FixedWindowRollingPolicy` is specified, then a complete set of fixed window files is created when logging begins. If one or more sets of fixed window files already exist when logging begins, then a new set of fixed window files is created that has an underscore character and a unique number appended to each filename.

For example, if `Unique` is set to `TRUE` and `FixedWindowRollingPolicy` is specified with a filename pattern of `Events%i.log`, a `minimum-index` of 1, and a `maximum-index` of 4, then log files are created as follows:

When logging first begins, the following empty files are created: `Events01.log`, `Events02.log`, `Events03.log`, and `Events04.log`. Messages are written to `Events01.log` and are rolled over to the other files in the group as specified by the triggering policy.

The next time logging begins, the following set of files is created and written to: `Events01_0.log`, `Events02_0.log`, `Events03_0.log`, and `Events04_0.log`.

Each subsequent time that logging begins, a new set of files is created with a new unique suffix (for example, `_1`, `_2`, `_3`).

TimeBasedRollingPolicy

specifies the use of a log filename that contains the current date. You do not need to specify a value for `triggeringPolicy` when you use this policy. To specify when a new log file is created, you can specify either of the following options:

- Creation of (rollover to) a new log file whenever the generated filename differs from the current filename. This is the default behavior.

For example, if the filename includes the current year, month, and day, then a new file is created when the system date changes to a new day.

- Creation of a new log file only when a new session begins.

When rollover occurs, the message **Log continues in *path-and-filename*** is written to the end of the current file. The message **Log continued from *path-and-filename*** is written to the beginning of the newly created file.

Use the following syntax to specify `TimeBasedRollingPolicy`:

```
<rollingPolicy class="TimeBasedRollingPolicy">
  <param name="FileNamePattern" value="filename-pattern"/>
  <param name="rollOver" value="TRUE | FALSE"/>
</rollingPolicy>
```

`name="FileNamePattern" value="path-and-filename-pattern"`

specifies the path to which the log file is written and the conversion pattern that is used to create the log filename. The conversion pattern can include the following characters:

`%d`

indicates where the current date appears. You can specify a date format or a date and time pattern in braces after `%d` if you want the date to appear in a format other than `yyyy-mm-dd`, or if you want to include additional information such as the hour.

See: [Chapter 8, "Pattern Layouts," on page 95](#)

`%S{key}`

indicates where system information (such as the host name, operating system, or system description) appears. You must specify a *key* to indicate the type of system information that appears.

See: ["S Conversion Character" on page 113](#)

Default: None

Requirement: This parameter is required.

Interaction: If both the File parameter and the FileNamePattern parameter are specified, then the File parameter takes precedence.

Example: For example, specify `c:\logs\MetadataServer_%d_%S{host_name}.log` if you want the log files to be written to the path `c:\logs\` and the filename to include the current date and the name of the metadata server host machine.

`name="rollOver" value="TRUE | FALSE"`

indicates whether a new log file is created whenever the generated filename differs from the current filename. Specify one of the following values:

TRUE

creates (rolls over to) a new file whenever the generated filename differs from the current filename.

FALSE

creates a new log file only when a new session begins.

Default: TRUE

Requirement: This argument is not required.

triggeringPolicy class="SizeBasedTriggeringPolicy | FilterBasedTriggeringPolicy"

specifies the policy that determines when a new log file is created. SAS provides the following instances of triggeringPolicy:

- [SizeBasedTriggeringPolicy on page 77](#)
- [FilterBasedTriggeringPolicy on page 78](#)

SizeBasedTriggeringPolicy

specifies the creation of a new log file when the number of bytes in the current log file is greater than or equal to the specified *maximum-file-size*. Along with this policy, specify `FixedWindowRollingPolicy` in the `RollingPolicy` parameter to control how new log filenames are assigned and the number of files that are created.

Use the following syntax to specify `SizeBasedTriggeringPolicy`:

```
<triggeringPolicy class="SizeBasedTriggeringPolicy">
  <param name="MaxFileSize" value ="maximum-file-size">
</triggeringPolicy>
```

`name="MaxFileSize" value="maximum-file-size"`
 specifies the maximum size, in bytes, of the log file. When the log file reaches this size, messages roll over to a new file. You can use the suffix **KB** (for kilobytes), **MB** (for megabytes), or **GB** (for gigabytes) when you are specifying the size. For example, **10KB** is interpreted as **10240** bytes.

FilterBasedTriggeringPolicy

specifies the creation of a new log file when a log event is received that meets the specified filtering criteria. Along with this policy, specify `FixedWindowRollingPolicy` in the `RollingPolicy` parameter to control how new log filenames are assigned and the number of files that are created.

Use the following syntax to specify `FilterBasedTriggeringPolicy`:

```
<triggeringPolicy class="FilterBasedTriggeringPolicy">
  <param name="Filters">
    <filter-definitions>
  </param>
</triggeringPolicy>
```

filter-definitions

specifies the filters that are used to trigger rollover to a new log file.

See: [Chapter 9, “Filters,” on page 121](#)

Details

You can configure an instance of `RollingFileAppender` to do the following:

- roll over to a new log file when the system date or time changes (for example, every day or every hour)
- roll over to a new log file when a new session begins
- roll over to a new log file when the file reaches a specified size
- roll over to a new log file when log events match the specified filtering criteria

In addition, `RollingFileAppender` provides all of the functionality of [FileAppender on page 52](#).

The following best practices apply to `RollingFileAppender`:

- Use of the `Unique` parameter is recommended to avoid overwriting log files. However, if numerous files are created that have the same root filename and different numerical suffixes, then the system must perform multiple comparisons to determine a unique number. To conserve system resources, consider specifying a *path-and-filename-pattern* that includes a unique identifier such as process ID (`%S{pid}`).

Examples

Example 1: Roll Over to a New Log File Every Day

This `RollingFileAppender` configuration writes messages to a log file whose name contains the current date (for example, `MetadataServer_2012-03-01.log`). When the system date changes, messages roll over to a new log file whose name contains the new date (for example, `MetadataServer_2012-03-02.log`).

```
<appender class="RollingFileAppender" name="TimeBasedRollingFile">
  <param name="Append" value="true"/>
  <param name="ImmediateFlush" value="true"/>
```

```

<rollingPolicy class="TimeBasedRollingPolicy">
  <param name="FileNamePattern" value="c:\logs\MetadataServer_%d.log"/>
</rollingPolicy>
<layout>
  <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
</layout>
</appender>

```

Example 2: Roll Over to a New Log When a New Session Begins

This RollingFileAppender configuration writes messages to a log file whose name contains the current date (for example, **MetadataServer_2012-03-01.log**). When a new session begins, messages roll over to a new log file whose name contains the current date (for example, **MetadataServer_2012-03-02.log**).

```

<appender class="RollingFileAppender" name="TimeBasedRollingFile">
  <param name="Append" value="true"/>
  <param name="ImmediateFlush" value="true"/>
  <rollingPolicy class="TimeBasedRollingPolicy">
    <param name="FileNamePattern" value="c:\logs\MetadataServer_%d.log"/>
    <param name="rollOver" value="false"/>
  </rollingPolicy>
  <layout>
    <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
  </layout>
</appender>

```

Example 3: Roll Over to a New Log File When the File Reaches a Specified Size

In this example, RollingFileAppender writes messages to a log file whose name contains an index number. The first file that is created is called **MetadataServer_01.log**. When the size of **MetadataServer_01.log** is greater than or equal to 100 KB, the file is renamed to **MetadataServer_02.log**, and subsequent messages are written to a newly created instance of **MetadataServer_01.log**.

The next time **MetadataServer_01.log** reaches or exceeds 100 KB, **MetadataServer_02.log** is renamed to **MetadataServer_03.log**, **MetadataServer_01.log** is renamed to **MetadataServer_02.log**, and subsequent messages are written to a newly created instance of **MetadataServer_01.log**.

Rollover continues until nine files have been created, at which point the contents of **MetadataServer_09.log** are overwritten when rollover occurs.

```

<!-- Rolling log file based on log file size -->
<appender class="RollingFileAppender" name="FixedWindowRollingFile">
  <param name="Append" value="true"/>
  <param name="ImmediateFlush" value="true"/>
  <rollingPolicy class="FixedWindowRollingPolicy">
    <param name="FileNamePattern" value="c:\logs\MetadataServ_%i.log"/>
    <param name="minIndex" value="1"/>
    <param name="maxIndex" value="9"/>
  </rollingPolicy>
  <triggeringPolicy class="SizeBasedTriggeringPolicy">
    <param name="MaxFileSize" value="100KB"/>
  </triggeringPolicy>
  <layout>
    <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
  </layout>
</appender>

```

```

    </layout>
</appender>

```

sLogAppender

Is a reserved class. You should not define new instances of this appender.

Note: Various appender definitions that rely on sLogAppender are enabled by default for several SAS servers. These appender definitions enable SAS client- and SAS-middle tier applications to access SAS server internal logging facilities.

CAUTION: **Do not modify sLogAppender definitions that are provided in default logging configuration files.** Modifying these definitions results in unpredictable behavior.

UNXFacilityAppender

Writes messages to the syslogd logging facility in UNIX operating systems.

Valid in: XML configuration

Syntax

```

<appender class="UNXFacilityAppender" name="LOG">
  <param name="facilitycode" value="log_value"/>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>

```

Syntax Description

name="facilitycode" value="log_value"

The **facilitycode** configuration option specifies the system facility value that generated the message. It can have any of the following **log** values:

LOG_USER

specifies that messages that are generated by user processes are logged.

LOG_USER is the default value for **facilitycode**.

LOG_LOCAL0 through **LOG_LOCAL7**

specifies values that are reserved for use by your site.

See: For information about the generic elements of the appender syntax, see [“General Appender Syntax” on page 12](#)

Details

UNXFacilityAppender Usage

UNXFacilityAppender is supported on the Solaris, HP, Linux, and AIX operating systems. The logging facility that these operating systems provide is named **syslogd**. The **syslogd** daemon must be running before you can see the output that is sent to it by UNXFacilityAppender. To enable UNXFacilityAppender to communicate with **syslogd**, make sure that you have an entry in the **/etc/syslog.conf** file for the

user facility, the **local0** through **local7** facilities, or insert ***** before you start **syslogd**. You can also have entries for both the **user** facility and the **local0** through **local7** facilities.

Note: The ***** specifies all facilities. Use caution when specifying *****. It can cause facilities other than **user** and **local0** through **local7** to log to the destination.

These entries should have a format of '**<facility>.<priority><destination>**'. The following examples show the formats for entries in the configuration file:

```
user.info      /tmp/userinfo.log
```

or

```
*.info        /tmp/allinfo.log
```

For more information about **syslogd** on the UNIX platform that you are using, see the documentation written by that provider.

UNXFacilityAppender Diagnostic Levels

The following SAS logging facility's diagnostic levels are sent to the UNIX **syslogd** logging facility with the specified **syslogd** priorities:

Table 6.1 UNXFacilityAppender Diagnostic Levels

SAS Logging Level	syslogd Priority
TRACE, DEBUG	debug
INFO	info
WARN	warning
ERROR	err
FATAL	crit
no level	notice

syslogd priority is sometimes referred to as *level* in UNIX documentation for **syslogd**. When you use **syslogd** priority as a value for the SAS logging level, it specifies the severity of the message. It can also be used to specify the part of the system that generated the message. The following list contains definitions for the SAS logging levels that are listed in the table above:

TRACE, DEBUG

specifies messages that are helpful in debugging a program.

INFO

specifies messages that contain general information.

WARN

specifies messages that contain warnings.

ERROR

specifies messages that identify error conditions.

FATAL

specifies messages that identify critical conditions.

no level

specifies messages that identify conditions that require special attention. These conditions are not error conditions. If you do not specify a SAS logging level, then the `syslogd` value of `notice` is the default.

Example

The following example is a typical XML configuration file that specifies UNXFacilityAppender.

```
<appender class="UNXFacilityAppender" name="LOG">
  <layout>
    <param name="ConversionPattern"
      value="%d %-5p [%t] %c (%F:%L) - %m"/>
  </layout>
</appender>
<root>
  <level value="trace"/>
  <appender-ref ref="LOG"/>
</root>
```

WindowsEventAppender

Writes messages to the Windows Event log.

Valid in: XML configuration

Syntax

```
<appender class="WindowsEventAppender" name="eventAppender">
  <param name="AppName" value="application name"/>
</appender>
```

Syntax Description

name="AppName" value="application name"

identifies the role of the SAS process. This parameter is ignored on Windows 5. On Windows 6, this parameter configures the WindowsEventAppender as a distinct Event Tracing for Windows (ETW) publisher for each role that the SAS process might perform. Here are some valid values for **application name**:

- SAS Foundation
- Metadata Server
- OLAP Server

See: For information about the generic elements of the appender syntax, see [“General Appender Syntax” on page 12](#).

Details

WindowsEventAppender is a logging facility appender that supports event logging on Windows operating systems.

On Windows 5 (Windows XP and Windows Server 2003) WindowsEventAppender sends events to the Windows Event Log. The Event Log on these earlier versions of Windows might be easily overloaded. You should configure WindowsEventAppender so that the event log does not receive more events than it can handle.

On Windows 6 (Windows Vista and Windows Server 2008), WindowsEventAppender uses Event Tracing for Windows (ETW). The Event Viewer is a consumer of ETW on these versions of Windows. ETW exists in earlier Windows versions, but WindowsEventAppender uses ETW beginning with Version 6. Scalability is significantly improved in ETW.

Note that even on Windows 6, where each SAS role is associated with a distinct publisher that has its own logging channel, error events are sent to the channel for the Windows Application log. These error events are considered to be actionable events of the Windows **admin** channel.

Example

The following example is a typical XML configuration file that specifies the WindowsEventAppender. The parameter identifies the role of SAS as the SAS Foundation.

```
<?xml version="1.0" encoding="UTF-8"?>
<logging:configuration
xmlns:logging="http://www.sas.com/xml/logging/1.0/">
  <appender name="eventLog" class="WindowsEventAppender">
    <param name="AppName" value="SAS Foundation"/>
    <layout>
      <param name="ConversionPattern"
        value="%d %-5p [%t] %c (%F:%L) - %m"/>
    </layout>
  </appender>
</root>
  <level value="trace"/>
  <appender-ref ref="eventLog"/>
</root>
</logging:configuration>
```

ZOSFacilityAppender

Enables multiple instances of SAS in the z/OS operating system to write messages to a common location.

Valid in: XML configuration

Syntax

```
<appender class="ZOSFacilityAppender" name="appender-name">
  <param name="stream" value="stream-name"/>
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
</appender>
```

Syntax Description

name="stream" value="stream-name"

specifies the logstream for ZOSFacilityAppender.

Restriction: The logstream must be defined, using the IBM IXCMIAPU utility.

See: For information about the generic elements of the appender syntax, see [“General Appender Syntax” on page 12](#).

Details

ZOSFacilityAppender is a logging facility appender that supports event logging on z/OS operating systems. ZOSFacilityAppender enables multiple instances of SAS in the z/OS environment to write log information to a common location. If the z/OS environment uses the coupling facility, all of the SAS jobs that run on all of the z/OS systems that are on the sysplex can write their logs to the same location.

To use z/OS Facility Appender, specify a class of ZOSFacilityAppender that has a **stream** parameter that names the z/OS logstream that is used. You can define coupling facility (CF) logstreams or DASD-only logstreams for use with ZOSFacilityAppender. RACF access must be granted to any SAS user or batch SAS job that writes to the system logger.

Examples

Example 1: Configuring ZOSFacility Appender

The following example is a typical XML configuration file that specifies ZOSFacilityAppender.

```
<appender class="ZOSFacilityAppender" name="LOG">
  <param name="stream" value="SAS.LOG"/>
  <layout>
    <param name="ConversionPattern"
      value="%d %-5p [%t] %c (%F:%L) - %m"/>
  </layout>
</appender>
<root>
  <level value="trace"/>
  <appender-ref ref="LOG"/>
</root>
```

Example 2: Defining the Logstream

The following example is for use with direct access storage devices (DASD) only. The values that are included are not necessarily the values that you might want to use. The values that you specify can depend on the amount of data that is being processed, or on other variables.

```
//STEP1 EXEC PGM=IXCMIAPU
//SYSIN DD *
DATA TYPE(LOGR)
DEFINE LOGSTREAM NAME(SAS.LOG)
LS_DATACLAS(STD)
LS_MGMTCLAS(STD)
LS_STORCLAS(STD)
HLQ(IXGLOGR)
MODEL(NO)
LS_SIZE(0)
STG_MGMTCLAS(STD)
STG_STORCLAS(STD)
STG_DATACLAS(STD)
STG_SIZE(3000)
LOWOFFLOAD(60)
HIGHOFFLOAD(95)
STG_DUPLEX(YES)
DUPLEXMODE(UNCOND)
RETPD(3)
AUTODELETE(YES)
OFFLOADRECALL(YES)
DASDONLY(YES)
DIAG(NO)
LOGGERDUPLEX()
EHLQ(NO_EHLQ)
MAXBUFSIZE(64000)
```

Example 3: Using the SUBSYS DD Statement

To see the contents of the logstream, you can activate the LOGR subsystem by placing SUBSYS SUBNAME(LOGR) in SYS1.PARMLIB(IEFSSNxx). The following example shows how, after the LOGR subsystem has started, you can treat the logstream as a data set by using the SUBSYS DD statement.

```
//REPRO EXEC PGM=IDCAMS,REGION=20M
//SYSUT1 DD DISP=SHR,DSN=SAS.LOG,
// SUBSYS=(LOGR),
// DCB=(DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760)
//SYSUT2 DD SYSOUT=*,
// DCB=(DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760)
//SYSIN DD *
REPRO INFILE(SYSUT1) OFILE(SYSUT2)
/*
```

Example 4: Limiting the Output

The FROM and TO options in the following example are options of the LOGR subsystem that limit the output.

```
//IN DD DSN=SAS.LOG,DISP=SHR,
// DCB=(DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760)
// SUBSYS=(LOGR,IXGSEXIT,
// 'FROM=(1997/152,05:00),TO=(1997/153,23:59),GMT'
```

Example 5: Deleting the Logstream Contents

To delete the entire contents of the logstream, copy the IBM sample PROC IXGDELAB into `SYS1.PROCLIB` and start it with the logstream name as the parameter. Although this code does not delete the logstream, it resets the stream to an empty condition.

```
S IXGDELAB, LOGSTRM=SAS.LOG
```

ZOSWtoAppender

Directs SAS application messages to the z/OS operating system console.

Valid in: XML configuration

See: For more information about the z/OS write-to-operator (WTO) service macro and the parameters that are listed here, see the IBM reference document, *MVS Assembler Services Reference*.

For information about the generic elements of the appender syntax, see [“General Appender Syntax” on page 12](#).

Syntax

```
<appender class="ZOSWtoAppender" name="appender-name">
  <layout>
    <param name="ConversionPattern" value="conversion-pattern"/>
  </layout>
  <param name="routecode" value="value"/>
  <param name="desccode" value="value"/>
  <param name="mcsflag" value="HRDCPY | BRDCST | NOTIME/>
</appender>
```

Syntax Description

name="routecode" value="value"

specifies the routing code that is used for ZOSWtoAppender messages.

name="desccode" value="value"

specifies the descriptor code that is used for ZOSWtoAppender messages.

name="mcsflag" value="value"

specifies the `mcs` flag that is used for ZOSWtoAppender messages. Valid values for the `mcs` flag parameter are HRDCPY, BRDCST, NOTIME, and BUSYEXIT.

Details

ZOSWtoAppender is a logging facility appender that supports event logging on z/OS operating systems. ZOSWtoAppender enables you to direct SAS application messages, such as automation messages in the Admin message category, to the operating system consoles. ZOSWtoAppender uses the z/OS write-to-operator (WTO) service macro to direct the messages to the consoles. The appender also enables SAS servers to send messages about the status of applications to z/OS for automation purposes.

The “routecode”, “desccode”, and “mcsflag” parameters can be included multiple times in your XML file. For example, you can have multiple “routecode” parameters if you want to specify more than one routing code.

Example

The following example initiates `ZOSWtoAppender`; specifies a conversion pattern; specifies the name of the logger; and specifies values for the “routecode”, “desccode”, and “mcsflag” parameters.

```
<appender name="WTO" class="ZOSWtoAppender">
  <layout>
    <param name="ConversionPattern"
      value="%d %-5p [%t] %c (%F:%L) - %m"/>
  </layout>
  <param name="routecode" value="11"/>
  <param name="desccode" value="7"/>
  <param name="mcsflag" value="HRDCPY"/>
</appender>
<logger name="ADMIN.OPERATIONS">
  <level value="trace"/>
  <appender-ref ref="WTO"/>
</logger>
```


Chapter 7

Database Connection Options Reference for DBAppender

Dictionary	89
DBAppender Connection Options for DB2	89
DBAppender Connection Options for ODBC	90
DBAppender Connection Options for Oracle	92
DBAppender Connection Options for SAS Tables	93
DBAppender Connection Options for Teradata	93

Dictionary

DBAppender Connection Options for DB2

The following is an example of the basic syntax for specifying options in the `ConnectionString` parameter of a DBAppender configuration for a DB2 database. If you need additional details, contact SAS Technical Support.

Valid in: `ConnectionString` parameter of DBAppender configuration

See: [“DBAppender” on page 46](#)

Syntax

```
CATALOG=catalog-identifier;  
DATABASE=database-specification;  
DRIVER=DB2;  
PWD=password;  
UID=user-id;  
CONOPTS=(valid-DB2-compliant-database-connection-string);
```

Syntax Description

The data source connection options for a DB2 database include the following:

CATALOG=*catalog-identifier*

specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, `catalog=db2`).

Requirement: You must specify a catalog. For the DB2 database, this is a logical catalog name to use as an SQL catalog identifier.

DATABASE=database-specification

Specifies the name of the DB2 database.

Alias: DB=

DRIVER=DB2;

identifies the data source to which you want to connect, which is a DB2 database.

Requirement: You must specify the driver.

PWD=password;

specifies the DB2 password.

UID=user-id;

specifies the DB2 login user ID.

CONOPTS=(valid-DB2-compliant-database-connection-string);

specifies, within parentheses, a DB2 connection string. This parameter enables you to specify connection options that cannot be specified with the other `ConnectionString` parameters.

For example, you can use this parameter as follows to specify a DSN called `logdata`:

```
CONOPTS=(DSN=logdata);
```

Example: Connection String for DB2

```
DRIVER=DB2;DB=SAMPLE;UID=User1;PWD>Password1;CATALOG=db2;
```

DBAppender Connection Options for ODBC

The following is an example of the basic syntax for specifying options in the `ConnectionString` parameter of a `DBAppender` configuration for an ODBC-compliant database. If you need additional details, contact SAS Technical Support.

Valid in: `ConnectionString` parameter of `DBAppender` configuration

See: [“DBAppender” on page 46](#)

Syntax

```
CATALOG=catalog-identifier;
ODBC_DSN=ODBC-DSN-name
DRIVER=ODBC;
PWD=password;
UID=user-id;
CONOPTS=(valid-ODBC-compliant-database-connection-string);
```

Syntax Description

The data source connection options for an ODBC-compliant database include the following:

CATALOG=catalog-identifier

specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas.

For the Microsoft SQL Server, you can specify a logical name for the catalog, and map it to the native catalog name that is defined in the SQL Server. For example, to

specify the logical catalog logcat and map it to a native catalog called sqlcat, you would specify the following:

```
catalog=(logcat=sqlcat);
```

For databases that do not support native catalogs, any identifier is valid (for example, `catalog=myodbc`).

Default: If this parameter is omitted for the Microsoft SQL Server, the default setting CATALOG=* is used.

Requirements:

For the Microsoft SQL Server, which is a multiple-catalog database, CATALOG= is optional.

For databases that do not support native catalogs, you must specify a catalog.

ODBC_DSN=ODBC-DSN-name

specifies a valid ODBC-compliant database DSN that contains information for connecting to the ODBC-compliant database.

Interaction: To specify database-connection options that cannot be specified with the other ConnectionString parameters, you can use the CONOPTS= option along with the ODBC_DSN option. However, do not specify the ODBC DSN in both CONOPTS= and ODBC_DSN=.

DRIVER=ODBC;

identifies the type of data source to which you want to connect, which is ODBC.

Requirement: You must specify the driver.

PASSWORD=password;

specifies the password that is associated with the user ID.

Alias: PWD=

USER=user-id;

specifies the user ID for logging on to the ODBC-compliant database.

Alias: UID=

Default: If no user ID is specified, the default user for the database is used to log on.

CONOPTS=(valid-ODBC-compliant-database-connection-string);

specifies, within parentheses, an ODBC-compliant database connection string. This optional parameter enables you to specify connection options that cannot be specified with the other ConnectionString parameters. Here is an example:

- If the database uses a DSN, you can use this parameter to specify a value for DSN= or FILESDSN=. Here is an example:

```
CONOPTS=(DSN=LogSql);
```

- For databases that do not use a DSN, use this parameter to specify the DRIVER= keyword. Here is an example:

```
CONOPTS=(DRIVER=SQL Server);
```

Interaction: Do not specify the ODBC DSN in both CONOPTS= and ODBC_DSN=.

Example: Connection String for ODBC

The following example uses ODBC to connect to a Microsoft SQL Server:

```
DRIVER=ODBC;ODBC_DSN=LogSql;UID=User1;PWD>Password1;
```

DBAppender Connection Options for Oracle

The following is an example of the basic syntax for specifying options in the `ConnectionString` parameter of a DBAppender configuration for an Oracle database. If you need additional details, contact SAS Technical Support.

Valid in: `ConnectionString` parameter of DBAppender configuration

See: [“DBAppender” on page 46](#)

Syntax

```
CATALOG=catalog-identifier;  
DRIVER=ORACLE;  
PATH=database-specification;  
PWD=password;  
UID=user-id;
```

Syntax Description

The data source connection options for an Oracle database include the following:

CATALOG=*catalog-identifier*;

specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, `catalog=oracle_log`).

Requirement: You must specify a catalog. For the Oracle database, this is a logical catalog name to use as an SQL catalog identifier.

DRIVER=ORACLE;

identifies the data source to which you want to connect, which is an Oracle database.

Requirement: You must specify the driver.

PATH=*database-specification*;

specifies the Oracle connect identifier. A connect identifier can be a net service name, a database service name, or a net service alias.

PWD=*password*;

specifies the Oracle database password that is associated with user ID specified in `UID=`.

Default: If this parameter is not specified, then the password for the default Oracle user ID (OPS\$sysid) is used, if it is enabled.

Requirement: This parameter is required if `UID=` is specified.

UID=*user-id*;

specifies an Oracle user ID. If the user ID contains blanks or national characters, enclose it in quotation marks.

Default: If this parameter is not specified, then the default Oracle user ID (OPS\$sysid) is used, if it is enabled.

Requirement: This parameter is required if `PWD=` is specified.

Example: Connection String for Oracle

```
DRIVER=oracle;UID=User1;PWD>Password1;PATH=MyDatabaseService;CATALOG=x;
```

DBAppender Connection Options for SAS Tables

The following is an example of the basic syntax for specifying options in the `ConnectionString` parameter of a DBAppender configuration if the data store is a SAS table. If you need additional details, contact SAS Technical Support.

Valid in: `ConnectionString` parameter of DBAppender configuration

See: [“DBAppender” on page 46](#)

Syntax

```
CATALOG=catalog-identifier;  
DRIVER=BASE;  
SCHEMA=(NAME=schema-name;PRIMARYPATH=schema-path);
```

Syntax Description

The data source connection options for a SAS table include the following:

CATALOG=*catalog-identifier*;

specifies an arbitrary identifier for the SAS catalog. Any identifier is valid (for example, `catalog=base`).

Requirement: You must specify a catalog.

DRIVER=BASE;

identifies the data source to which you want to connect, which is a Base SAS table.

Requirement: You must specify the driver.

SCHEMA=(NAME=*schema-name*;PRIMARYPATH=*schema-path*);

specifies the name and path for the schema. Any name is valid (for example, `name=MySchema`). For PRIMARYPATH=, specify the path where the SAS table is located, relative to the local machine.

Requirement: You must specify the SCHEMA= option.

Example: Connection String for SAS Tables

```
DRIVER=base;CATALOG=base;schema=(name=MySchema;primarypath='C:\LogsData');
```

DBAppender Connection Options for Teradata

The following is an example of the basic syntax for specifying options in the `ConnectionString` parameter of a DBAppender configuration for a Teradata database. If you need additional details, contact SAS Technical Support.

Valid in: `ConnectionString` parameter of DBAppender configuration

Note: To set up the connection, you must run the Teradata TTU.exe client.

See: [“DBAppender” on page 46](#)

Syntax

```
CATALOG=catalog-identifier;  
DATABASE=database-name;  
DRIVER=TERADATA;  
PASSWORD=password;  
SERVER=Teradata-server-identifier;  
USER=user-id;
```

Syntax Description

The data source connection options for a Teradata database include the following:

CATALOG=*catalog-identifier*;

specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, **catalog=tera**).

Requirement: You must specify a catalog.

DATABASE=*database-name*;

specifies the Teradata database. If this option is not specified, a connection will be made to the default Teradata database for the user ID that is associated with the process. If the database name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks.

DRIVER=TERADATA;

identifies the data source to which you want to connect, which is a Teradata database.

Requirement: You must specify the driver.

PASSWORD=*password*;

specifies a Teradata password. The password that you specify must be correct for your USER= value.

Alias: PWD=

Requirement: You must specify the PASSWORD= option.

SERVER=*Teradata-server-identifier*;

specifies the Teradata server identifier.

USER=*user-id*;

specifies a Teradata user ID. If the user ID contains blanks or national characters, enclose it in quotation marks.

Alias: UID=

Requirement: You must specify the USER= option.

Example: Connection String for Teradata

```
DRIVER=TERADATA;UID=User1;PWD>Password1;server=dbc;database=mydatabase;CATALOG=X;
```

Chapter 8

Pattern Layouts

About Pattern Layouts	95
Overview of Pattern Layouts	95
Conversion Patterns Supplied by SAS	96
Dictionary	104
Syntax for a Pattern Layout	104
c Conversion Character	107
d Conversion Character	108
E Conversion Character	109
F Conversion Character	110
L Conversion Character	110
m Conversion Character	110
n Conversion Character	111
p Conversion Character	111
r Conversion Character	112
severity Conversion Character	112
S Conversion Character	113
sn Conversion Character	115
t Conversion Character	115
u Conversion Character	115
uuid Conversion Character	116
x Conversion Character	116
X Conversion Character	116
%% Conversion Character	117
Format Modifiers	117

About Pattern Layouts

Overview of Pattern Layouts

A pattern layout is a template that you create in order to format log messages for the appender classes in the SAS logging facility. The pattern layout that you define identifies the type of data, the order of the data, and the format of the data that is generated in a log event and that is delivered as output. A unique pattern layout is created for each instance of an appender class. You configure a pattern layout by using the <layout> appender subelement in a logging configuration file or the PATTERN attribute of an appender language element.

The pattern layout is created by using a conversion pattern, which consists of literal text and format-control directives. Format-control directives are also called conversion specifiers.

The conversion patterns that you use to format log messages are similar to, but not identical to, the conversion patterns that are used in the C language PRINTF statement.

Note: The conversion patterns that you use to format log messages are also similar to, but not identical to, these formatting methods that are used in these contexts:

- the directives that are used in the SAS LOGPARM= system option to format log names.
- the set of conversion patterns that are used by the ARMApender. For details, see in *SAS Interface to Application Response Measurement (ARM): Reference*

The meaning of a specific character that is used in a pattern can vary according to the context. Do not interchange characters.

SAS issues an error message if the message layout fails.

Here is an example of a formatted log event that is delivered to the appropriate output device:

Table 8.1 Example of Generated Logged Event

Date	Level	Thread ID	Logger	Filename or Line Number	Message
20008-06-25 10:24:22, 234;	WARN;	3;	Appender.IOM CallContext;	(yn14.sas.c: 149);	Numeric maximum was larger than 8, am setting to 8.

Here is another view of the formatted log event as output:

```
2008-06-25-10:24:22,234; WARN; 3; Appender.IOMCallContext; (yn14.sas.c:149);
Numeric maximum was larger than 8, am setting to 8.
```

Conversion Patterns Supplied by SAS

SAS supplies several conversion patterns that you can specify as the value for HeaderPattern, ConversionPattern, and FooterPattern parameters in appender configurations. You specify a conversion pattern name in place of the conversion pattern.

In this example, DEFAULTHEADER and DEFAULT are conversion pattern names:

```
<appender name="myDefault" class="FileAppender">
  <param name="File" value="default.log"/>
  <param name="ImmediateFlush" value="true"/>
  <param name="Append" value="false"/>
  <layout>
    <param name="HeaderPattern" value='DEFAULTHEADER'/>
    <param name="ConversionPattern" value='DEFAULT'/>
  </layout>
</appender>
```

Here are some rules for using conversion pattern names:

- Conversion pattern names are case sensitive and must be specified in uppercase letters.
- Conversion pattern names cannot be specified with other conversion specifiers. For example, the following ConversionPattern value is not valid:

```
<param name="ConversionPattern" value="%d:DEFAULT"/>
```

SAS supplies DEFAULT and DEFAULTHEADER conversion patterns for each version of SAS as well as a version-specific DEFAULT*version-number* and DEFAULTHEADER*version-number* conversion patterns. If the default layouts change in a new version of SAS, you can use the version-specific conversion patterns to use the previous version's layout. If you have a program to parse logs, you might want to use the version-specific default. If the default conversion pattern changes in a new version of SAS, you do not need to change your program.

The following table lists the conversion patterns that SAS supplies.

Conversion Pattern Name and Description	Conversion Pattern*
DEFAULTHEADER A default conversion pattern for the HeaderPattern value.	Host: '%S{hostname}', OS: '%S{os_family}', Release: '%S{os_release}', SAS Version: '%S{sup_ver_long2}', Command: '%S{startup_cmd}'
DEFAULT A default conversion pattern for the ConversionPattern parameter.	%d %-5p [%t] %u - %m
DEFAULTHEADER9.3 A default conversion pattern for the HeaderPattern parameter in SAS 9.3.	Host: '%S{hostname}', OS: '%S{os_family}', Release: '%S{os_release}', SAS Version: '%S{sup_ver_long2}', Command: '%S{startup_cmd}'
DEFAULT9.3 A default conversion pattern for the ConversionPattern parameter in SAS 9.3.	%d %-5p [%t] %u - %m
TRACE A conversion pattern to use for tracing messages.	%d %-5p [%t] (%F:%L) %c - %u - %m
TRACE9.3 A conversion pattern to use for tracing messages in SAS 9.3.	%d %-5p [%t] (%F:%L) %c - %u - %m
CBE101HEADER A HeaderPattern parameter value for the Common Base Event V1.0.1.	<?xml version="1.0" encoding="UTF-8"?> <CommonBaseEvents xmlns="http://www.ibm.com/AC/commonbaseevent1_0_1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

Conversion Pattern Name and Description	Conversion Pattern*
<p>CBE101</p> <p>A ConversionPattern parameter value for the Common Base Event V1.0.1.</p>	<pre><CommonBaseEvent creationTime="%d{ISO8601ZONEDOT}"%n version="1.0.1"%n severity="%0.1severity{CBE}"%n sequenceNumber="%0.1sn"%n msg="%m"%n xmlns="http://www.ibm.com/AC/commonbaseevent1_0_1"%n xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">%n <contextDataElements name="ServerType" type="uuid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n <sourceComponentId%n component="%S{TKIOM.SERVER_COMPONENT_NAME SAS} #S{sup_ver_long2}, %S{version}"%n subComponent="%c"%n componentIdType="ProductName"%n location="%S{hostname}"%n locationType="Hostname"%n processId="%S{pid}"%n threadId="%0.1t"%n application="%S{App.Name SAS}"%n componentType="Logger"/>%n <situation categoryName="ReportSituation">%n <situationType xsi:type="ReportSituation"%n reasoningScope="INTERNAL"%n reportCategory="LOG"/>%n </situation>%n </CommonBaseEvent></pre>
<p>CBE101FOOTER</p> <p>A FooterPattern parameter value for the Common Base Event V1.0.1.</p>	<pre></CommonBaseEvents></pre>
<p>CBE101CORRELATORSHEADER</p> <p>A Header Pattern parameter value to use with ARM correlators and Common Base Event V1.0.1.</p>	<pre><?xml version="1.0" encoding="UTF-8"?> <CommonBaseEvents xmlns="http://www.ibm.com/AC/commonbaseevent1_0_1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></pre>

Conversion Pattern Name and Description	Conversion Pattern*
<p>CBE101CORRELATORS</p> <p>A ConversionPattern parameter value to use with ARM correlators and Common Base Event V1.0.1.</p>	<pre> <CommonBaseEvent creationTime="%d{ISO8601ZONEDOT}"%n version="1.0.1"%n severity="%0.1severity{CBE}"%n sequenceNumber="%0.1sn"%n msg="%m"%n xmlns="http://www.ibm.com/AC/commonbaseevent1_0_1"%n xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">%n <contextDataElements name="ServerType" type="uuid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n <contextDataElements name="ParentCorrelator" type="string">%n <contextValue>%X{ARM.ParentCorrelator}</contextValue>%n </contextDataElements>%n <contextDataElements name="CurrentCorrelator" type="string">%n <contextValue>%X{ARM.CurrentCorrelator}</contextValue>%n </contextDataElements>%n <sourceComponentId%n component="%S{TKIOM.SERVER_COMPONENT_NAME SAS}%n #S{sup_ver_long2},%S{version}"%n subComponent="%c"%n componentIdType="ProductName"%n location="%S{hostname}"%n locationType="Hostname"%n processId="%S{pid}"%n threadId="%0.1t"%n application="%S{App.Name SAS}"%n componentType="Logger"/>%n <situation categoryName="ReportSituation">%n <situationType xsi:type="ReportSituation"%n reasoningScope="INTERNAL"%n reportCategory="LOG"/>%n </situation>%n </CommonBaseEvent> </pre>
<p>CBE101CORRELATORSFOOTER</p> <p>A FooterPattern parameter value to use with ARM correlators and Common Base Event 1.0.1.</p>	<pre> </CommonBaseEvents> </pre>
<p>CBE111HEADER</p> <p>A HeaterPattern parameter value to use with Common Base Event V1.1.1.</p>	<pre> <?xml version="1.0" encoding="UTF-8"?> <cbe:CommonBaseEvents xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_1"> </pre>

Conversion Pattern Name and Description	Conversion Pattern*
<p>CBE111</p> <p>A ConversionPattern parameter value to use with Common Base Event V1.1.1.</p>	<pre> <cbe:CommonBaseEvent>%n <sourceComponentId%n component="%S{TKIOM.SERVER_COMPONENT_NAME SAS} #S{sup_ver_long2},%S{version}"%n subComponent="%c"%n componentIdType="Logger"%n location="%S{hostname}"%n locationType="Hostname"%n processId="%S{pid}"%n threadId="%0.1t"%n application="%S{App.Name SAS}"/>%n <situationInformation%n creationTime="%d{ISO8601ZONEDOT}"%n severity="%0.1severity{CBE}"%n sequenceNumber="%0.1sn"%n msg="%m">%n <contextDataElements name="ServerType" type="clsid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n <situationType category="REPORT"%n successDisposition="SUCCESSFUL"%n situationQualifier="LOG"%n reasoningScope="INTERNAL"/>%n </situationInformation>%n </cbe:CommonBaseEvent> </pre>
<p>CBE111FOOTER</p> <p>A FooterPattern parameter value to use with Common Base Event V1.1.1.</p>	<pre> </cbe:CommonBaseEvents> </pre>
<p>CBE111CORRELATORSHEADER</p> <p>A HeaderPattern parameter value to use for ARM correlators and Common Base Event V1.1.1.</p>	<pre> <?xml version="1.0" encoding="UTF-8"?> <cbe:CommonBaseEvents x xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_1"> </pre>

Conversion Pattern Name and Description	Conversion Pattern*
<p>CBE111CORRELATORS</p> <p>A ConversionPattern parameter value to use for ARM correlators and Common Base Event V1.1.1.</p>	<pre> <cbe:CommonBaseEvent>%n <sourceComponentId%n component="%S{TKIOM.SERVER_COMPONENT_NAME SAS} #S{sup_ver_long2},%S{version}"%n subComponent="%c"%n componentIdType="Logger"%n location="%S{hostname}"%n locationType="Hostname"%n processId="%S{pid}"%n threadId="%0.lt"%n application="%S{App.Name SAS}"/>%n <situationInformation%n creationTime="%d{ISO8601ZONEDOT}"%n severity="%0.lseverity{CBE}"%n sequenceNumber="%0.lsn"%n msg="%m">%n <contextDataElements name="ServerType" type="clsid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n <contextDataElements name="ParentCorrelator" type="string">%n <contextValue>%X{ARM.ParentCorrelator}</contextValue>%n </contextDataElements>%n <contextDataElements name="CurrentCorrelator" type="string">%n <contextValue>%X{ARM.CurrentCorrelator}</contextValue>%n </contextDataElements>%n <situationType category="REPORT"%n successDisposition="SUCCESSFUL"%n situationQualifier="LOG"%n reasoningScope="INTERNAL"/>%n </situationInformation>%n </cbe:CommonBaseEvent> </pre>
<p>CBE111CORRELATORSFOOTER</p> <p>A FooterPattern parameter value to use for ARM correlators and Common Base Event V1.1.1.</p>	<pre> </cbe:CommonBaseEvents> </pre>
<p>WEFHEADER</p> <p>A HeaderPattern parameter value to use for Web Services Distributed Management (WSDM) Event Format (WEF).</p>	<pre> <?xml version="1.0" encoding="UTF-8"?> <ManagementEvents> </pre>

Conversion Pattern Name and Description	Conversion Pattern*
<p>WEF</p> <p>A ConversionPattern parameter value to use for WSDM Event Format.</p>	<pre> <muws1:ManagementEvent ReportTime="%d{ISO8601ZONEDOT}"%n xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"%n xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd">%n <muws1:EventId>uuid:%uuiid</muws1:EventId>%n <muws1:SourceComponent>%n <Host>%S{hostname}</Host>%n <Pid>%S{pid}</Pid>%n <Logger name="%c" level="%p"/>%n <Component name="%S{TKIOM.SERVER_COMPONENT_NAME SAS} #%S{sup_ver_long2},%S{version}"/>%n </muws1:SourceComponent>%n <muws2:EventCorrelationProperties sequenceNumber="%0.1sn"/>%n <muws2:Situation>%n <muws2:SituationTime>%d{ISO8601ZONEDOT}</muws2:SituationTime>%n <muws2:SituationCategory>%n <muws2:LogReport><muws2:ReportSituation/></muws2:LogReport>%n </muws2:SituationCategory>%n <muws2:Severity>%0.1severity{WEF}</muws2:Severity>%n <muws2:Message>%m</muws2:Message>%n </muws2:Situation>%n <contextDataElements name="ServerType" type="clsid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n </muws1:ManagementEvent> </pre>
<p>WEFFOOTER</p> <p>A FooterPattern parameter value to use for WSDM Event Format.</p>	<pre> </ManagementEvents> </pre>
<p>WEFCORRELATORSHEADER</p> <p>A HeaderPattern parameter value to use for ARM correlators and WSDM Event Format.</p>	<pre> <?xml version="1.0" encoding="UTF-8"?> <ManagementEvents> </pre>

Conversion Pattern Name and Description	Conversion Pattern*
<p>WEFCORRELATORS</p> <p>A ConversionPattern parameter value to use for ARM correlators and WSDM Event Format.</p>	<pre> <muws1:ManagementEvent ReportTime="%d{ISO8601ZONEDOT}"%n xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"%n xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd">%n <muws1:EventId>uuid:%uuuid</muws1:EventId>%n <muws1:SourceComponent>%n <Host>%S{hostname}</Host>%n <Pid>%S{pid}</Pid>%n <Logger name="%c" level="%p"/>%n <Component name="%S{TKIOM.SERVER_COMPONENT_NAME SAS} #S{sup_ver_long2},%S{version}"/>%n </muws1:SourceComponent>%n <muws2:EventCorrelationProperties sequenceNumber="%0.1sn"/>%n <muws2:Situation>%n <muws2:SituationTime>%d{ISO8601ZONEDOT}</muws2:SituationTime>%n <muws2:SituationCategory>%n <muws2:LogReport><muws2:ReportSituation/></muws2:LogReport>%n </muws2:SituationCategory>%n <muws2:Severity>%0.1severity{WEF}</muws2:Severity>%n <muws2:Message>%m</muws2:Message>%n </muws2:Situation>%n <contextDataElements name="ServerType" type="clsid">%n <contextValue>%S{TKIOM.BASE_CLSID}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerPort" type="number">%n <contextValue>%S{TKIOM.PORT}</contextValue>%n </contextDataElements>%n <contextDataElements name="ServerKnownBy" type="string">%n <contextValue>%S{TKIOM.KNOWN_BY}</contextValue>%n </contextDataElements>%n <contextDataElements name="UserId" type="string">%n <contextValue>%u</contextValue>%n </contextDataElements>%n <contextDataElements name="ParentCorrelator" type="string">%n <contextValue>%X{ARM.ParentCorrelator}</contextValue>%n </contextDataElements>%n <contextDataElements name="CurrentCorrelator" type="string">%n <contextValue>%X{ARM.CurrentCorrelator}</contextValue>%n </contextDataElements>%n </muws1:ManagementEvent> </pre>
<p>WEFCORRELATORSFOOTER</p> <p>A FooterPattern parameter value to use for ARM correlators and WSDM Event Format.</p>	<pre> </ManagementEvents> </pre>

* **Operating Environment Information for z/OS:** When you specify %uuuid or WEF in a logging facility configuration file, the identifiers that SAS creates are unique only for the current computer node. %uuuid or WEF does not create a UUID. You can obtain a computer node name by using the %S{hostname} conversion character.

Dictionary

Syntax for a Pattern Layout

Note: A pattern layout is configured by using the `<layout>` and `</layout>` elements. Any `<appender>` element attributes and subelements in the syntax are present to show the context for the pattern layout elements.

Syntax

XML Configuration

```
<appender class="AppenderName" name="appender-reference-name">
  <layout>
    <param name="Header" value="header-text"/>
    <param name="HeaderPattern" value="conversion-pattern"/>
    <param name="ConversionPattern" value="conversion-pattern"/>
    <param name="Footer" value="footer-text"/>
    <param name="FooterPattern" value="conversion-pattern"/>
    <param name="XmlEscape" value="TRUE | FALSE"/>
  </layout>
</appender>
```

Syntax Description

class= "AppenderName" name="appender-reference-name"

specifies the name of the appender reference that is to be used as an appender-ref for a logger.

See: For appender class names, see [Chapter 6, “Appender Reference,”](#) on page 41.

name="Header" value="header-text"

specifies the header text that the appender uses when it starts a new log.

Valid in: XML configuration files for the File, RollingFile, and Console appender classes.

name="HeaderPattern" value="conversion-pattern"

specifies the pattern layout that is used to identify, order, and format information in a header. A conversion pattern consists of optional literal text and optional format-control directives, which are called *conversion specifiers*. Each conversion specifier begins with a percent sign (%) and is followed by optional *format modifiers* and one or more instances of the S conversion character or the d conversion character. The format modifiers control field width, padding, and left and right justification.

Here is the syntax for a header pattern:

```
[literal-text] %[format-modifier-1] conversion-character-1
[... [literal-text] %[format-modifier-n] conversion-character-n] />
```

Valid in: XML configuration files for the File, RollingFile, and Console appender classes.

Restriction: The header pattern is limited to the S and d conversion characters in a logging configuration file.

Interaction: The Header parameter *header-text* takes precedence over the HeaderPattern *conversion-pattern* parameter.

Tips:

The specification of format modifiers is optional.

There is no explicit separator between literal text and the conversion specifier.

The pattern parser recognizes the end of a conversion specifier when it detects the S or d conversion character.

See:

[“d Conversion Character” on page 108](#)

[“S Conversion Character” on page 113](#)

[“Format Modifiers” on page 117](#)

Example:

```
<param name="HeaderPattern" value="%d %S{os_name} %S{jobid} %S{host_name}
    %S{user_name}"/>
```

name="ConversionPattern" value="conversion-pattern"

specifies the pattern layout that is used to identify, order, and format information in the log event. A conversion pattern consists of literal text and format-control directives called *conversion specifiers*. Each conversion specifier begins with a percent sign (%) and is followed by optional *format modifiers* and a *conversion character*. The conversion character specifies the type of data (for example, category, priority, date, and thread name). The format modifiers control field width, padding, and left and right justification.

Here is the syntax for a conversion pattern:

```
[literal-text] %[format-modifier-1] conversion-character-1
[... [literal-text] %[format-modifier-n] conversion-character-n] />
```

Valid in: XML configuration files for any appender class.

Default: None. The log event produces an empty string if a conversion pattern is not specified.

Restrictions:

Conversion specifiers are case sensitive.

Do not use these problematic characters, known as variants, in pattern layouts in EBCDIC encoding environments: ! # \$ @ \ [] ^ ~ { } | ~ \n. \n represents the use of the New Line (or Enter) key

These characters are problematic because they might occupy different code positions in various encodings that are supported by SAS. For example, the EBCDIC code point location 5A (hexadecimal) represents the exclamation point (!) in U.S. English and the right bracket (]) in Spanish.

Tips:

The specification of format modifiers is optional.

There is no explicit separator between literal text and a conversion specifier. The pattern parser recognizes the end of a conversion specifier when it detects a conversion character.

See:

[“S Conversion Character” on page 113](#)

[“Format Modifiers” on page 117](#)

For more information about variant characters, see the *SAS National Language Support (NLS): Reference Guide*.

Example: `<param name="ConversionPattern" value="%d; %-5p; %t; %c; (%F: %L); %m"/>`

name="Footer" value="footer-text"

specifies the footer text that the appender uses when it ends a log.

Valid in: XML configuration files for the File, RollingFile, and Console appender classes.

name="FooterPattern" value="conversion-pattern"

specifies the pattern layout that is used to identify, order, and format information in a footer. A conversion pattern consists of optional literal text and optional format-control directives, which are called *conversion specifiers*. Each conversion specifier begins with a percent sign (%) and is followed by optional format modifiers and one or more instances of the S or d conversion characters. The format modifiers control field width, padding, and left and right justification.

Here is the syntax for a footer pattern:

```
[literal-text] %[format-modifier-1] conversion-character-1
[... [literal-text] %[format-modifier-n] conversion-character-n] />
```

Valid in: XML configuration files for the File, RollingFile, and Console appender classes.

Restriction: The footer pattern is limited to the S and d conversion characters in a logging configuration file.

Interaction: The Footer parameter *footer-text* takes precedence over the FooterPattern *conversion-pattern* parameter.

Tip: There is no explicit separator between literal text and a conversion specifier. The pattern parser recognizes the end of a conversion specifier when it detects a conversion character.

See:

“d Conversion Character” on page 108 “S Conversion Character” on page 113
“Format Modifiers” on page 117

Example: `<param name="FooterPattern" value="%d %S {host_name}"/>`

name="XmlEscape" value="TRUE | FALSE"

specifies whether certain characters that can be specified in the m, x, X, and S conversion specifiers are converted to their entity representations.

TRUE

specifies that the following characters are converted when they are used in the m, x, X, and S conversion specifiers:

- "<" is converted to "<"
- ">" is converted to ">"
- "\"" (double quotation marks) is converted to """
- "'" (single quotation mark) is converted to "'"
- "&" is converted to "&"

FALSE

specifies that no character conversion to entity representations is performed.

Example: Example of a Pattern Layout

Here is an excerpt of an XML file that contains a pattern layout:

```
<layout>
  <param name="ConversionPattern" value="%d %-5p %t %c{2} (%F:%L) %m"/>
</layout>
```

Here is an explanation:

Table 8.2 Example of a Layout Pattern

Pattern Layout	Explanation	Example
%d	reports the date of the log event and formats the date using the default format, ISO8601.	2011-06-25 10:24:22,234
%-5p	reports the level of the log event and left-justifies the level in output. If the level occupies fewer than five characters, the level is padded on the right.	WARN
%t	reports the identifier of the thread that generated the log event.	0000000
%c{2}	reports the name of the logger that generated the log event. The precision specifier limits the logger name to two subfields, causing left-truncation.	The full logger name is Log4SAS.Appender.IOMCallContext. The formatted output is Appender.IOMCallContext;.
(%F:%L)	reports the filename and the line number in the application that generated the log event. The parentheses and colon are literal text that was specified in the conversion pattern.	(ynl4sas.c:149) The parentheses () and colon (:) are literal text.
%m	reports the message that is supplied by the application and that is associated with the log event.	Numeric maximum was larger than 8, am setting to 8.

c Conversion Character

Reports the name of the logger that generates the log event.

Alias: logger

Default: Complete logger name; for example, "Logging.Appender.IOMCallContext".

Syntax

c [{precision-specifier}]

Optional Argument

precision-specifier

specifies a decimal constant to indicate the number of logger levels to display in the message. If you specify *precision-specifier*, only the corresponding number of right-most components of the logger name are included in the output.

Requirement: If the precision specifier is used, it must be enclosed in a pair of braces.

Interaction: If the precision specifier is used, only the corresponding number of right-most components of the logger name are included in the output.

Example

For the logger name "Logging.Appender.IOMCallContext", the pattern `%c{2}` generates this output:

```
"Appender.IOMCallContext"
```

d Conversion Character

Reports the date of the log event.

Alias: date

Default: ISO8601 format, which is represented as *yyyy-MM-dd HH:mm:ss,SSS*

Requirement: If the date conversion specifier is used, it must be enclosed in a pair of braces.

Tip: The d conversion character formats three digits for the precision of milliseconds, regardless of the number of S simple date format characters that are specified, and regardless of the machine precision of the timing that is available.

Syntax

```
d [ {date-conversion-specifier} ]
```

Optional Argument

date-conversion-specifier

specifies the format of the date. Here are the supported *date conversion specifiers*:

ABSOLUTE

specifies the time in the format *HH:mm:ss,SSS*.

Example: 15:49:37,459

DATE

specifies the date and time in the format *dd MMM yyyy HH:mm:ss,SSS*.

Example: 06 Nov 2010 15:49:37,459

ISO8601

specifies the date and time in the format *yyyy-MM-dd HH:mm:ss,SSS*. An example is

Example: 2011-11-27 15:49:37,459

Simple Date Format

specifies a date in the form of a string that can contain any of these sets of characters:

aa: a.m. or p.m. marker (string)	FF: Day of week in month (numeric)
dd: Day in month (numeric)	GG: Era designator (string "AD")
hh: Hour in a.m. or p.m. (numeric 1-12)	HH: Hour in day (numeric 0-23)
mm: Minute in hour (numeric)	KK: Hour in a.m. or p.m. (numeric 0-11)
ss: Second in minute (numeric)	MM: Month in year (numeric 1-12)
yy: Two-digit year (numeric)	MMM: Month in year (abbreviated string)
yyyy: Four-digit year (numeric)	MMMM: Month in year (string)
z: Time zone (string)	SSS: Millisecond (numeric)
DD: Day in year (numeric)	Z: RFC 822 Time Zone (string)
EE: Day in week (abbreviated string)	<i>'literal string within single quotation marks '</i>
EEEE: Day in week (string)	

Example

Here are examples of the d conversion character:

```
d{ABSOLUTE}
```

```
d{EEEE MMMM yyyy HH:mm:ss,SSS 'Ship date'}
```

E Conversion Character

Reports library access audit information.

Syntax

```
E{key}
```

Optional Arguments

key can be one of the following:

Audit.Dataset.Libref

reports the libref that was used to access the library.

Audit.Dataset.Engine

reports the engine that is associated with the library.

Audit.Dataset.Member

reports the name of the data set member.

Audit.Dataset.Memtype

reports the type of data set member.

Audit.Dataset.Openmode

reports whether the data set is open in READ, WRITE, or UPDATE mode.

Audit.Dataset.Path

reports the path to the library.

Example

Here is an example of the E conversion character:

```
libref=%E{Audit.Dataset.Libref}
```

See Also

[Appendix 1, “Audit Messages for SAS Library Access,” on page 183](#)

F Conversion Character

Reports the name of the file in the application that generated the log event.

Alias: file

Syntax

F

L Conversion Character

Reports the line number in the application that generated the log event.

Alias: line

Syntax

L

m Conversion Character

Writes the messages that are associated with the log event.

Alias: message

Default: None

Requirement: If the prefix identifier is used, it must be closed in a pair of braces.

Syntax

m [{*prefix-identifier*}]

Required Argument

prefix-identifier

specifies an optional identifier to precede all lines after the first line. The following are valid values for *prefix-identifier*:

HYPHEN inserts a hyphen (-) before each message.

PLUS inserts a plus sign (+) before each message.

Details

When the message is more than one line long, all lines are written. When a prefix identifier is specified, all lines after the first line are preceded by the prefix identifier.

n Conversion Character

Enables you to supply discretionary newline characters among the data items that compose the log event.

Default: None

Requirement: If the prefix identifier is used, it must be enclosed in a pair of braces.

Syntax

n [{*newline-prefix-identifier*}]

Optional Argument

newline-prefix-identifier

specifies an optional newline identifier. The following are valid values for *newline-prefix-identifier*:

HYPHEN inserts a hyphen (-) before each message.

PLUS inserts a plus sign (+) before each message.

p Conversion Character

reports the level of the log event.

Alias: level

Syntax

p

Details

Here are the supported levels:

- TRACE
- DEBUG

- INFO
- WARN
- ERROR
- FATAL

r Conversion Character

Reports the number of milliseconds that elapsed between the start of the application and the creation of the log event.

Alias: relative

Syntax

r

severity Conversion Character

Translates error levels to Common Base Event (CBE) and Web Services Distributed Management Event Format (WEF) severity codes.

Syntax

severity {*key*}

Required Arguments

key can be one of the following:

CBE

reports a CBE severity code. The following table correlates levels with CBE severity codes:

Logging Facility Level	CBE Severity Code
FATAL	60
ERROR	50
WARN	30
INFO	10
DEBUG	5
TRACE	0

WEF

reports a WEF severity code. The following table correlates levels with WEF severity codes:

Logging Facility Level	WEF Severity Code
FATAL	6
ERROR	4
WARN	2
INFO	1
DEBUG	0
TRACE	0

WEF has two additional codes that do not correlate to the logging facility levels:

- 5 is a critical condition.
- 3 is a minor problem of relatively low severity.

S Conversion Character

Delivers various system information to the log event.

Alias: systemInfo

Requirements: The S conversion character must be followed by the specified value, which is also referred to as a key, and must be enclosed in a pair of braces. If you specify a default value, you must specify *key* and *default*, separated by “ | “ (the vertical bar), enclosed in a pair of braces.

Syntax

S {*key*}

S {*key* | *default*}

Required Arguments

default

specifies the value that is used when the information that is specified by *key* cannot be found. The value of *default* is a character string that appears between | (the vertical bar) and)(the closing bracket).

Note: If the character string contains quotation marks, the quotation marks become part of the value of *default*. Quotation marks in an XML attribute must be specified using the character entity ".

Example: %S{App.Log|Spawner.log}

key

key can be one of the following types of information:

App.Log

reports the filename that is specified by the LOG= system option when SAS starts. Otherwise, these actions occur:

- If the LOG= system option does not specify a filename, but a filename is specified by the SYSIN= system option, the filename that is specified by the SYSIN= option is used. The file extension is changed to .log.
- If the filename that is specified by the SYSIN= system option lacks a full pathname, the path of the current working directory is prepended to the filename.

App.Name

reports the value of the LOGAPPLNAME= system option.

App.Sysin

reports the filename that is specified by the SYSIN= system option.

model_name

reports the name of the manufacturer of the computer hardware. Examples are HP, SUN, and IBM.

model_num

reports the model number of the computer hardware. Examples are Itanium, X86, RS/6000, SPARC, and 9000/800.

host_name | hostname

reports the node name that is assigned to the computer hardware. An example is apex.com.

serial

reports the serial number of the operating system.

os_name

reports the name of the operating system. Examples are LINUX, HP-UX, SUNOS, and XP_HOME.

os_version

reports the version of the operating system.

os_release

reports the release number of the operating system. Examples are Linux2.6, Linux 5, Linux 9, and Linux 11.22.

os_family

reports the family of operating system. Examples are LINUX ITANIUM, LINUX, SUN 64, HP IPF, and WIN.

jobid | pid

reports the job ID or the process ID, as appropriate.

user_name | username

reports the user name in the appropriate form.

Note: The user_name is the identity that owns the process rather than the client identity that is associated with the current thread.

See: [“u Conversion Character” on page 115](#)

startup_cmd

reports the arguments that are specified when the application was started.

version

reports either of these versions: TK_BASE_MAJOR or TK_BASE_MINOR.

system_desc

reports a description of the hardware and software environment. Examples are X86_64 Linux, HP Itanium Processor Family, and Sun Sparc 64-bit.

build_date

reports the date on which the kernel for threaded processing was built.

build_time

reports the time at which the kernel for threaded processing was built.

sup_ver

reports the version number of the SAS supervisor.

sup_ver_long2

reports the version number of the SAS supervisor that is Y2K compliant.

Example

Here is an example of the S conversion character:

```
%S{os_family}
```

sn Conversion Character

Reports the sequence number of the log event.

Alias: sequenceNumber

Syntax

sn

t Conversion Character

Reports the identifier of the thread that generated the log event.

Alias: thread

Syntax

t

u Conversion Character

Reports the client identity that is associated with the current thread or task.

Alias: username

See: [user_name](#) | [username on page 114](#) in the S conversion character.

Syntax

u

Details

If the current thread or task does not have an associated identity, the identity that owns the current process is reported to the log event.

uuid Conversion Character

Reports the universal unique identifier (UUID) for the log event.

z/OS specifics: Identifiers created by the uuid conversion character on z/OS are unique only to the current computer node. If this conversion pattern is encountered on z/OS, a warning is written to the log.

Syntax

uuid

x Conversion Character

Reports the NDC (nested diagnostic context) that is associated with the thread that generated the log event.

Alias: ndc

Syntax

x

X Conversion Character

Reports the MDC (mapped diagnostic context) that is associated with the thread that generated the log event.

Alias: properties

Requirements: The key must be enclosed in a pair of braces. If you specify a default value, you must specify *key* and *default*, separated by “|” (the vertical bar), enclosed in a pair of braces.

Syntax

X {*key*}

X {*key* | *default*}

Required Arguments

default

specifies the value that is used when the information that is specified by *key* cannot be found. The value of *default* is a character string that appears between | (the vertical bar) and } (the closing bracket).

Note: If the character string contains quotation marks, the quotation marks become part of the value of *default*. Quotation marks in an XML attribute must be specified using the character entity `"`;

Example: `%X{clientNumber}|clientNumberNotFound}`

key

specifies a parameter that is used to identify the portion of the log to retrieve using MDC.

Details

MDC is used to distinguish interleaved log output from different sources. Log output is typically interleaved when a server manages multiple clients in parallel. The MDC is managed on a per-thread basis.

The X conversion character must be followed by the key for the map. The value in the MDC that corresponds to the key is reported.

Example

Here is an example of the X conversion character:

`%X{clientNumber}`, where `clientNumber` is the key.

%% Conversion Character

Enables you to specify a literal percent sign symbol in a text string of a conversion pattern.

Syntax

`%%`

Details

A single percent sign is interpreted as a conversion specifier. Two percent signs are interpreted as literal text, which is delivered as a single percent sign in the log event.

Example

Here is an example of the %% conversion character:

```
<param name="ConversionPattern" value="%d;text%%text;%m"/>
```

Here is sample output:

```
2011-06-25-10:24:22,234; text;text;Numeric maximum was larger than 8, am setting to 8
```

Format Modifiers

Controls the field width, padding, and justification of the specified data item in log output.

Syntax

- (hyphen)

minimum-field-width-modifier

maximum-field-width-modifier

Format Modifier Descriptions

- (hyphen)

specifies left-justification of the data item that is defined by the conversion character.

Examples:

`%-p`

The `p` conversion character reports the level that is specified by the log event. For example, the text of the level, "WARN" is left-justified within its field in the log event.

minimum-field-width-modifier

specifies a decimal constant to indicate the minimum width of the field for the data item that is specified by the conversion character. If the data item is smaller than the minimum field width, the field is padded on either the left or the right until the minimum width is reached. The padding character is a space. If the data item exceeds the minimum field width, the field is expanded to accommodate the data item.

Default: Pad on the right (left-justify)

Examples:

`%10p`

The constant value, 10, provides a minimum width for the data item that is specified by the `p` conversion character. For example, the text of the level, "WARN", is left-justified and is padded to the right with six spaces.

maximum-field-width-modifier

specifies a period (.) and a decimal constant to indicate the maximum width of the field for the data item that is specified by the conversion character.

Default: If the data item exceeds the maximum field width, characters are left-truncated rather than right-truncated.

Restriction: The behaviors of *maximum-field-width-modifier* in the SAS logging facility and in the C language PRINTF statement are different. The PRINTF statement uses right-truncation rather than left-truncation.

Examples:

`%.3p` is the pattern layout. "DEBUG" is the data item. "BUG" is the generated output.

The constant value, 3, provides a maximum width for the data item that is specified by the `p` conversion character. For example, the text of the level, "DEBUG" is left-truncated to form "BUG".

Example

Here are examples of format modifiers that are used with the `c` conversion character. The `c` conversion character reports the name of the logger.

Table 8.3 Examples of Format Modifiers

Format Modifier	Left Justification	Minimum Width	Maximum Width	Explanation
%20c	no	20	none	If the data item occupies fewer than 20 characters, pad to the left, using spaces.
%-20c	yes	20	none	If the data item occupies fewer than 20 characters, pad to the right, using spaces.
%.30c	not applicable	none	30	If the data item exceeds 30 characters, left-truncate the data item.
%20.30c	no	20	30	If the data item occupies fewer than 20 characters, pad to the left, using spaces. If the data item exceeds 30 characters, left-truncate the data item.
%-20.30c	yes	20	30	If the data item occupies fewer than 20 characters, pad to the right, using spaces. If the data item exceeds 30 characters, left-truncate the data item.

Chapter 9

Filters

Overview of Filters	121
Filter Examples	123
Example 1: Filter for a Specific User's Error Messages	123
Example 2: Filter for a Specific Date	123
Dictionary	124
Syntax for Filters	124
AndFilter	124
DenyAllFilter	126
LevelMatchFilter	126
LevelRangeFilter	127
RepeatMatchFilter	129
StringMatchFilter	129

Overview of Filters

In addition to filtering log events by thresholds, using logger and appender configurations, the logging facility has filter classes to filter log events, based on character strings and thresholds:

Filter Class Name	Description
RepeatMatchFilter	filters repeated log messages.
StringMatchFilter	filters log messages based on a character string in the log message.
LevelRangeFilter	filters log messages based on a range of message thresholds.
LevelMatchFilter	filters log messages based on a single message threshold.
AndFilter	filters log messages based on the results of a list of other filters.
DenyAllFilter	denies log events that did not meet the criteria of previous filters in a filter policy.

By using filter classes to filter messages, you can choose whether to accept or deny a log event if a match occurs between a filter parameter and the string or threshold in the log event.

You configure filter classes by using the <filter> subelements within an appender configuration.

Filters are processed in the order in which they appear in the appender definition, creating a filtering policy for the appender.

The results of filtering depend on filter arguments. The `AcceptOnMatch` argument in the `RepeatMatchFilter`, `StringMatchFilter`, `LevelMatchFilter`, and `LevelRangeFilter` filters indicate whether to accept the log event if there is a match. The following lists describe the process of deciding whether a log event is accepted or denied:

- `RepeatMatchFilter`
 - If the immediate previous message (%m) is the same as the message (%m) specified in the log event and if `AcceptOnMatch` is `TRUE`, then the appender processes the log event.
 - If the previous message (%m) is the same as the message (%m) specified in the log event and if `AcceptOnMatch` is `FALSE`, then the appender denies the log event.
- `StringMatchFilter` and `LevelMatchFilter`
 - If there is a match between the filter string or the filter threshold (level) and the log event, and if `AcceptOnMatch` is `TRUE`, then the appender processes the log event.
 - If there is a match between the filter string or the filter threshold (level) and the log event, and if `AcceptOnMatch` is `FALSE`, then the appender denies the log event.
 - If there is no match between the filter and the log event, then the appender processes the next filter in the filtering policy. If the log event has not been denied and if there are no other filters in the filtering policy, then the appender processes the log event.
- `LevelRangeFilter`
 - If there is a match between the minimum and maximum thresholds (inclusive) in the filter and the log event, and if `AcceptOnMatch` is `TRUE`, the appender processes the log event.
 - If there is no match, the appender denies the log event.
 - If there is a match between the minimum and maximum thresholds (inclusive) in the filter and the log event, and if `AcceptOnMatch` is `FALSE`, then the appender processes the next filter in the filtering policy. If the log event has not been denied and if there are no other filters in the filtering policy, the appender accepts and processes the log event.
- `AndFilter` uses `StringMatchFilter`, `LevelMatchFilter`, and `LevelRangeFilter` as arguments. The results of these filters as arguments to the `AndFilter` class is the same as it is in the individual filters.

You can include `DenyAllFilter` as the last filter in the filtering policy to deny any log events that do not meet the filtering policy for the appender.

The following example is a simple filtering policy to log only performance messages for the ARM subsystem:

```

<filter class="StringMatchFilter">
  <param name="StringToMatch" value="Perf.ARM"/>
  <param name="AcceptOnMatch" value="true"/>
</filter>
<filter class="DenyAllFilter">
</filter>

```

Note: Filter definitions are not available in the logging facility language elements for SAS programs.

Filter Examples

Example 1: Filter for a Specific User's Error Messages

In this example, the filtering policy first checks to determine whether the message has already been logged. The filtering policy then writes to the Windows Event Log the messages whose log event threshold is ERROR and which are issued by user sasuser1:

```

<?xml version="1.0" encoding="UTF-8"?>
  <logging:configuration xmlns:
logging="http://www.sas.com/xml/logging/1.0/">
    <appender name="eventLog" class="WindowsEventAppender">
      <param name="AppName" value="SAS Foundation"/>
      <layout>
        <param name="ConversionPattern"
          value="%d %-5p [%t] %c (%F:%L) %u - %m"/>
      </layout>
      <filter class="RepeatMatchFilter">
        <param name="AcceptOnMatch" value="false"/>
      </filter>
      <filter class="AndFilter">
        <param name="AcceptOnMatch" value="true"/>
        <filter class="LevelMatchFilter">
          <param name="LevelToMatch" value="error"/>
          <param name="AcceptOnMatch" value="true"/>
        </filter>
        <filter class="StringMatchFilter">
          <param name="StringToMatch" value="sasuser1"/>
          <param name="AcceptOnMatch" value="true"/>
        </filter>
      </filter>
      <filter class="DenyAllFilter">
      </filter>
    </appender>
  <root>
    <level value="trace"/>
    <appender-ref ref="eventLog"/>
  </root>
</logging:configuration>

```

Example 2: Filter for a Specific Date

The following filtering policy denies log events that were sent on 2011-09-22:

```

<filter class="StringMatchFilter">
  <param name="StringToMatch" value="2011-09-22"/>
  <param name="AcceptOnMatch" value="false"/>
</filter>

```

Dictionary

Syntax for Filters

Note: Filters are configured, using the `<filter>` and `</filter>` elements and their respective filter parameters. Any `<appender>` element attributes and subelements in the syntax are present to show the context for the pattern layout elements. See the syntax for each filter for the parameters that are used by that filter.

Syntax

XML Configuration

```

<appender class="AppenderName" name="log-name">
  <filter class="filter-class">
    <param name="filter-parameter-1" value="parameter-value-1"/>
    <param name="filter-paramter-n" value="parameter-value"/>
  </filter>
</appender>

```

AndFilter

Use when you want to log messages that meet multiple criteria.

Syntax

```

<filter class="AndFilter">
  <param name="AcceptOnMatch" value="TRUE | FALSE">
  <filter class="filter-name">
    <param name="filter-parameter" value="filter-parameter-value"/>
    <param name="AcceptOnMatch" value="TRUE | FALSE"/>
  </filter/>-1
  [... <filter class="filter-name">
    <param name="filter-parameter-name" value="filter-parameter-value"/>
    <param name="AcceptOnMatch" value="TRUE | FALSE"/>
  </filter/>-n ]
</filter>

```

Syntax Description

class="AndFilter"

specifies to apply the AND logical operation on the subfilter results to determine whether the log event is accepted by the appender.

name="AcceptOnMatch" value="TRUE | FALSE"

for AndFilter, specifies whether to accept or deny the log event if the result of the logical AND is TRUE. For subfilter definitions, specifies whether to accept or deny the log event if the threshold or string matches. Valid values are TRUE or FALSE.

TRUE

specifies to accept the log event.

FALSE

for AndFilter, StringMatchFilter, and LevelMatchFilter, specifies to deny the log event.

class="filter-name"

specifies the name of a filter to use as an argument to the AND logical operation. Here is a list of valid filters:

- AndFilter
- LevelMatchFilter
- LevelRangeFilter
- StringMatchFilter

name="filter-parameter-name" value="filter-parameter-value"

specifies the name of a filter parameter and the parameter value that is used to compare with either the log event threshold or the message. The following table shows the filter parameters:

Filter Name	Filter Parameter Name	Filter Parameter Value
LevelMatchFilter	LevelToMatch	DEBUG TRACE INFO WARN ERROR FATAL
LevelRangeFilter	LevelMax LevelMin	DEBUG TRACE INFO WARN ERROR FATAL
StringMatchFilter	StringToMatch	a character string enclosed in quotation marks
AndFilter	AcceptOnMatch	TRUE FALSE

In addition to the AcceptOnMatch parameter, specify two or more filters as arguments to a nested AndFilter.

Details

AndFilter syntax allows for two or more subfilter definitions within the AndFilter definition. The subfilters are evaluated to decide whether to accept or deny the log event. AndFilters performs a logical AND operation, using the results of each subfilter evaluation to determine the results of AndFilter.

An example of using `AndFilter` might be that you want to filter log messages that have a threshold of `INFO` and that contain the string "New client connection".

You can filter by a single threshold, a range of thresholds, or by string matching.

Example

The following filter accepts log events that have a threshold of `INFO` and the string `RETURN` in the following log message:

```
<filter class="AndFilter">
  <param name="AcceptOnMatch" value="true"/>
  <filter class="LevelMatchFilter">
    <param name="LevelToMatch" value="info"/>
    <param name="AcceptOnMatch" value="true"/>
  </filter>
  <filter class="StringMatchFilter">
    <param name="StringToMatch" value="RETURN"/>
    <param name="AcceptOnMatch" value="true"/>
  </filter>
</filter>
<filter class="DenyAllFilter">
</filter>
```

DenyAllFilter

`DenyAllFilter` can be configured as the last filter in a chain of filters to deny all messages that do not meet the filter specifications in the filter chain.

Syntax

```
<filter class="DenyAllFilter">
```

Syntax Description

class="DenyAllFilter"

specifies to deny all messages that do not meet the filter chain criteria.

TIP Use `DenyAllFilter` as the last filter if you use `AcceptOnMatch="TRUE"` and you want only the messages that match to be processed.

LevelMatchFilter

Use `LevelMatchFilter` when you want to filter log events for a single message threshold. For example, you might want to log only error messages, or you might want all messages that do not have a threshold of `FATAL`.

Syntax

```
<filter class="LevelMatchFilter">
  <param name="LevelToMatch" value="DEBUG | TRACE | INFO | WARN | ERROR | FATAL"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

Syntax Description

class="LevelMatchFilter"

specifies to filter messages based on a log event threshold.

name="AcceptOnMatch" value="TRUE | FALSE"

specifies whether to accept or deny the log event if the log event threshold matches the value in this filter. Valid values are TRUE or FALSE:

TRUE

specifies to accept the log event.

FALSE

specifies to deny the log event.

name="LevelToMatch" value="DEBUG | TRACE | INFO | WARN | ERROR | FATAL"

specifies the threshold to filter log events for this appender. Valid values are DEBUG, TRACE, INFO, WARN, ERROR, or FATAL.

See: [“Logging Thresholds” on page 16](#)

Details

To use this filter you specify a threshold, and you specify whether to accept or deny the log event if the filter threshold matches the log event threshold. If there is no match, the filtering process continues with the next filter in the filtering policy. If there are no other filters in the filtering policy and if the log event has not been denied, the appender accepts and processes the log event.

Example

The following filter denies log events whose threshold is INFO:

```
<filter class="LevelMatchFilter">
  <param name="LevelToMatch" value="info"/>
  <param name="AcceptOnMatch" value="false"/>
</filter>
```

LevelRangeFilter

Use LevelRangeFilter when you want to filter log event messages whose message threshold falls within a range of message thresholds.

Syntax

```
<filter class="LevelRangeFilter">
  <param name="LevelMax" value="threshold"/>
  <param name="LevelMin" value="threshold"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

Syntax Description

class="LevelRangeFilter"

specifies to use the LevelRangeFilter

name="LevelMax" value="threshold"

specifies the highest threshold that can be written to the appender.

name="LevelMin" value="threshold"

specifies the lowest threshold that can be written to the appender.

name="AcceptOnMatch" value="TRUE | FALSE"

specifies whether to accept the log event when the log event message threshold falls within the threshold range that is specified by the filter. Valid values are TRUE or FALSE:

TRUE

specifies to accept the log event.

FALSE

specifies to pass the filtering process to the next filter in the filtering policy. If the log event has not been denied and there are no other filters in the filtering policy, the appender accepts and processes the log event.

Details

The thresholds are, from lowest to highest: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. For example, if the minimum threshold is DEBUG and the maximum threshold is ERROR, and if AcceptOnMatch is FALSE, messages that have the thresholds TRACE and FATAL are denied.

To use this filter you specify a minimum and a maximum threshold range to compare with the log event threshold. If there is no match, the log event is denied. If there is a match and if AcceptOnMatch is TRUE, the appender accepts and processes the log event. If there is a match and AcceptOnMatch is FALSE, the next filter in the filtering policy is processed. If there are no other filters in the filtering policy and if the log event has not been denied, the appender accepts and processes the log event.

Example

The following filter accepts log events only if the log event threshold is between WARN and ERROR:

```
<filter class="LevelRangeFilter">
  <param name="LevelMax" value="error"/>
  <param name="LevelMin" value="warn"/>
  <param name="AcceptOnMatch" value="true"/>
</filter>
```

RepeatMatchFilter

Use RepeatMatchFilter to discard a message if the message has already been logged by the appender.

Syntax

```
<filter class="RepeatMatchFilter">
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

Syntax Description

name="AcceptOnMatch" value="TRUE | FALSE"

specifies whether to accept or deny the log event when the log event message is identical to the last message processed by the appender. Valid values are TRUE or FALSE:

TRUE
specifies to accept the log event.

FALSE
specifies to deny the log event.

Details

This filter compares only the last message processed by this appender with the current log event message.

As a best practice, you can use this filter as the first filter in appender filtering policies to discard repeated messages.

Example

The following filter definition does not accept log events if a log event message has already been logged by this appender:

```
<filter class="RepeatMatchFilter">
  <param name="AcceptOnMatch" value="FALSE"/>
</filter>
```

StringMatchFilter

Use StringMatchFilter when you want to filter messages based on a string in the log event message.

Syntax

```
<filter class="StringMatchFilter">
  <param name="StringToMatch" value="character-string"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

Syntax Description

name="StringToMatch" value="character-string"

specifies the string to search for in the log event message.

name="AcceptOnMatch" value="TRUE | FALSE"

specifies whether to accept or deny the log event when the log event message contains *character-string*. Valid values are TRUE or FALSE:

TRUE

specifies to accept the log event.

FALSE

specifies to deny the log event.

Details

To use this filter you specify a character string, and you specify whether to accept or deny the log event if the filter character string matches a character string in the log event message. If there is no match, the filtering process continues with the next filter in the filtering policy. If there are no other filters in the filtering policy and if the log event has not been denied, the appender accepts and processes the log event.

Example

The following filter definition does not accept log events that contain the string "RETURN":

```
<filter class="StringMatchFilter">
  <param name="StringToMatch" value="RETURN"/>
  <param name="AcceptOnMatch" value="true"/>
</filter>
```

Part 3

The Logging Facility for SAS Programs

<i>Chapter 10</i>	
The SAS Logging Facility in the SAS Language	133
<i>Chapter 11</i>	
Autocall Macro Reference	141
<i>Chapter 12</i>	
Function Reference	155
<i>Chapter 13</i>	
Component Object Reference	165

Chapter 10

The SAS Logging Facility in the SAS Language

Overview of the SAS Logging Facility in the SAS Language	133
Initializing the SAS Logging Facility for SAS Programs	134
Which Language Elements Need Initializing?	134
Initializing the Logging Facility Autocall Macros	134
The LOGCONFIGLOC= System Option	134
Creating and Using Appenders in a SAS Program	135
Creating Appenders	135
Associating Appenders with Loggers	135
Creating Loggers in a SAS Program	136
Using SAS Language Elements to Create Loggers	136
Updating Logger Attributes	137
Message Categories in the SAS Language	137
Creating Log Events in a SAS Program	138
Example of Creating Logger and Appender Categories	139

Overview of the SAS Logging Facility in the SAS Language

The SAS language enables you to use the SAS logging facility in a DATA step and in macro programs. By using the SAS logging facility language elements, you can create appenders, loggers, and log events within 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 write a log event in your SAS program, the logger that you specify in the log event can be one that has been created within your SAS program or one that is configured in the logging configuration file.

SAS logging facility language elements are both [functions on page 155](#) and [DATA step component objects on page 165](#) for DATA step programming. The elements are also [autocall macros on page 141](#) for macro programming.

Appenders and loggers must be defined before SAS can process a log event in your SAS program. You include log events at any point in your SAS programs where you want to log a message of any diagnostic level. The levels, from lowest to highest, are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. Log events specify the logger, the diagnostic level, and the message.

The logging facility is enabled for SAS programs at all times. If the LOGCONFIGLOC= system option is not specified when SAS starts, all SAS logging facility messages are written to the SAS log as well as to the appender destinations that are associated with the logger that is named in a log event. When the LOGCONFIGLOC= system option is specified when SAS starts, messages are written to destinations, based on the logger hierarchy. For more information, see [“Hierarchical Logger Names” on page 9](#) and [“LOGCONFIGLOC= System Option” on page 24](#).

Initializing the SAS Logging Facility for SAS Programs

Which Language Elements Need Initializing?

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.

Initializing the Logging Facility Autocall Macros

In order to use autocall macros in SAS, you must set the MAUTOSOURCE system option. When SAS starts, the MAUTOSOURCE option is set, and no further action is required unless this option is turned off.

The logging facility autocall macro %LOG4SAS must be invoked before SAS processes any other logging facility autocall macros. The %LOG4SAS autocall macro defines all other logging facility autocall macros to the SAS session. You can invoke the %LOG4SAS autocall macro in an autoexec file, in an INITSTMT= system option, or at the beginning of your SAS program.

After the MAUTOSOURCE system option is set and the %LOG4SAS autocall macro has been invoked, you can invoke any of the logging facility autocall macros in your SAS program.

The LOGCONFIGLOC= System Option

If your SAS program does not write log events for SAS server loggers, the LOGCONFIGLOC= system option does not need to be set. If the program does write log events using SAS server loggers, you can check that the LOGCONFIGLOC= system option names a logging configuration file. You can check either by issuing the OPTIONS procedure or by viewing the LOGCONFIGLOC= system option in the SAS System Options window.

For more information, see [“LOGCONFIGLOC= System Option” on page 24](#) and [“SAS Server Logger Names” on page 9](#).

Creating and Using Appenders in a SAS Program

Creating Appenders

You create appenders in your SAS program or in the logging configuration file before you define loggers or before you 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`.

Although appenders can be created at any time in a SAS program, it is a good programming practice to create a named appender only once. In order to prevent the DATA step from processing the creation of the same appender in each iteration of the implicit loop, you can create an appender in only the first iteration of the implicit loop by using an IF-THEN statement:

```
if _n_ = 1 then
  do;
    rc=log4sas_appender("myAppenderName", "FileRefAppender", "fileref=myfile");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
```

When you create an appender, you specify the appender name, the keyword `FileRefAppender`, and appender options. You use appender options to specify a `fileref` that references a log file, a conversion pattern to format the message, and the appender message threshold. The appender `THRESHOLD` argument enables appender-level log event message filtering at the appender level. The filtering occurs after the logging facility processes logger-level message filtering.

The appender name is case sensitive. Be sure to specify the appender name exactly as it is specified in the respective appender syntax.

An appender that is created by using an autocall macro is defined to SAS for the duration of the SAS program. An appender that is created in a DATA step exists only for the duration of the DATA step. After the DATA step or SAS program is complete, appenders that are created in a DATA step are no longer defined to SAS.

For details, see the following language elements that create appenders in the SAS language:

- [“%LOG4SAS_APPENDER Autocall Macro” on page 145](#)
- [“LOG4SAS_APPENDER Function” on page 159](#)
- [“DECLARE Statement, Appender Object” on page 170](#)

Associating Appenders with Loggers

After an appender is defined to SAS, you can associate one or more appenders with a logger. All logger language elements have an `APPENDER-REF` argument whose value must be one or more appender names that are defined to SAS either in the logging

configuration file or in a SAS program. When a log event is invoked, the message is written to all destinations that are associated with the logger.

For details about the logger APPENDER-REF argument, see the following logger language elements:

- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “LOG4SAS_LOGGER Function” on page 161
- “DECLARE Statement, Logger Object” on page 172

Creating Loggers in a SAS Program

Using SAS Language Elements to Create Loggers

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 define appenders and before you invoke log events.

A named logger can be created only once. In order to prevent the DATA step from processing the creation of the same logger in each iteration of the implicit loop, you can create a logger in only the first iteration of the implicit loop by using an IF-THEN statement:

```
if _n_ = 1 then
  do;
    rc=log4sas_logger("myLoggerName", "appender-ref=(myAppenderName) level=info");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
```

When you create a logger, you specify the logger name and optional arguments for the message threshold and for one or more appender references. The logger name is case sensitive and can be a one-level or multiple-level name. The LEVEL argument specifies the message threshold that the logger processes. The logger-level threshold is the first level of message filtering. If a log event threshold is the same or greater than the logger threshold, the logger accepts the log event and the logging facility uses the appender arguments to process the log event. The thresholds, from lowest to highest, are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. Loggers can be associated with one or more appenders by specifying appender names in the APPENDER-REF argument. You separate appender names with a space and enclose the appender names in parentheses.

A logger is defined for the duration of the SAS session. For information about loggers, see the following topics:

- “Loggers” on page 7
- “Creating Log Events in a SAS Program” on page 138
- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “LOG4SAS_LOGGER Function” on page 161
- “DECLARE Statement, Logger Object” on page 172

Updating Logger Attributes

You can update a logger's attributes by using one of the language elements that creates loggers. To update logger attributes, you specify the logger creation language element by using the name of a logger that already exists and the new attributes. SAS updates the attributes of the logger with the new attributes.

Note: If the logger IMMUTABILITY attribute is set to TRUE, you cannot change the additivity setting or the level setting using the SAS language.

Message Categories in the SAS Language

When you create a logger in the SAS language, you create a category for messages that are logging messages. The message category is user-specified and is meaningful in your environment for the types of messages that you want to log. For example, if you are testing an existing SAS program where you have added new functionality, you might want messages in preexisting code to be logged as regression messages. Log messages for new code could be logged as new feature messages. Other logger categories might include department names, SAS program names, or analytical model names. For an example of logger category definitions, see [“Example of Creating Logger and Appender Categories” on page 139](#).

Message categories that you create in the SAS language differ from the types of message categories for SAS servers in that the SAS language message categories are user-defined, and the SAS server message categories are defined by SAS.

You can create message categories in a hierarchy where the hierarchy levels are separated by a . (period). Here are examples: IT, IT.Pgm1, and IT.Pgm2. The attributes that are defined in the higher-level logger can be used by lower-level loggers when the lower-level logger does not define an attribute. For example, you could create a high-level logger IT for your IT department. The logger IT specifies the level as INFO. Loggers IT.Pgm1 and IT.Pgm2 do not specify a level attribute. Therefore, they inherit the level of the next highest logger, which in this case is IT. Because the logger IT specifies the level as INFO, when a log event specifies the IT.Pgm1 or IT.Pgm2 logger, the logger level INFO is compared to the log event message level. The logger definitions in this scenario might look like the following functions:

```

/* Create the context for logging regression messages. */
/* Regression log events of level info or higher are written * /
/* to the destination, specified by the appender to be defined as ITPgmRegression. */

if _n_=1 then
  do;
    rc=log4sas_logger("IT", "appender-ref=(ITPgmRegression) level=info");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;

/* Create the context for Pgm1 in the IT department. */
/* Do not specify a level; use the IT logger level. */

```

```

if _n_=1 then
  do;
    rc=log4sas_logger("IT.Pgm1", "appender-ref=(ITPgm1Regression)");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;

/* Create the context for Pgm2 in the IT department. */
/* Do not specify a level; use the IT logger level. */

if _n_=1 then
  do;
    rc=log4sas_logger("IT.Pgm2", "appender-ref=(ITPgm2Regression)");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
end;

```

Creating Log Events in a SAS Program

After loggers and appenders are defined either in a logging configuration file or a SAS program, you can add log events to your program. You insert log events at any point 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.

The logger that you specify in the log event names the message category for the message. It can be a category that you created in your SAS program or a category that is defined for SAS servers. The diagnostic level indicates one of the following diagnostic types for the message: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. The log message is the message that you want to appear in the log. Enclose the message in single or double quotation marks.

For more information, see the following topics:

- [“Loggers” on page 7](#)
- [“Logging Thresholds” on page 16](#)
- [“%LOG4SAS_TRACE Autocall Macro” on page 149](#)
- [“%LOG4SAS_DEBUG Autocall Macro” on page 150](#)
- [“%LOG4SAS_INFO Autocall Macro” on page 151](#)
- [“%LOG4SAS_WARN Autocall Macro” on page 152](#)
- [“%LOG4SAS_ERROR Autocall Macro” on page 153](#)
- [“%LOG4SAS_FATAL Autocall Macro” on page 153](#)
- [“LOG4SAS_LOGEVENT Function” on page 163](#)
- [“TRACE Method” on page 178](#)

- “DEBUG Method” on page 169
- “INFO Method” on page 176
- “WARN Method” on page 179
- “ERROR Method” on page 174
- “FATAL Method” on page 175

Example of Creating Logger and Appender Categories

The following appender and logger functions create regression and new function categories for testing a SAS program. This example assumes that filerefs that are named myPgmReg and myPgmNew have been created in the SAS program.

```

/* Define the destination where regression messages are written. */

if _n_ = 1 then
  do;
    rc=log4sas_appender("myPgmRegression", "FileRefAppender", "fileref=myPgmReg");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;

/* Define the destination where new function messages are to be written. */

if _n_ = 1 then
  do;
    rc=log4sas_appender("myPgmNewFunction", "FileRefAppender", "fileref=myPgmNew");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;

/* Create the context for logging regression messages. */
/* Regression log events of level info or higher are written * /
/* to the destination specified by the appender defined as myPgmRegression. */

if _n_=1 then
  do;
    rc=log4sas_logger("regression", "appender-ref=(myPgmRegression) level=info");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
end;

```

```
/* Create the context for logging new function messages. */
/* New functionality log events of level debug or higher are written */
/* to the destination that is specified by the appender defined as myPgmNewFunction. */

if _n_=1 then
  do;
    rc=log4sas_logger("regression", "appender-ref=(myPgmNewFunction) level=debug");
    if rc ne 0 then do
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
end;
```

Chapter 11

Autocall Macro Reference

Using Autocall Macros to Log Messages	141
Example of Using Autocall Macros to Log Messages	142
Dictionary	145
%LOG4SAS Autocall Macro	145
%LOG4SAS_APPENDER Autocall Macro	145
%LOG4SAS_LOGGER Autocall Macro	147
%LOG4SAS_TRACE Autocall Macro	149
%LOG4SAS_DEBUG Autocall Macro	150
%LOG4SAS_INFO Autocall Macro	151
%LOG4SAS_WARN Autocall Macro	152
%LOG4SAS_ERROR Autocall Macro	153
%LOG4SAS_FATAL Autocall Macro	153

Using Autocall Macros to Log Messages

SAS supplies a set of autocall macros that you can use in your SAS programs to log messages by using the SAS logging facility. SAS writes the SAS language logging facility messages to a file using the appender, FileRefAppender. FileRefAppender is the only logging facility appender that is available in the SAS language.

You use the following autocall macros to log messages by using the logging facility:

%LOG4SAS (p. 145)

initializes the autocall macro logging environment.

%LOG4SAS_APPENDER (p. 145)

defines an appender, which names the destination of the log message, a message layout, and a message threshold.

%LOG4SAS_LOGGER (p. 147)

defines a logger, which defines a message category for log messages and the appenders that are associated with the message category.

%LOG4SAS_DEBUG (p. 150)

is the log event that you use to write debug messages.

%LOG4SAS_TRACE (p. 149)

is the log event that you use to write trace messages.

%LOG4SAS_WARN (p. 152)

is the log event that you use to write warning messages.

[%LOG4SAS_INFO](#) (p. 151)

is the log event that you use to write informational messages.

[%LOG4SAS_ERROR](#) (p. 153)

is the log event that you use to write error messages.

[%LOG4SAS_FATAL](#) (p. 153)

is the log event that you use to write fatal messages.

In order to use the logging facility autocall macros, you must set the MAUTOSOURCE system option in order to activate the autocall facility. The MAUTOSOURCE system option is set by default.

For more information about autocall macros and the MAUTOSOURCE system option, see *SAS Macro Language: Reference*.

Example of Using Autocall Macros to Log Messages

The macro program first retrieves the number of variables in a SAS data set and then appends each variable name to a macro variable. Logging facility debug log events send progress messages to a file that is referenced by the REV1 fileref. You can see from the SAS log output that the messages are written in the SAS log as well. By writing debug messages to a separate file, the logging facility acts as a filter where only the messages that you want to see are written to the file. The shaded code lines that follow are the statements that create logging messages.

```
filename rev1 ("c:\mySAS\Logs\rev1.log");
%log4sas();
%log4sas_appender(dsvar2mvar, "FileRefAppender", 'fileref=rev1');
%log4sas_logger(macroVar, 'level=debug appender-ref=(dsvar2mvar)');

/* Create sample data */

data one;
  input x y z;
datalines;
1 2 3
;

%macro lst(dsn);
  %global x;
  %let x=;
  /* Open the data set */
  %let dsid=%sysfunc(open(&dsn));

  /* Assign the number of variables into the macro variable CNT */
  %let cnt=%sysfunc(attrn(&dsid,nvars));
  %put cnt=&cnt;
  %log4sas_debug(macroVar, 'The number of variables is set in CNT');

  /* Create a macro variable that contains all data set variables */
  %do i = 1 %to &cnt;
    %let x=&x%sysfunc(varname(&dsid,&i));
    %log4sas_debug(macroVar, 'data set variable appended to macro variable');
```

```

%end;

/* Close the data set */
%let rc=%sysfunc(close(&dsid));
%mend lst;
%log4sas_debug(macroVar, 'lst Macro complete');

/* Call the macro and pass the name of the data set to be processed */
%log4sas_debug(macroVar, 'calling lst(one)');
%lst(one)
%put macro variable x = &x

%log4sas_debug(macroVar, 'macro lst(one) complete');

```

The file that is referenced by fileref REV1 contains these lines of text:

Output 11.1 Contents of the File Referenced by the REV1 Fileref

```

lst Macro complete
calling lst(one)
The number of variables is set in CNT
data set variable appended to macro variable
data set variable appended to macro variable
data set variable appended to macro variable
macro lst(one) complete

```

The following messages were written to the SAS log:

```

1  %log4sas();
2  %log4sas_appender(dsvar2mvar, "FileRefAppender", 'fileref=rev1');
3  %log4sas_logger(macroVar, 'level=debug appender-ref=dsvar2mvar');
4
5  /* Create sample data */
6
7  data one;
8      input x y;
9  datalines;

NOTE: The data set WORK.ONE has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds

11 ;
12
13 %macro lst(dsn);
14     %global x;
15     %let x=;
16     /* Open the data set */
17     %let dsid=%sysfunc(open(&dsn));
18
19     /* Assign the number of variables into the macro variable CNT */
20     %let cnt=%sysfunc(attrn(&dsid,nvars));
21     %put cnt=&cnt;
22     %log4sas_debug(macroVar, 'The number of variables is set in CNT!');
23
24     /* Create a macro variable that contains all data set variables */
25     %do i = 1 %to &cnt;
26         %let x=&x%sysfunc(varname(&dsid,&i));
27         %log4sas_debug(macroVar, 'data set variable appended to macro
variable');
28     %end;
29
30     /* Close the data set */
31     %let rc=%sysfunc(close(&dsid));
32 %mend lst;
33 %log4sas_debug(macroVar, 'lst Macro complete');
lst Macro complete
34
35 /* Call the macro and pass the name of the data set to be processed */
36 %log4sas_debug(macroVar, 'calling lst(one)');
calling lst(one)
37 %lst(one)
cnt= 3
The number of variables is set in CNT
data set variable appended to macro variable
data set variable appended to macro variable
data set variable appended to macro variable
38 %put macro variable x = &x
macro variable x = xyz
39
40 %log4sas_debug(macroVar, 'macro lst(one) complete');
macro lst(one) complete

```

Dictionary

%LOG4SAS Autocall Macro

Initializes the logging environment to use autocall macros.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
This macro must be invoked before any other logging autocall macro can be invoked.

Syntax

%LOG4SAS()

Details

You invoke the %LOG4SAS autocall macro in order to initialize the logging environment for SAS programming. To ensure that the logging environment is initialized when SAS starts, you can invoke the %LOG4SAS autocall macro as follows:

- in your autoexec file
- as a statement that is specified in the INITSTMT system option, which can be placed in the SAS configuration file or on the SAS command line.

You can also invoke the %LOG4SAS autocall macro by placing it at the beginning of your SAS program.

See Also

- [“%LOG4SAS_APPENDER Autocall Macro” on page 145](#)
- [“%LOG4SAS_LOGGER Autocall Macro” on page 147](#)
- [“%LOG4SAS_DEBUG Autocall Macro” on page 150](#)
- [“%LOG4SAS_TRACE Autocall Macro” on page 149](#)
- [“%LOG4SAS_WARN Autocall Macro” on page 152](#)
- [“%LOG4SAS_INFO Autocall Macro” on page 151](#)
- [“%LOG4SAS_ERROR Autocall Macro” on page 153](#)
- [“%LOG4SAS_FATAL Autocall Macro” on page 153](#)
- [“Example of Using Autocall Macros to Log Messages” on page 142](#)

%LOG4SAS_APPENDER Autocall Macro

Defines an appender.

Category: Logging

- Requirements:** The MAUTOSOURCE system option must be set.
- The %LOG4SAS autocall macro must be invoked before this macro is invoked to initialize the logging facility autocall macros.
- Arguments that follow the FileRefAppender argument must be enclosed as a group in single quotation marks.

Syntax

```
%LOG4SAS_APPENDER(name, "FileRefAppender"
    <,'<FILEREf=fileref><PATTERN="pattern"><THRESHOLD=threshold>'>
)
```

Syntax Description

name

specifies the name of the appender.

Tip: Appender names are case sensitive.

"FileRefAppender"

specifies the FileRefAppender class to which the defined appender belongs.

Note: This is the only supported appender class. More appender classes might be supported in the future.

Requirements:

Appender classes are case sensitive. In this instance, the class name must be *FileRefAppender*.

FileRefAppender must be enclosed in double quotation marks.

FILEREf=*fileref*

specifies the destination for log events that the FileRefAppender class processes.

PATTERN="*pattern*"

specifies the conversion pattern that is used to format the log message.

Requirement: The pattern must be enclosed in double quotation marks.

Tip: If PATTERN is not specified, the default is the message.

THRESHOLD=*threshold*

specifies the level at which log events are filtered out for the FileRefAppender. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Details

Appender Names

The *name* argument of the appender is specified as the value in the APPENDER-REF argument in the %LOG4SAS_LOGGER autocall macro. Here is an example:

```
filename myfile "my.log";
%log4sas();
%log4sas_appender(myAppender, "FileRefAppender", 'fileref=myfile');
%log4sas_logger(testlogger, 'level=info appender-ref=(myAppender)');
```

Patterns

Patterns are a feature of SAS logging that enables you to associate a layout with a particular logging output destination. The layout specifies how the output is formatted before it is sent to the output destination. The layout is specified as a pattern string that is

similar to the pattern strings that are used in the C language PRINTF statement. The pattern layout contains literal text and format directives that are called conversion specifiers.

Each conversion specifier has the following syntax:

```
%[format_modifiers] conversion_character
```

If a pattern is not specified, the default pattern contains just the application message.

For more information, see [Chapter 8, “Pattern Layouts,” on page 95](#).

Thresholds

An appender can be configured to have a threshold level. By default, appenders do not have a threshold set. When a threshold is set, all log events that have a level that is lower than the threshold level are ignored by the appender.

For more information, see [“Logging Thresholds” on page 16](#).

Example

The following appender definition names the appender `debugMyProgram`, names a log message destination using the fileref `debugOut`, and specifies a pattern that reports the filename and the line number of the application that generated the log event:

```
filename debugOut="c:\myDebugOutput.txt";
%log4sas();
%log4sas_appender(debugMyProgram, "FileRefAppender",
                  'fileref=debugOut pattern="(F:L)m" threshold=trace');
```

See Also

- [“%LOG4SAS Autocall Macro” on page 145](#)
- [“%LOG4SAS_LOGGER Autocall Macro” on page 147](#)
- [“%LOG4SAS_DEBUG Autocall Macro” on page 150](#)
- [“%LOG4SAS_TRACE Autocall Macro” on page 149](#)
- [“%LOG4SAS_WARN Autocall Macro” on page 152](#)
- [“%LOG4SAS_INFO Autocall Macro” on page 151](#)
- [“%LOG4SAS_ERROR Autocall Macro” on page 153](#)
- [“%LOG4SAS_FATAL Autocall Macro” on page 153](#)
- [“Example of Using Autocall Macros to Log Messages” on page 142](#)

%LOG4SAS_LOGGER Autocall Macro

Defines a logger.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked before this macro is invoked to initialize SAS logging.

Syntax

```
%LOG4SAS_LOGGER(name<, "<ADDITIVITY=TRUE | FALSE>
<APPENDER-REF=(appender-list)><LEVEL=level> "> )
```

Syntax Description

name

specifies the name of the logger.

Tip: You can specify the root logger by setting *name* equal to either two double quotation marks with no space between them (" "), or to "root". If you specify the root logger, these settings are in effect only during the lifespan of the DATA step. Root setting before and after the DATA step are based on the logging configuration file.

ADDITIVITY=TRUE | FALSE

specifies whether to pass log events to only the appender that is associated with the logger or to all of the appenders in the logger's hierarchy.

Restriction: ADDITIVITY can be modified for a logger only if the logger's IMMUTABILITY option in the logging configuration file is set to FALSE. If you cannot modify a logger's ADDITIVITY option, contact your system administrator.

APPENDER-REF=(*appender-list*)

specifies one or more appender names to which log events are passed.

Requirements:

The appender names must already exist. Appendings are created by the %LOG4SAS_APPENDER autocall macro or defined in a logging configuration file.

When you specify more than one appender, the list must be enclosed in parentheses.

Interaction: If ADDITIVITY=TRUE, log events are also passed to all of the appenders that are associated with the logger's hierarchy.

LEVEL=*level*

specifies the level at which log events of the specified level and higher are processed by the logger. The following are the valid level values, from lowest to highest: TRACE, DEBUG, INFO, WARN, ERROR, FATAL.

Restriction: LEVEL can be modified for a logger only if the logger is defined in a logging configuration file and if the IMMUTABILITY attribute in the logger configuration file is set to FALSE. If you cannot modify a logger's LEVEL option, contact your system administrator.

Details

Logger Names

A logger is an ancestor of another logger if the logger name, followed by a dot, is the prefix of the other logger.

In the following example, MYPROGRAM is the parent logger. MYPROGRAM is the ancestor of the UNITTEST logger, and both MYPROGRAM and UNITTEST are ancestors of the REV1 logger.

```
MYPROGRAM
MYPROGRAM.UNITTEST
MYPROGRAM.UNITTEST.REV1
```

The hierarchical organization of loggers enables them to inherit log event levels and appenders from their ancestors.

Additivity

By default, each log event is passed to the appenders that are referenced by the logger and to the appenders that are referenced by loggers in the logger's hierarchy. This is the meaning of the term *appender additivity*.

For example, by default, when a log event is processed by the logger `MyProgram.UnitTest.Rev1`, the log message is also directed to the appenders that are referenced in the `MyProgram.UnitTest` and `MyProgram` loggers. If `ADDITIVITY=FALSE`, the log message is directed only to the appenders that are referenced by `MyProgram.UnitTest.Rev1`.

Levels

A logging request is applied if its level is greater than or equal to the level of the logger. Otherwise, the logging request is ignored. Loggers without an explicitly assigned level inherit their level from the hierarchy. For more information about logging levels, see “Logging Thresholds” on page 16.

See Also

- “%LOG4SAS Autocall Macro” on page 145
- “%LOG4SAS_APPENDER Autocall Macro” on page 145
- “%LOG4SAS_DEBUG Autocall Macro” on page 150
- “%LOG4SAS_TRACE Autocall Macro” on page 149
- “%LOG4SAS_WARN Autocall Macro” on page 152
- “%LOG4SAS_INFO Autocall Macro” on page 151
- “%LOG4SAS_ERROR Autocall Macro” on page 153
- “%LOG4SAS_FATAL Autocall Macro” on page 153
- “Example of Using Autocall Macros to Log Messages” on page 142

%LOG4SAS_TRACE Autocall Macro

Logs a TRACE message if the specified logger accepts TRACE messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

```
%LOG4SAS_TRACE(logger-name, message)
```

Syntax Description

logger-name

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_TRACE autocall macro is a log event for trace messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- [“%LOG4SAS Autocall Macro” on page 145](#)
- [“%LOG4SAS_APPENDER Autocall Macro” on page 145](#)
- [“%LOG4SAS_LOGGER Autocall Macro” on page 147](#)
- [“%LOG4SAS_DEBUG Autocall Macro” on page 150](#)
- [“%LOG4SAS_WARN Autocall Macro” on page 152](#)
- [“%LOG4SAS_INFO Autocall Macro” on page 151](#)
- [“%LOG4SAS_ERROR Autocall Macro” on page 153](#)
- [“%LOG4SAS_FATAL Autocall Macro” on page 153](#)
- [“Example of Using Autocall Macros to Log Messages” on page 142](#)

%LOG4SAS_DEBUG Autocall Macro

Logs a DEBUG message if the specified logger accepts DEBUG messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

%LOG4SAS_DEBUG(*logger-name*, *message*)

Syntax Description**logger-name**

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_DEBUG autocall macro is a log event for debugging messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- “%LOG4SAS Autocall Macro” on page 145
- “%LOG4SAS_APPENDER Autocall Macro” on page 145
- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “%LOG4SAS_TRACE Autocall Macro” on page 149
- “%LOG4SAS_WARN Autocall Macro” on page 152
- “%LOG4SAS_INFO Autocall Macro” on page 151
- “%LOG4SAS_ERROR Autocall Macro” on page 153
- “%LOG4SAS_FATAL Autocall Macro” on page 153
- “Example of Using Autocall Macros to Log Messages” on page 142

%LOG4SAS_INFO Autocall Macro

Logs an INFO message if the specified logger accepts INFO messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

`%LOG4SAS_INFO(logger-name, message)`

Syntax Description

logger-name

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_INFO autocall macro is a log event for informational messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- “%LOG4SAS Autocall Macro” on page 145
- “%LOG4SAS_APPENDER Autocall Macro” on page 145
- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “%LOG4SAS_DEBUG Autocall Macro” on page 150
- “%LOG4SAS_TRACE Autocall Macro” on page 149

- “%LOG4SAS_WARN Autocall Macro” on page 152
- “%LOG4SAS_ERROR Autocall Macro” on page 153
- “%LOG4SAS_FATAL Autocall Macro” on page 153
- “Example of Using Autocall Macros to Log Messages” on page 142

%LOG4SAS_WARN Autocall Macro

Logs a WARN message if the specified logger accepts WARN messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

`%LOG4SAS_WARN(logger-name, message)`

Syntax Description

logger-name

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_WARN autocall macro is a log event for warning messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- “%LOG4SAS Autocall Macro” on page 145
- “%LOG4SAS_APPENDER Autocall Macro” on page 145
- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “%LOG4SAS_DEBUG Autocall Macro” on page 150
- “%LOG4SAS_WARN Autocall Macro” on page 152
- “%LOG4SAS_INFO Autocall Macro” on page 151
- “%LOG4SAS_ERROR Autocall Macro” on page 153
- “%LOG4SAS_FATAL Autocall Macro” on page 153
- “Example of Using Autocall Macros to Log Messages” on page 142

%LOG4SAS_ERROR Autocall Macro

Logs an ERROR message if the specified logger accepts ERROR messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.
The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

`%LOG4SAS_ERROR(logger-name, message)`

Syntax Description

logger-name

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_ERROR autocall macro is a log event for error messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- [“%LOG4SAS Autocall Macro” on page 145](#)
- [“%LOG4SAS_APPENDER Autocall Macro” on page 145](#)
- [“%LOG4SAS_LOGGER Autocall Macro” on page 147](#)
- [“%LOG4SAS_DEBUG Autocall Macro” on page 150](#)
- [“%LOG4SAS_TRACE Autocall Macro” on page 149](#)
- [“%LOG4SAS_WARN Autocall Macro” on page 152](#)
- [“%LOG4SAS_INFO Autocall Macro” on page 151](#)
- [“%LOG4SAS_FATAL Autocall Macro” on page 153](#)
- [“Example of Using Autocall Macros to Log Messages” on page 142](#)

%LOG4SAS_FATAL Autocall Macro

Logs a FATAL message if the specified logger accepts FATAL messages.

Category: Logging

Requirements: The MAUTOSOURCE system option must be set.

The %LOG4SAS autocall macro must be invoked to initialize SAS logging before this macro is invoked.

Syntax

`%LOG4SAS_FATAL(logger-name, message)`

Syntax Description

logger-name

specifies the name of the logger to process this log event.

message

specifies the log event message.

Requirement: The message must be enclosed in single or double quotation marks.

Details

The %LOG4SAS_FATAL autocall macro is a log event for fatal messages. In order to log messages using this macro, you must previously define loggers and appenders either in a SAS program or in a logging configuration file.

See Also

- “%LOG4SAS Autocall Macro” on page 145
- “%LOG4SAS_APPENDER Autocall Macro” on page 145
- “%LOG4SAS_LOGGER Autocall Macro” on page 147
- “%LOG4SAS_DEBUG Autocall Macro” on page 150
- “%LOG4SAS_TRACE Autocall Macro” on page 149
- “%LOG4SAS_WARN Autocall Macro” on page 152
- “%LOG4SAS_INFO Autocall Macro” on page 151
- “%LOG4SAS_ERROR Autocall Macro” on page 153
- “Example of Using Autocall Macros to Log Messages” on page 142

Chapter 12

Function Reference

Using the Logging Facility Functions in the DATA Step	155
Logging Example Using Functions	156
Dictionary	159
LOG4SAS_APPENDER Function	159
LOG4SAS_LOGGER Function	161
LOG4SAS_LOGEVENT Function	163

Using the Logging Facility Functions in the DATA Step

SAS supplies three logging facility functions that you can use in the DATA step:

[LOG4SAS_APPENDER](#) (p. 159)

creates an appender. FileRefAppender is the only type of appender that can be created by using the SAS language.

[LOG4SAS_LOGGER](#) (p. 161)

logs a message by using a specific logger.

[LOG4SAS_LOGEVENT](#) (p. 163)

logs a message by using a specific logger.

You use logging facility functions in the same way as you use other functions in SAS: by assigning the function to a variable.

```
rc=log4sas_appender("myAppenderName", "FileRefAppender", "fileref=myfiles");
```

When you create appenders and loggers, remember to create them only once in a DATA step, as in this example:

```
if _n_ = 1 then
do;
rc=log4sas_appender("myAppenderName", "FileRefAppender",
"fileref=myfile");

if rc ne 0 then do;
msg = sysmsg();
put msg;
ABORT;
end;

rc=log4sas_logger("myLoggerName", "appender-ref=(myAppenderName) level=info");
```

```

    if rc ne 0 then do;
        msg = sysmsg();
        put msg;
        ABORT;
    end;
end;

```

Logging Example Using Functions

The example program determines the number of years, months, and days between two SAS date values. It uses logging facility functions to write progress messages to an external file. The program is structured so that the appender and the logger are created, and variables are initialized during the first iteration of the DATA step in order to ensure efficiency of the program.

```

data a;
    input @1 dob mmdyy10.;
    format dob tod mmdyy10.;

    /* In the first iteration of the DATA step, create an appender          */
    /* and a logger, and initialize variables tod and bdays. Then, determine */
    /* the number of days in the month prior to the current month.          */

    if _n_ = 1 then
        do;
            rc=log4sas_appender("functionAppender", "FileRefAppender",
                "fileref=myfile");
            if rc ne 0 then do;
                msg = sysmsg();
                put msg;
                ABORT;
            end;

            rc=log4sas_logger("functionLogger", "appender-ref=(functionAppender)
                level=info");

            if rc ne 0 then do;
                msg = sysmsg();
                put msg;
                ABORT;
            end;

            /* Get the current date from the operating system */
            tod=today();
            retain tod;

            rc=log4sas_logevent("functionLogger", "info", "Obtained today's date.");
            if rc ne 0 then do;
                msg = sysmsg();
                put msg;
                ABORT;
            end;

            /* Determine the number of days in the month prior to current month */

```

```

        bdays=day(intnx('month',tod,0)-1);
        retain bdays;

        rc=log4sas_logevent("functionLogger", "info",
                            "Determined the number of business
days.");
        if rc ne 0 then do;
            msg = sysmsg();
            put msg;
            ABORT;
            end;

end; /* end the processing for first iteration */

/* Find the difference in days, months, and years between */
/* start and end dates                                     */
dd=day(tod)-day(dob);
mm=month(tod)-month(dob);
yy=year(tod)-year(dob);

rc=log4sas_logevent("functionLogger", "info", "Found date differences.");
if rc ne 0 then do;
    msg = sysmsg();
    put msg;
    ABORT;
    end;

/* If the difference in days is a negative value, add the number */
/* of days in the previous month and reduce the number of months */
/* by 1.                                                         */
if dd < 0 then do;
    dd=bdays+dd;
    mm=mm-1;

    rc=log4sas_logevent("functionLogger", "info", "Made adjustments in
days.");
    if rc ne 0 then do;
        msg = sysmsg();
        put msg;
        ABORT;
        end;
    end;

/* If the difference in months is a negative number add 12 */
/* to the month count and reduce the year count by 1.      */
if mm < 0 then do;
    mm=mm+12;
    yy=yy-1;

    rc=log4sas_logevent("functionLogger", "info", "Made adjustments in
months.");
    if rc ne 0 then do;
        msg = sysmsg();
        put msg;
        ABORT;
    end;
end;

```

```

        end;
    end;

    datalines;
    01/01/1986
    02/28/1990
    12/03/2006
    02/28/2000
    02/29/2000
    03/01/2000
    05/10/1974
    05/11/1974
    05/12/1974
    ;

proc print label;
    label dob='Date of Birth'
           tod="Today's Date"
           dd='Difference in Days'
           mm= 'Difference in Months'
           yy='Difference in Years';
    var dob tod yy mm dd;
run;

```

The file that is represented by the MYFILE fileref contains the following logging facility messages:

```

Obtained today's date.
Determined the number of business days.
Found date differences.
Found date differences.
Made adjustments in days.
Found date differences.
Made adjustments in months.
Found date differences.
Made adjustments in days.
Found date differences.
Made adjustments in days.
Found date differences.
Found date differences.
Made adjustments in months.
Found date differences.
Made adjustments in months.
Found date differences.
Made adjustments in months.

```

Here is the program output:

Display 12.1 Program Output for Determining Date Differences

<i>The SAS System</i>					
Obs	Date of Birth	Today's Date	Difference in Years	Difference in Months	Difference in Days
1	01/01/1986	03/27/2008	22	2	26
2	02/28/1990	03/27/2008	18	0	28
3	12/03/2006	03/27/2008	1	3	24
4	02/28/2000	03/27/2008	8	0	28
5	02/29/2000	03/27/2008	8	0	27
6	03/01/2000	03/27/2008	8	0	26
7	05/10/1974	03/27/2008	33	10	17
8	05/11/1974	03/27/2008	33	10	16
9	05/12/1974	03/27/2008	33	10	15

Dictionary

LOG4SAS_APPENDER Function

Creates a fileref appender that can be referenced by a logger.

Category: Logging

Example: [“Logging Example Using Functions” on page 156](#)

Syntax

```
LOG4SAS_APPENDER("name", "FileRefAppender", 'options')
```

Required Arguments

"name"

specifies a name for the appender.

Tip: The appender name is case sensitive.

"FileRefAppender"

specifies that a fileref is used as the destination for the appender.

'options'

specify one or more of the following values:

FILEREF=*fileref*

specifies a fileref that is used as the log message destination for this appender.

Requirement: Yes

`PATTERN="pattern"`

specifies one or more message layout patterns that are used to format the log message.

See: [Chapter 8, “Pattern Layouts,” on page 95](#)

`THRESHOLD="threshold"`

specifies a level at which log events that are lower than *threshold* are filtered out for the appender. Valid values for *threshold*, from lowest to highest, are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Requirement: Options must be enclosed in single quotation marks.

Details

Appender Names

Appender names follow SAS naming conventions. An appender is associated with a logger by using the appender name as one of the values of the APPENDER-REF option in the LOG4SAS_LOGGER function.

FileRefAppender

A FileRefAppender is the only type of appender that can be used in the SAS language.

Patterns

Patterns are a feature of SAS logging that enable you to associate a layout with a particular logging output destination. The layout specifies how the output is formatted before it is sent to the output destination. The layout is specified as a pattern string that is similar to the pattern strings that are used in the C language PRINTF statement. The pattern layout contains literal text and format directives that are called conversion specifiers.

Each conversion specifier has the following syntax:

```
%[format_modifiers]conversion_character
```

If a pattern is not specified, the default is the message.

For more information, see [Chapter 8, “Pattern Layouts,” on page 95](#).

Thresholds

An appender can be defined to have a threshold level. By default, appenders do not have a threshold. When a threshold is set, all log events that have a level that is lower than the threshold level are ignored by the appender.

For more information, see [“Logging Thresholds” on page 16](#).

Processing Appendors in the DATA Step

An appender needs to be created only one time for each DATA step. Because the DATA step uses the implicit loop to process observations in a data set, you can use the automatic variable `_N_` in an IF statement to process the LOG4SAS_APPENDER function during the first DATA step iteration:

```
if _n_ = 1 then
  do;
    rc=log4sas_appender("myAppenderName", "FileRefAppender", "fileref=myfiles");
    if rc ne 0 then do;
      msg = sysmsg();
      put msg;
```

```

        ABORT;
    end;
end;

```

See Also

Functions:

- [“LOG4SAS_LOGGER Function” on page 161](#)
- [“LOG4SAS_LOGEVENT Function” on page 163](#)

LOG4SAS_LOGGER Function

Creates a logger.

Category: Logging

Example: [“Logging Example Using Functions” on page 156](#)

Syntax

```
LOG4SAS_LOGGER("name", <"options"> )
```

Required Arguments

"name"

specifies a name for the logger.

Requirement: The name must be enclosed in double quotation marks.

Tips:

Requests to create a logger are ignored if they use the name of an existing logger.

You can specify the root logger by setting *name* equal to either two double quotation marks with no space between them (" "), or to *root*. If you specify the root logger, these settings are in effect only during the lifespan of the DATA step. Root settings before and after the DATA step are based on the logging configuration file.

Example: App.Security

"options"

specify one or more of the following options for this logger:

ADDITIVITY=(TRUE | FALSE)

specifies whether to pass a log event to only the appender that is associated with the logger or to all of the appenders in the logger's hierarchy. TRUE specifies to send a log event to all of the appenders in the logger's hierarchy. FALSE specifies to send a log event to only the appenders that are referenced by the APPENDER-REF= option.

Default: TRUE

Restriction: ADDITIVITY can be modified for a logger only if the logger's IMMUTABILITY option in the logging configuration file is set to FALSE. If you cannot modify a logger's ADDITIVITY option, contact your system administrator.

APPENDER-REF=(*appender_name_list*)

specifies one or more appender names to which log events for the logger are passed. Separate the appender names with a space or a comma.

Requirement: An appender must be defined in a SAS program before it can be used in *appender_name_list*.

Tip: If the value of ADDITIVITY is TRUE, then the log events are processed by appenders that are found in the logger's hierarchy.

LEVEL=*level*

specifies the ranking, or level, of a log event message that the logger processes. The logger processes log events whose level is the same as or greater than *level*. The levels, from the lowest level to the highest level are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Restriction: LEVEL can be modified for a logger only if the logger's IMMUTABILITY option in the logger configuration file is set to FALSE. If you cannot modify a logger's LEVEL option, contact your system administrator.

Details

Logger Names

The logger name associates a logger with a log message. You can send log messages to be processed by a logger by specifying the logger name as the *name* argument in the LOG4SAS_LOGEVENT function.

A logger is an ancestor of another logger if the logger name, followed by a dot, is the prefix of the other logger. The following names are logger names:

```
Testing
Testing.MyProg
Testing.MyProg.TraceMsgs
```

Testing is the parent logger and the ancestor of the loggers MyProg and TraceMsgs. MyProg is the ancestor of TraceMsgs. The logger Testing.MyProg.TraceMsgs provides a message category that can be used to log trace messages when you are testing the program MyProg.

The hierarchical organization of loggers enables loggers to inherit levels and appenders from their ancestors. For information about configuring loggers in a hierarchy, see [“Hierarchical Logger Names” on page 9](#).

Appender Reference and Additivity

The appenders that are in *appender_name_list* must be defined by using the LOG4SAS_APPENDER function or in a logging configuration file before the LOG4SAS_LOGGER function executes.

By default, each log event is passed to the appenders that are referenced by the logger and to the appenders that are referenced by loggers in the logger's hierarchy. This is the meaning of the term *appender additivity*.

For example, by default, when a log event is processed by the logger Testing.MyProg.TraceMsgs, the log message is also directed to the appenders that are referenced in the Testing.MyProg and Testing loggers. If ADDITIVITY=FALSE, the log message is directed to only the appenders that are referenced by Testing.MyProg.TraceMsgs.

Levels

A log event is applied if the level of the log event is the same or greater than the level of the logger. If the level of the log event is lower than the level of the logger, then the log event is discarded. For more information about levels, see [“Logging Thresholds” on page 16](#).

If a logger does not define a level, the logger inherits the level from the next highest ancestor that has an assigned level.

Processing Loggers in the DATA Step

A logger needs to be created only one time for each DATA step. Because the DATA step uses the implicit loop to process observations in a data set, you can use the automatic variable `_N_` in an IF statement to process the LOG4SAS_LOGGER function during the first DATA step iteration:

```
if _n_ = 1 then
  do;
    rc=log4sas_logger("myLoggerName", "appender-ref=(functionAppender) level=info");
    if rc ne 0 then do;
      msg = sysmsg();
      put msg;
      ABORT;
    end;
  end;
end;
```

See Also

Functions:

- [“LOG4SAS_APPENDER Function” on page 159](#)
- [“LOG4SAS_LOGEVENT Function” on page 163](#)

LOG4SAS_LOGEVENT Function

Logs a message by using a specific logger.

Category: Logging

Example: [“Logging Example Using Functions” on page 156](#)

Syntax

`Log4SAS_logevent(name, level, message)`

Required Arguments

"name"

specifies a name for the logger that processes the log event.

Requirement: The name must be enclosed in quotation marks.

"level"

specifies one of the following message levels:

TRACE	produces the most detailed information about your application. This level is primarily used by SAS Technical Support or development.
DEBUG	produces detailed information that you use to debug your application. This level is primarily used by SAS Technical Support or development.
INFO	provides information that highlights the progress of an application.
WARN	provides messages that identify potentially harmful situations.
ERROR	provides messages that indicate that errors have occurred. The application might be able to continue.
FATAL	provides messages that indicate that severe errors have occurred. These errors will probably cause the application to end.

Requirement: The level must be enclosed in quotation marks.

"message"

specifies the message that is logged.

Requirement: The message must be enclosed in quotation marks.

Interaction: The only variables that the message can resolve are macro variables. DATA step variables do not resolve in the message.

Details

Name

The log message *name* argument names a logger to process the log message.

A logger is an ancestor of another logger if the logger name, followed by a dot, is the prefix of the other logger. The following names are logger names:

```
Testing
Testing.MyProg
Testing.MyProg.TraceMsgs
```

Testing is the parent logger and the ancestor of the loggers MyProg and TraceMsgs. MyProg is the ancestor of the logger TraceMsgs. The logger Testing.MyProg.TraceMsgs provides a message category that can be used to log trace messages when you are testing the program MyProg.

The hierarchical organization of loggers enables loggers to inherit levels and appenders from their ancestors. For information about configuring loggers in a hierarchy, see [“Hierarchical Logger Names” on page 9](#).

See Also

Functions:

- [“LOG4SAS_APPENDER Function” on page 159](#)
- [“LOG4SAS_LOGGER Function” on page 161](#)

Chapter 13

Component Object Reference

The Logger and Appender Component Object Interface	165
Dot Notation and DATA Step Component Objects	166
Definition	166
Syntax	166
Dictionary	167
ADDITIVITY Attribute	167
APPENDERREF Attribute	168
DEBUG Method	169
DECLARE Statement, Appender Object	170
DECLARE Statement, Logger Object	172
ERROR Method	174
FATAL Method	175
INFO Method	176
LEVEL Attribute	177
TRACE Method	178
WARN Method	179

The Logger and Appender Component Object Interface

SAS provides two predefined component objects that you can use in a DATA step to access SAS logging: the appender object and the logger object. These objects enable you to record log events and write these events to the appropriate destinations.

The DATA step Component Interface enables you to create and manipulate the logger and appender objects by using statements, attributes, and methods. You use the DECLARE statement to declare and create a component object. You use DATA step object dot notation to access the component object's attributes and methods. Attributes are the properties that specify the information that is associated with an object. Methods define the operations that an object can perform.

An appender and logger object need to be created only one time for each DATA step. Because the DATA step uses the implicit loop to process observations in a data set, you can use the automatic variable `_N_` in an IF statement to process the appender and logger object code during the first DATA step iteration.

Note: The DATA step component object statements, attributes, and methods are limited to those defined for these objects. You cannot use SAS Component Language functionality with these predefined DATA step objects.

Dot Notation and DATA Step Component Objects

Definition

Dot notation provides a shortcut for invoking methods and for setting and querying attribute values. Using dot notation makes your SAS programs easier to read.

To use dot notation with a DATA step component object, you must declare and instantiate the component object by using the DECLARE statement.

Syntax

```
object.attribute
```

or

```
object.method(<argument_tag-1: value-1<, ...argument_tag-n: value-n>>);
```

Arguments

object

specifies the variable name for the DATA step component object.

attribute

specifies an object attribute to assign or query.

When you set an attribute for an object, the code takes this form:

```
object.attribute = value;
```

When you query an object attribute, the code takes this form:

```
value = object.attribute;
```

method

specifies the name of the method to invoke.

argument_tag

identifies the arguments that are passed to the method. Enclose the argument tag in parentheses. The parentheses are required whether or not the method contains argument tags.

All DATA step component object methods take this form:

```
return_code = object.method(<argument_tag-1: value-1  
                           <, ...argument_tag-n: value-n>>);
```

The return code indicates whether the method is successful or failed. A return code of zero indicates success; a nonzero value indicates failure. If you do not supply a return code variable for the method call and if the method fails, an error message is printed to the log.

value

specifies the argument value.

Dictionary

ADDITIVITY Attribute

Specifies whether to pass a log event only to the appender that is associated with the logger or to the appenders in the logger's hierarchy.

Applies to: logger object

Syntax

```
object.ADDITIVITY = "TRUE | FALSE";
```

Required Arguments

object

specifies the name of the logger object.

"TRUE | FALSE"

determines whether a log event is processed by the appenders that exist in the specified logger's hierarchy.

Default: TRUE

Details

By default, each log event is passed to the appenders that are associated with the logger and to the appenders that are associated with the logger's hierarchy. This is the meaning of the term *appender additivity*.

For example, by default, when a log event is processed by the logger `Testing.MyProg.TraceMsgs`, the log message is also directed to the appenders that are referenced in the `Testing.MyProg` and `Testing` loggers. If `ADDITIVITY=FALSE`, the log message is directed to only the appenders that are referenced by `Testing.MyProg.TraceMsgs`.

Note: You can also specify the logger additivity in the logger's constructor by using the `DECLARE` statement. For more information, see [“DECLARE Statement, Logger Object” on page 172](#).

Example

The following code sets the additivity attribute to `FALSE`.

```
data _null_;  
  if _n_ = 1 then do;  
    declare logger logobj("mylog");  
  end;  
  logobj.additivity="false";  
run;
```

Alternatively, you can set the additivity attribute in the `DECLARE` statement.

```

data _null_;
  if _n_ = 1 then do;
    declare logger logobj("mylog", additivity:"false");
  end;
run;

```

See Also

Statements:

- [“DECLARE Statement, Logger Object” on page 172](#)

APPENDERREF Attribute

Specifies the name of the appender to which log events are passed.

Applies to: logger object

Syntax

```
object.APPENDERREF = "appender-name";
```

Required Arguments

object

specifies the name of the logger object.

appender-name

specifies the name of the appender to which the log events for the specified logger are passed.

Interactions:

If the ADDITIVITY attribute is set to TRUE, the log events are also passed to all the appenders that are associated with the logger's hierarchy.

The appender name must already exist. Appender names are created by using the DECLARE statement or in a logging configuration file.

See: [“DECLARE Statement, Appender Object” on page 170](#)

Details

You can specify more than one appender for each logger.

Example

The logger object in the following example references the myappd appender.

```

filename myfref "my.log";

data _null_;
  if _n_ = 1 then do;
    declare appender appobj("myappd", "FileRefAppender", "FileRef=myfref");
    declare logger logobj("mylog", level: "info");
    logobj.appenderref="myappd";
  end;

```

```

logobj.additivity="false";
logobj.info("my info message");
run;

```

See Also

Attributes:

- [“ADDITIVITY Attribute” on page 167](#)

Statements:

- [“DECLARE Statement, Appender Object” on page 170](#)

DEBUG Method

Logs a DEBUG message if the specified logger accepts DEBUG messages.

Applies to: logger object

Syntax

```
object.DEBUG ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the debug level.

Requirement: The message must be enclosed in quotation marks.

Details

The debug level designates fine-grained informational events that are most useful to debug an application. For more information about logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates a debugging message for the logger.

```

data _null_;
  if _n_=1 then do;
    declare appender appobj("myappender", "FileRefAppender", "fileref=myfile");
    declare logger logobj("testlog", AppenderRef: "myappender");
  end;
  ...more-sas-code...

  logobj.debug("Test debug message");

  ...more-sas-code...
run;

```

See Also

Methods:

- “ERROR Method” on page 174
- “FATAL Method” on page 175
- “INFO Method” on page 176
- “TRACE Method” on page 178
- “WARN Method” on page 179

DECLARE Statement, Appender Object

Declares an appender object; creates an instance of an appender object and initializes data for an appender object.

Valid in: DATA step

Category: Action

Type: Executable

Alias: DCL

Syntax

```
DECLARE APPENDER appender-object ("appender-name",
"FileRefAppender", "FILEREf=fileref" <, PATTERN: "pattern">
<, THRESHOLD: "threshold"> );
```

Required Arguments

appender-object

specifies the name of the appender object.

appender-name

specifies the name of the appender to which the log events are passed.

Requirement: The name must be enclosed in double quotation marks.

Interaction: This name is valid for use wherever an AppenderRef is accepted (for example, in the DECLARE statement for the logger object).

Tip: Appender names are case sensitive.

FileRefAppender

specifies the FileRefAppender class to which the defined appender instance belongs.

Requirements:

Appender class names are case sensitive. In this instance, the name must be “FileRefAppender”.

FileRefAppender must be enclosed in double quotation marks.

Note: This is the only supported appender class. More appender classes might be supported in the future.

FILEREf="*fileref*"

specifies the destination for log events that the FileRefAppender class processes.

Requirement: If the FileRefAppender argument is specified, this argument also must be specified.

PATTERN: "pattern"

specifies the conversion pattern that is used to format the log message.

Requirement: The pattern must be enclosed in double quotation marks.

Tip: If a conversion pattern is not specified, the default pattern is %m, the message.

THRESHOLD: "threshold"

specifies the level at which log events are filtered out for the specified appender object. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Requirement: The level must be enclosed in double quotation marks.

Details

Appender Names

Appender names follow the rules for SAS naming conventions. For appender objects, the name can be referenced by the logger object in the APPENDERREF attribute. For more information, see [“APPENDERREF Attribute” on page 168](#). Here is an example:

```
filename myfile "my.log";
declare appender appobj("workappd", "FileRefAppender", "FileRef=myfile");
declare logger logobj("testlog");
logobj.appenderref="workappd";
```

FileRefAppender

A FileRefAppender is the only type of appender that can be used in the SAS language.

Patterns

Patterns are a feature of SAS logging that enables you to associate a layout with a particular logging output destination. The layout specifies how the output is formatted before it is sent to the output destination. The layout is specified as a pattern string that is similar to the pattern strings that are used in the C language PRINTF statement. The pattern layout contains literal text and format directives that are called conversion specifiers.

Each conversion specifier has the following syntax:

```
% [format_modifiers] conversion_character
```

If a pattern is not specified, the default pattern is %m, the message.

For more information, see [Chapter 8, “Pattern Layouts,” on page 95](#).

Thresholds

An appender can be configured to have a threshold level. By default, appenders do not have a threshold set. When a threshold is set, all log events that have a level that is lower than the threshold level are ignored by the appender.

For more information, see [“Logging Thresholds” on page 16](#).

Example

This example creates an appender object.

```
data _null_;
  if _n_ = 1 then do;
```

```

        declare appender appobj("testappd", "FileRefAppender", "fileref=testfref",
            pattern:"%nrstr(%d{yyyMMdd:HH:mm:ss.SS}: %t:%8p %m)",
            threshold:"fatal");
    end;
run;

```

See Also

Attributes:

- “APPENDERREF Attribute” on page 168

Statements:

- “DECLARE Statement, Logger Object” on page 172

DECLARE Statement, Logger Object

Declares a logger object; creates an instance of a logger object and initializes data for a logger object.

Valid in: DATA step

Category: Action

Type: Executable

Alias: DCL

Syntax

```

DECLARE LOGGER logger-object ("logger-name" <, ADDITIVITY: TRUE | FALSE>
<, LEVEL: "level">
<, APPENDERREF: "appender-name"<..., APPENDERREF: "appender-name">> );

```

Required Arguments

logger-object

specifies the name of the logger object.

logger-name

specifies the name of the logger to which the specified options are applied.

Requirement: The name must be enclosed in double quotation marks.

Tip: You can specify the root logger by setting *name* equal to either double quotation marks with no space between them (" "), or to "root". If you specify the root logger, these settings are in effect only during the lifespan of the DATA step. Root settings before and after the DATA step are based on the logging configuration file.

ADDITIVITY: "TRUE | FALSE"

specifies whether to pass a log event only to the appender that is associated with the logger or to all the appenders in the logger's hierarchy.

Restriction: ADDITIVITY can be modified for a logger only if the logger's IMMUTABILITY option in the logging configuration file is set to FALSE. If you cannot modify a logger's ADDITIVITY option, contact your system administrator.

Tip: You can also specify this optional argument by using the “[ADDITIVITY Attribute](#)” on page 167 after the logger instance has been created.

APPENDERREF: "appender-name"

specifies the name of the appender to which log events are passed.

Requirement: The appender name must already exist. Appender names are created by using the DECLARE Statement, Appender Object or are defined in a logging configuration file.

Interaction: If the ADDITIVITY argument is set to TRUE, the log events are also passed to all the appenders that are associated with the logger's hierarchy.

Tips:

You can specify more than one appender for each logger.

You can also specify this optional argument by using the APPENDERREF attribute after the logger instance has been created. For more information, see “[APPENDERREF Attribute](#)” on page 168.

See: “[DECLARE Statement, Appender Object](#)” on page 170

LEVEL: "level"

specifies the level at which a logging request is applied for the specified logger object. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Restriction: LEVEL can be modified for a logger only if the logger’s IMMUTABILITY option in the logger configuration file is set to FALSE. If you cannot modify a logger’s LEVEL option, contact your system administrator.

Requirement: The level must be enclosed in double quotation marks.

Tip: You can also specify this optional argument by using the “[LEVEL Attribute](#)” on page 177 after the logger instance has been created.

Details

Logger Names

A logger instance is said to be an ancestor of another logger instance if the logger instance name, followed by a dot, is the prefix of the other logger.

In the following example, IOM is the parent logger, IOM is an ancestor of the APP logger, and both IOM and IOM.APP are ancestors of the WORKSPACE logger.

```
logobj.name="IOM";
logobj.name="IOM.APP";
logobj.name="IOM.APP.WORKSPACE";
```

The hierarchical organization of loggers enables them to inherit log event levels and appenders from their ancestors.

Additivity

By default, each log event is passed to the appenders that are associated with the logger and to the appenders that are associated with the logger's hierarchy. This is the meaning of the term *appender additivity*.

For example, by default, when a log event is processed by the logger Iom.App.Workspace, the log message is also directed to the appenders that are referenced in the Iom.App and Iom loggers. If ADDITIVITY=FALSE, the log message is directed only to the appenders that are referenced by Iom.App.Workspace.

Levels

A logging request is applied if its level is greater than the level of the logger. Otherwise, the logging request is ignored. Loggers that do not have an explicitly assigned level inherit their level from the hierarchy. For more information about the logging levels, see “Logging Thresholds” on page 16.

Example

The following example creates a logger object, `logobj`.

```
data _null_;
  if _n_ = 1 then do;
    declare appender appobj("myappd", "FileRefAppender", "fileref=myfref");
    appobj.threshold="trace";
    declare logger logobj("mylog");
    logobj.appenderref="myappd";
  end;
  logobj.level="trace";
  logobj.debug("Test debug message");
  logobj.level="info";
  logobj.info("Test info message");
run;
```

See Also

Attributes:

- “[ADDITIVITY Attribute](#)” on page 167
- “[APPENDERREF Attribute](#)” on page 168
- “[LEVEL Attribute](#)” on page 177

Statements:

- “[DECLARE Statement, Appender Object](#)” on page 170

ERROR Method

Logs an ERROR message if the specified logger accepts ERROR messages.

Applies to: logger object

Syntax

```
object.ERROR ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the error level.

Requirement: The message must be enclosed in double quotation marks.

Details

The error level designates error events. The application might be able to continue running. For more information about the logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates an error message for the logger.

```
data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
  end;
  logobj.error("Test error message");
run;
```

See Also

Methods:

- [“DEBUG Method” on page 169](#)
- [“FATAL Method” on page 175](#)
- [“INFO Method” on page 176](#)
- [“TRACE Method” on page 178](#)
- [“WARN Method” on page 179](#)

FATAL Method

Logs a FATAL message if the specified logger accepts FATAL messages.

Applies to: logger object

Syntax

```
object.FATAL ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the fatal level.

Requirement: The message must be enclosed in double quotation marks.

Details

The fatal level designates very severe error events that might cause the application to end abruptly. For more information about logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates a fatal message for the logger.

```
data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
  end;
  logobj.fatal("Test fatal message");
run;
```

See Also

Methods:

- [“DEBUG Method” on page 169](#)
- [“ERROR Method” on page 174](#)
- [“INFO Method” on page 176](#)
- [“TRACE Method” on page 178](#)
- [“WARN Method” on page 179](#)

INFO Method

Logs an INFO message if the specified logger accepts INFO messages.

Applies to: logger object

Syntax

```
object.INFO ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the info level.

Requirement: The message must be enclosed in double quotation marks.

Details

The info level designates informational events that highlight the progress of an application at a coarse-grained level. For more information about logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates an info message for the logger.

```
data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
```

```

end;
logobj.info("Test info message");
run;

```

See Also

Methods:

- [“DEBUG Method” on page 169](#)
- [“ERROR Method” on page 174](#)
- [“FATAL Method” on page 175](#)
- [“TRACE Method” on page 178](#)
- [“WARN Method” on page 179](#)

LEVEL Attribute

Defines the level of message that is accepted by the specified logger.

Applies to: logger object

Syntax

```
object.LEVEL="level";
```

Required Arguments

object

specifies the name of the logger object.

level

specifies the level at which a logging request is applied for the specified logger object. Valid values are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Requirement: The level must be enclosed in double quotation marks.

Details

A logging request is applied if its level is greater than the level of the logger. Otherwise, the logging request is ignored. Loggers without an explicitly assigned level inherit their level from the hierarchy. For more information about the logging levels, see [“Logging Thresholds” on page 16](#).

Note: You can specify the logger level in the logger's constructor by using the DECLARE statement. For more information, see [“DECLARE Statement, Logger Object” on page 172](#).

Example

The following code sets the attribute level to trace.

```

data _null_;
  if _n_ = 1 then do;
    declare logger logobj("mylog");

```

```

end;
logobj.additivity="false";
logobj.level="trace";
run;

```

Alternatively, you can set the level attribute in the DECLARE statement constructor.

```

data _null_;
  if _n_ = 1 then do;
    declare logger logobj("mylog", additivity:"false", level:"trace");
  end;
run;

```

See Also

Statements:

- [“DECLARE Statement, Logger Object” on page 172](#)

TRACE Method

Logs a TRACE message if the specified logger accepts TRACE messages.

Applies to: logger object

Syntax

```
object.TRACE ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the trace level.

Requirement: The message must be enclosed in double quotation marks.

Details

The trace level designates finer-grained informational events than DEBUG. For more information about logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates a trace message for a logger.

```

data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
  end;
  logobj.trace("Test trace message");
run;

```

See Also

Methods:

- [“DEBUG Method” on page 169](#)
- [“ERROR Method” on page 174](#)
- [“FATAL Method” on page 175](#)
- [“INFO Method” on page 176](#)
- [“WARN Method” on page 179](#)

WARN Method

Logs a WARN message if the specified logger accepts WARN messages.

Applies to: logger object

Syntax

```
object.WARN ("message");
```

Required Arguments

object

specifies the name of the logger object.

message

specifies the message to write at the warn level.

Requirement: The message must be enclosed in double quotation marks.

Details

The warn level designates potentially harmful situations. For more information about logging levels, see [“Logging Thresholds” on page 16](#).

Example

The following example creates a warn message for a logger.

```
data _null_;  
  if _n_ = 1 then do;  
    declare logger logobj("testlog");  
  end;  
  logobj.warn("Test warn message");  
run;
```

See Also

Methods:

- [“DEBUG Method” on page 169](#)
- [“ERROR Method” on page 174](#)

- [“FATAL Method” on page 175](#)
- [“INFO Method” on page 176](#)
- [“TRACE Method” on page 178](#)

Part 4

Appendix

<i>Appendix 1</i>	
Audit Messages for SAS Library Access	183

Appendix 1

Audit Messages for SAS Library Access

About Audit Messages for SAS Library Access	183
Sample: XML Logger and Appender	184
Sample Logger for Auditing SAS Library Access	184
Sample Appender for Auditing SAS Library Access	184
Sample Code to Look for Log Files and Build an Audit Report Data Set	185

About Audit Messages for SAS Library Access

The logging facility provides a logger, `Audit.Data.Dataset.Open`, to audit access to any SAS library, including database tables that have been assigned by a `LIBNAME` statement. You can obtain the following information about a SAS library:

- the libref
- the engine associated with the library
- the library member name
- the library member type, such as catalog or data set
- the mode that the library was opened for: INPUT, OUTPUT, UPDATE
- the path to the library or to a database table

Note: Audit messages are not available for in-database tables, filerefs, or SAS cubes.

Based on the XML configuration files that SAS provides, auditing messages appear when the logging threshold is TRACE.

To configure the XML configuration file for auditing SAS libraries, you specify a path and filename for the audit log in the appender `fileNamePattern` parameter. Then, specify a conversion pattern in the `conversionPattern` parameter.

Here is an example audit path and filename:

```
SAS-configuration-directory/Levl/
    SAS-application/server-name/Logs/
    Audit.Library_server_%d_%S{hostname}_%S{pid}.log"
```

In the appender `conversionPattern` parameter, you specify the type of information that you want the message to contain using the `%E` conversion character. The following table lists the conversion character syntax for the information that you can obtain about a SAS library:

Information Type	Conversion Pattern
libref	%E{Audit.Dataset.Libref}
engine	%E{Audit.Dataset.Engine}
member name	%E{Audit.Dataset.Member}
member type	%E{Audit.Dataset.Memtype}
open mode	%E{Audit.Dataset.Openmode}
path	%E{Audit.Dataset.Path}

Here is a sample conversion pattern:

```
DateTime=%d Userid=%u Libref=%E{Audit.Dataset.Libref}
Engine=%E{Audit.Dataset.Engine} Member=%E{Audit.Dataset.Member}
MemberType=%E{Audit.Dataset.Memtype} OpenMode=%E{Audit.Dataset.Openmode}
Path=%E{Audit.Dataset.Path}
```

Here is a message from the audit log:

```
Userid=user1@deptProg, Libref=MULTI, Engine=V9, Member=myTestData,
MemberType=DATA, OpenMode=INPUT, Path=c:\temp
```

For a sample logger and appender, see [“Sample: XML Logger and Appender”](#) on page 184.

See Also

[“E Conversion Character”](#) on page 109

Sample: XML Logger and Appender

Sample Logger for Auditing SAS Library Access

Here is a sample logger for auditing access to SAS libraries:

```
<!-- Audit.Data.Dataset.Open logger definition -->

<logger name="Audit.Data.Dataset.Open" additivity="false">
  <appender-ref ref="AuditLibraryFile"/>
  <level value="Trace"/>
</logger>
```

Sample Appender for Auditing SAS Library Access

Here is a sample appender for auditing access to SAS libraries:

```
<!-- Audit.Data.Dataset File Appender Defintion -->

<appender name="AuditLibraryFile" class="FileAppender">
```

```

<param name="Append" value="true"/>
<param name="ImmediateFlush" value="true"/>
<param name="fileNamePattern" value=
    "SAS-configuration-directory/Levl/
    SAS-application/server-name/Logs/
    Audit.Library_server_%d_%S{hostname}_%S{pid}.log"/>
<layout>
  <param name="ConversionPattern"
    value="DateTime=%d Userid=%u Libref=%E{Audit.Dataset.Libref}
    Engine=%E{Audit.Dataset.Engine} Member=%E{Audit.Dataset.Member}
    MemberType=%E{Audit.Dataset.Memtype}
    OpenMode=%E{Audit.Dataset.Openmode} Path=%E{Audit.Dataset.Path}"/>
</layout>
</appender>

```

Sample Code to Look for Log Files and Build an Audit Report Data Set

Here is a sample SAS program that creates an audit report data set from your audit log data:

```

/* Specify the directory where your log files are saved. */

%let logdir=your-log-directory;

/* Specify a directory to save the audit data. */

libname audit "your-audit-data-directory";

/* Expand a passed in directory name and find all the filenames. */

%macro findfiles(logdir);
  data filelist (keep=directory logfile hostname pid);
    format directory logfile $512. hostname $80. pid $8.;
    directory="%&logdir.";
    rc=filename("ONE", "&logdir.");
    if rc ne 0 then do;
      msgLine="NOTE: Unable to assign a filename to directory &logdir.";
      put msgLine;
    end;
  else do;
    did=dopen("ONE");
    if did > 0 then do;
      numfiles=dnum(did);
      put numfiles=;
      do i=1 to numfiles;
        logfile=dread(did,i);
        hostname=scan(logfile,-3,'_');
        pid=scan(logfile,-2,'_');
        output;
      end;
    end;
  end;
%macroend;

```

```

        /* close the open filename and data set pointer */

        rc=filename("ONE");
        did=dclose(did);
    end;
run;
%mend;

/* Read through a data set of Directory name and filenames and read the audit logs.*/

%macro readfiles(list);
    %let dsid = %sysfunc(open(&list));
    %if &dsid %then %do;
        %syscall set(dsid);
        %let nobs = %sysfunc(attrn(&dsid,nlobs));
        %do i=1 %to &nobs;
            %let rc=%sysfunc(fetch(&dsid));
            %let ldir=%sysfunc(getvarc(&dsid,%sysfunc(varnum(&dsid,DIRECTORY))));
            %let lfile=%sysfunc(getvarc(&dsid,%sysfunc(varnum(&dsid,LOGFILE))));
            %let host=%sysfunc(getvarc(&dsid,%sysfunc(varnum(&dsid,HOSTNAME))));
            %let pid=%sysfunc(getvarc(&dsid,%sysfunc(varnum(&dsid,PID))));
            filename auditlog "&ldir.\&lfile.";
        data auditlib;
            infile auditlog recfm=V lrecl=512 pad missover;
            informat DateTime B8601DT23.3; format    DateTime datetime23.3;
            length Userid          $ 80;    label    Userid='Userid';
            length Libref          $ 16;
            length Engine          $ 16;
            length Member          $ 32;
            length MemberType      $ 16;
            length OpenMode        $ 16;
            length Path            $ 4096;
            length Hostname        $ 80;
            length Pid             $ 8;
            input DateTime= Userid= Libref= Engine= Member= MemberType= OpenMode= Label=
                Path=;

            /* Populate values that will come from log filename */

            Hostname=trim("&Hostname.");
            Pid=trim("&Pid.");
        run;

        proc append base=audit.file_opens data=auditlib; run;
        %end;
        %let rc = %sysfunc(close(&dsid));
        %end;
    %mend;

    /* Look for files to process in a directory. */

    %findfiles(&logdir);

    /* Read the log files to produce a data set for reporting. */

```

```
%readfiles(filelist);
```


Glossary

additivity flag

a flag that is associated with a logger. An additivity flag controls whether ancestor loggers receive log events. By default, a log event is passed to the logger that is associated with the event as well as to any ancestor loggers. If a logger's additivity flag is set to false, then log events are not passed to ancestor loggers. For example, if the additivity flag for App.Meta is set to false, then App.Meta.IO events are passed to the App.Meta.IO and App.Meta loggers, but they are not passed to the App logger.

ancestor logger

a logger that is at a higher level in relation to other loggers in the logger hierarchy. For example, the Audit logger is an ancestor logger of Audit.Meta and Audit.Authentication.

appender

a named entity that represents a specific output destination for messages. Destinations include fixed files, rolling files, operating system facilities, and client applications.

appender additivity

a feature that causes each log event to be passed to the appenders that are associated with the logger as well as to appenders that are associated with the logger's ancestor loggers. For example, App.Meta.IO events are passed to appenders that are associated with App.Meta.IO as well as to appenders that are associated with App.Meta and App.

appender reference

an expression that identifies an appender whose destination receives messages for log events for a particular logger.

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.

autocall macro

a macro whose uncompiled source code and text are stored in an autocall macro library. Unlike a stored compiled macro, an autocall macro is compiled before execution the first time it is called.

conversion character

a single character that represents a data item that is generated in a log event. For example, d specifies the date of the event and t identifies the thread that generated the event.

conversion pattern

an expression that specifies an appender definition's pattern layout. A conversion pattern consists of a combination of user-supplied literal text and conversion specifiers.

conversion specifier

an expression in a conversion pattern that consists of a percent sign (%), a conversion character, and optional format modifiers.

descendant logger

a logger that is at a lower level in relation to other loggers in the logger hierarchy. For example, Audit.Meta and Audit.Authentication are descendant loggers of the Audit logger.

diagnostic level

the degree of severity of a log event that can occur during SAS program processing. Examples of levels are trace, debug, informational, warning, error, and fatal. Short form: level.

DTD

Document Type Definition. A file that specifies how the markup tags in a group of SGML or XML documents should be interpreted by an application that displays, prints, or otherwise processes the documents.

filter

See package filter.

format modifier

an optional set of characters in a conversion specifier that controls the field width, padding, and justification of the specified data item in log output.

Integrated Object Model

the set of distributed object interfaces that make SAS software features available to client applications when SAS is executed as an object server. Short form: IOM.

Integrated Object Model server

See IOM server.

IOM

See Integrated Object Model.

IOM server

a SAS object server that is launched in order to fulfill client requests for IOM services. Short form: IOM server.

level

See diagnostic level.

log

See SAS log.

log event

an occurrence that is reported by a program for possible inclusion in a log.

logger

a named entity that identifies a message category. Logger names have a hierarchical format that enables you to configure logging at a broad or a fine-grained level.

logging configuration

an XML file or a set of SAS program statements that determines how log events are processed. You use the logging configuration to assign thresholds to loggers, to configure appenders, and to specify which categories and levels of log events are written to each appender.

package filter

specified criteria that are applied to data in order to identify the subset of data for a subsequent operation, such as continued processing.

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.

planned deployment

a method of installing and configuring a SAS business intelligence system. This method requires a deployment plan that contains information about the different hosts that are included in the system and the software and SAS servers that are to be deployed on each host. The deployment plan then serves as input to the SAS Deployment Wizard.

root logger

the highest-level logger in the logger hierarchy. In a logging configuration, all other loggers inherit the root logger's attributes.

SAS console log

a file that contains information, warning, and error messages if the SAS log is not active. The SAS console log is normally used only for fatal system initialization errors or for late-termination messages.

SAS Deployment Wizard

a cross-platform utility that installs and initially configures many SAS products. Using a SAS installation data file and, when appropriate, a deployment plan for its initial input, the wizard prompts the customer for other necessary input at the start of the session, so that there is no need to monitor the entire deployment.

SAS log

a file that contains a record of the SAS statements that you enter, as well as messages about the execution of your program.

threshold

the lowest event level that is processed. Log events whose levels are below the threshold are ignored.

Index

Special Characters

%LOG4SAS_APPENDER autocall macro 145

%LOG4SAS_DEBUG autocall macro 150

%LOG4SAS_ERROR autocall macro 153

%LOG4SAS_FATAL autocall macro 153

%LOG4SAS_INFO autocall macro 151

%LOG4SAS_LOGGER autocall macro 147

%LOG4SAS_TRACE autocall macro 149

%LOG4SAS_WARN autocall macro 152

%LOG4SAS autocall macro 145

A

accessibility 4

additivity
 See appender additivity

ADDITIVITY attribute 167

appender additivity 149, 162
 logger objects and 173
 passing log events and 167

appender categories
 creating 139

appender classes
 formatting log messages for 95

appender objects
 creating an instance of 170
 declaring 170
 FileRefAppender and 171
 initializing data for 170
 patterns and 171
 thresholds and 171

APPENDERREF attribute 168

appenders 11
 associating with loggers 135
 ConsoleAppender 45
 creating and using in SAS programs 135
 creating fileref appenders 159
 DBAppender 46
 defining 145
 definition 5
 FileAppender 52
 FileRefAppender 141, 160, 171
 FilteringAppender 55
 in SAS language 15
 IOMServerAppender 58
 JavaAppender 60
 JMSAppender 67
 names of 146, 160, 168, 171
 patterns and 146
 processing in DATA step 160
 referencing in loggers 15
 RollingFileAppender 72
 SAS appenders for server logging 14
 syntax 12
 UNXFacilityAppender 80
 WindowsEventAppender 82
 XML elements for configuring 12
 xml sample, SAS library access 184
 ZOSFacilityAppender 83
 ZOSWtoAppender 86

application messages
 directing to z/OS consoles 86

ARM appender 41
 example 44
 syntax 42

attributes 137, 165

auditing
 SAS libraries 183

autocall macros 141
 examples 142
 FileRefAppender and 141
 initializing the logging environment 134, 145

B

best practices for SAS server logging 19

C

client applications
 adjusting logging levels in 19
 viewing logging messages in 19
 Component Interface 165
 component objects 165
 dot notation and 166
 configuration
 changing 20
 for SAS server logging 18
 logging configuration 6
 XML elements for configuring
 appenders 12
 configuration file 33
 error messages 35
 restrict modification of 8, 38
 sample files 34
 structure of 33
 syntax conventions 29
 typographical conventions 29
 XML 24
 XML elements for 30
 ConnectionString parameter of
 DBAppender 89
 console
 directing application messages to 86
 writing messages to 45
 ConsoleAppender 45
 conversion patterns 96
 conversion specifiers 96
 creating in a SAS program 172
 current console
 writing messages to 45

D

DATA step
 logging facility in 155
 processing appenders in 160
 processing loggers in 163
 DATA step Component Interface 165
 DATA step component objects 165
 dot notation and 166
 dates
 filtering for specific date 123
 DB2 tables
 connecting to 89
 writing messages to 49
 DBAppender 46
 specifying connection options for 89
 DEBUG level 16, 20
 DEBUG messages 150, 169

DEBUG method 169

DECLARE statement, Appender object
 170

DECLARE statement, Logger object 172

DenyAllFilter 126

diagnostic levels

 UNIX 81

dot notation

 component objects and 166

 definition 166

 syntax 166

E

environment
 initializing for autocall macros 145
 setting up 6
 ERROR level 16
 ERROR messages 153, 163, 174
 ERROR method 174
 event logging
 UNIX 80
 Windows 82
 z/OS 83

F

FATAL level 16
 FATAL messages 153, 175
 FATAL method 175
 FileAppender 52
 fileref appenders 159
 patterns and 160
 thresholds and 160
 FileRefAppender
 appender objects and 171
 autocall macros and 141
 functions and 160
 files
 writing messages to 52
 writing messages to rolling files 72
 filter classes 17, 121
 filtering policy 18, 122
 FilteringAppender 55
 filters 5, 16
 DenyAllFilter 126
 denying messages not meeting
 specifications 126
 examples 123
 for specific date 123
 for specific user's error messages 123
 LevelMatchFilter 126
 LevelRangeFilter 127
 log events for a single threshold 126
 message filtering 17
 overview 121

- range of message thresholds 127
- syntax 124
- format modifiers
 - examples 118
- formatted log event 96
- formatting messages 17, 95
- functions 155
 - examples 156
 - FileRefAppender and 160
 - LOG4SAS_APPENDER 159
 - LOG4SAS_LOGEVENT 163
 - LOG4SAS_LOGGER 161

H

- hierarchical logger names 9

I

- INFO level 16
- INFO messages 151, 176
- INFO method 176
- IOM servers
 - writing messages to 58
- IOMServerAppender 58

J

- Java classes
 - writing messages to 60
- JavaAppender 60
- JMSAppender 67

L

- LEVEL attribute 177
- LevelMatchFilter 126
- LevelRangeFilter 127
- levels 5, 16, 20
 - adjusting in client applications 19
 - adjusting temporarily 20
 - creating loggers and 163
 - defining 177
 - defining loggers, to inherit the level 39
 - defining loggers and 149
 - logger objects and 174
 - UNIX diagnostic levels 81
- log events 5
 - additivity and passing events 167
 - creating in SAS programs 138
 - filtering for a single threshold 126
 - formatted 96
 - passing 168
- LOG4SAS_APPENDER function 159
- LOG4SAS_LOGEVENT function 163
- LOG4SAS_LOGGER function 161

- LOGAPPLNAME= system option 23
- logconfig_trace.xml file 20
- LOGCONFIGLOC= system option 24, 134

- logger categories 139

- logger objects
 - additivity and 173
 - creating an instance of 172
 - declaring 172
 - initializing data for 172
 - levels and 174

- loggers 7

- associating appenders with 135
- creating in a SAS program 136, 147, 161
- defining to inherit the level 39
- defining to log error messages for application in production 39
- definition 5
- hierarchical logger names 9
- in SAS language 11
- logging messages with a specific logger 163
- names of 148, 162, 164, 173
- processing in DATA step 163
- referencing appenders in 15
- SAS server logger names 9
- updating attributes 137
- XML elements for configuring 8
- xml sample, SAS library access 184

- logging configuration 6

- logging configuration file
 - See [configuration file](#)

- logging facility 4

- compared with SAS log 4
- enabling for SAS server logging 21
- enabling in SAS programs 21
- in DATA step 155
- in SAS language 133
- initializing autocall macros 134
- initializing for SAS programs 134
- terminology 5
- users of 4

- logging process 6

- setting up 6

- logging thresholds

- See [thresholds](#)

M

- macros

- See also [autocall macros](#)
- write-to-operator (WTO) 86

- message queues

- writing messages to 67

- messages

- directing to z/OS consoles 86
- error messages for applications in
 - production 39
- filtering 17
- formatting 17
- formatting for appender classes 95
- logging with a specific logger 163
- viewing in client applications 19
- writing to current console 45
- writing to custom Java classes 60
- writing to DBMS tables 46
- writing to IOM servers 58
- writing to message queues 67
- writing to rolling files 72
- writing to SAS tables 46
- writing to specified file 52

methods 165

- DEBUG 169
- ERROR 174
- FATAL 175
- INFO 176
- TRACE 178
- WARN 179

N

names

- appender names 146, 160, 168, 171
- hierarchical logger names 9
- logger names 148, 162, 164, 173
- SAS server logger names 9
- SAS session names 22, 23

O

ODBC-compliant databases

- connecting to 90
- writing messages to 50

operating system consoles

- directing application messages to 86

Oracle tables

- connecting to 92
- writing messages to 50

P

passing log events 167, 168

pattern layouts 17

- appender objects and 171
- defining appenders and 146
- definition 6
- example 106
- fileref appenders and 160
- overview 95

R

referencing appenders in loggers 15

rolling files

- writing messages to 72

RollingFileAppender 72

S

SAS Data Integration Studio

- logging messages and levels 19

SAS language

- appenders in 15
- loggers in 11
- logging facility in 133
- message categories in 137

SAS libraries

- auditing 183

SAS log

- compared with logging facility 4

SAS Management Console

- logging messages and levels 19

SAS programs

- creating and using appenders in 135
- creating log events in 138
- creating loggers in 136
- enabling logging facility in 21
- initializing logging facility for 134

SAS server logging

- adjusting logging levels in client applications 19
- best practices for 19
- enabling logging facility for 21
- initial configuration for 18
- viewing logging messages in client applications 19

SAS servers

- appenders for 14
- logger names 9

SAS session names 22, 23

SAS tables

- connecting to 93
- writing messages to 51

syntax conventions 8, 12, 29, 166

T

tables

- writing messages to 46

Teradata tables

- connecting to 93
- writing messages to 51

terminology 5

thresholds 16

- appender objects and 171
- defining appenders and 147
- definition 6

- fileref appenders and 160
- filtering log events for a single threshold 126
- filters for a range of 127
- TRACE level 16, 20
- TRACE messages 149, 178
- TRACE method 178
- typographical conventions 29

U

- UNIX
 - diagnostic levels 81
 - event logging on 80
 - writing messages to current console 45
- UNXFacilityAppender 80
- updates
 - updating logger attributes 137
- users 4
 - filter for specific user's error messages 123

W

- WARN level 16

- WARN messages 152, 179
- WARN method 179
- Windows
 - event logging on 82
 - writing messages to current console 45
- WindowsEventAppender 82
- write-to-operator (WTO) service macro 86

X

- XML configuration file 24
- XML elements
 - for configuration file 30
 - for configuring appenders 12

Z

- z/OS
 - directing application messages to console 86
 - event logging on 83
- ZOSFacilityAppender 83
- ZOSWtoAppender 86

