



THE
POWER
TO KNOW.

SAS[®] 9.2 Logging Configuration and Programming Reference



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® 9.2 Logging: Configuration and Programming Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.2 Logging: Configuration and Programming Reference

Copyright © 2009, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59994-434-0

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

For a hard-copy 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.

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 printing, February 2009

1st electronic book, March 2009

2nd electronic book, August 2009

3rd electronic book, May 2010

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

<i>What's New</i>	<i>vii</i>
Overview	vii
General Enhancements	vii

PART 1 SAS Logging 1

Chapter 1 △ The SAS Logging Facility	3
Accessibility Features of the SAS Logging Facility	3
Overview of the SAS Logging Facility	4
Logging Facility Terminology	5
How the Logging Facility Works	6
Loggers	7
Appenders	10
Logging Thresholds	14
Formatting Messages	15
Message Filtering	16
Using the SAS Logging Facility in the SAS Intelligence Platform	17
Chapter 2 △ Enabling the SAS Logging Facility	19
Enabling the Logging Facility for SAS Server Logging	19
Enabling the Logging Facility in SAS Programs	19
Naming a SAS Session	19

PART 2 XML Configuration Overview and Reference 23

Chapter 3 △ Overview of the Logging Configuration File	25
Typographical Conventions	25
Syntax Conventions	25
XML Elements for Configuring SAS Logging	26
Structure of the Logging Configuration File	28
Sample Configuration Files	29
Chapter 4 △ Logger Reference	31
SAS Logger Overview and Syntax	31
Chapter 5 △ Appender Reference	33
ARMAppender	34
ConsoleAppender	38
FileAppender	39
FilteringAppender	43
IOMServerAppender	46
RollingFileAppender	48

sLogAppender	56
UNXFacilityAppender	56
WindowsEventAppender	59
ZOSFacilityAppender	60
ZOSWtoAppender	63

Chapter 6	△	Pattern Layout	65
Overview of Pattern Layouts			65
Syntax for a Pattern Layout			65
Pattern Layout Syntax Description			66
Conversion Characters			68
Format Modifiers			73
Examples			73

Chapter 7	△	Filters	77
Overview of Filters			77
Syntax for Filters			79
AndFilter			79
DenyAllFilter			81
LevelMatchFilter			81
LevelRangeFilter			82
StringMatchFilter			83
Filter Examples			84

PART 3 The Logging Facility for SAS Programs 87

Chapter 8	△	The SAS Logging Facility in the SAS Language	89
Overview of the SAS Logging Facility in the SAS Language			89
Initializing the SAS Logging Facility for SAS Programs			90
Creating and Using Appenders in a SAS Program			90
Creating Loggers in a SAS Program			91
Creating Log Events in a SAS Program			94
Example of Creating Logger and Appender Categories			94

Chapter 9	△	Autocall Macro Reference	97
Using Autocall Macros to Log Messages			97
Dictionary			98
Example of Using Autocall Macros to Log Messages			108

Chapter 10	△	Function Reference	111
Using the Logging Facility Functions in the DATA Step			111
Dictionary			112
Logging Example Using Functions			117

Chapter 11	△	Logger and Appender Object Language Reference	121
The Logger and Appender Component Object Interface			121
Dot Notation and DATA Step Component Objects			122

Dictionary 123

Appendix 1 \triangle Recommended Reading 137

Recommended Reading 137

Glossary 139

Index 143

What's New

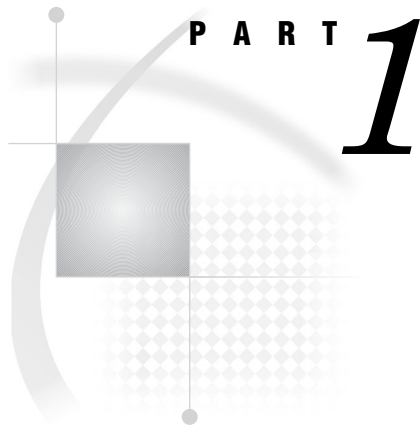
Overview

The logging facility, new with SAS 9.2, 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.

General Enhancements

In the second maintenance release after SAS 9.2, FilteringAppender has been added to the documentation. This appender enables you to filter events to determine whether they should be passed to a referenced appender. The appender can apply a layout to the events before they are passed.

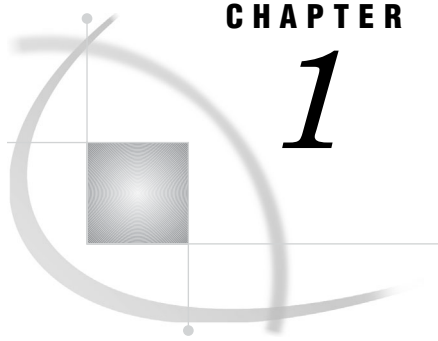
The primary use of FilteringAppender is to specify different layouts for different categories of events that are to appear together in the same log.



SAS Logging

Chapter 1.....**The SAS Logging Facility** 3

Chapter 2.....**Enabling the SAS Logging Facility** 19



CHAPTER

1

The SAS Logging Facility

<i>Accessibility Features of the SAS Logging Facility</i>	3
<i>Overview of the SAS Logging Facility</i>	4
<i>What Is the Logging Facility?</i>	4
<i>Who Uses the Logging Facility?</i>	4
<i>Comparing the SAS Logging Facility and the SAS Log</i>	4
<i>Logging Facility Terminology</i>	5
<i>How the Logging Facility Works</i>	6
<i>Setting Up the Logging Process</i>	6
<i>The Logging Process</i>	6
<i>Loggers</i>	7
<i>What Is a Logger?</i>	7
<i>XML Elements for Configuring Loggers</i>	7
<i>Hierarchical Logger Names</i>	8
<i>SAS Server Logger Names</i>	9
<i>Loggers in the SAS Language</i>	10
<i>Appenders</i>	10
<i>Appender Overview</i>	10
<i>XML Elements for Configuring Appenders</i>	11
<i>General Appender Syntax</i>	11
<i>SAS Appenders for Server Logging</i>	13
<i>Appenders in the SAS Language</i>	13
<i>Referencing Appenders in a Logger</i>	14
<i>Logging Thresholds</i>	14
<i>Formatting Messages</i>	15
<i>Message Filtering</i>	16
<i>Using the SAS Logging Facility in the SAS Intelligence Platform</i>	17
<i>About the Initial Logging Configuration for SAS Servers</i>	17
<i>Viewing SAS Logging Messages and Adjusting Logging Levels in Client Applications</i>	17
<i>Best Practices for SAS Server Logging</i>	18

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 9.2 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 files, operating system facilities, and client applications. 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 server.

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.

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 not only to the SAS log, but also 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, and client applications. 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.Mgmt. 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.

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.

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 14 and "Appendors" on page 10.

XML Elements for Configuring Loggers

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

```
<logger name="logger-name">
  <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.

```
<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 (.).

As an 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 that are related to user authentication (including accepted and rejected authentication requests) and to security administration (including the administration of users, groups, and access controls).

IOM

processes log events for servers that use the Integrated Object Model (IOM) workspace interface. 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

In most cases, the categories are self-explanatory. 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.

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.

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
- DCL Logger object constructor in a DATA step

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

- “%LOG4SAS_LOGGER Autocall Macro” on page 101
- “LOG4SAS_LOGGER Function” on page 113
- “DECLARE Statement, Logger Object” on page 128

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.

Appendixes

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

For a complete list and description of the SAS server appenders, see “SAS Appenders for Server Logging” on page 13.

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 Chapter 5, “Appender Reference,” on page 33 and “Creating and Using Appenders in a SAS Program” on page 90.

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
- FileAppender
- FilteringAppender
- IOMServerAppender
- RollingFileAppender

- `sLogAppender`
- `UNXFacilityAppender`
- `WindowsEventAppender`
- `ZOSFacilityAppender`
- `ZOSWtoAppender`

For more information about logging facility appenders, see Chapter 5, “Appender Reference,” on page 33.

`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 5, “Appender Reference,” on page 33 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, the time, and the message, respectively, in the log.

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

For more information, see “Formatting Messages” on page 15.

`<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

For more information, see Chapter 7, “Filters,” on page 77.

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

ARMAppender processes all Application Response Measurement (ARM) messages that are submitted by an external ARM agent or by the SAS ARM agent. See "ARMAppender" on page 34.

ConsoleAppender

ConsoleAppender writes messages to the UNIX and Windows operating system consoles. See "ConsoleAppender" on page 38.

FileAppender

FileAppender writes messages to the specified file in the specified path. See "FileAppender" on page 39.

FilteringAppender

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. See "FilteringAppender" on page 43.

IOMServerAppender

IOMServerAppender commits messages from any IOM server to a volatile runtime cache. See "IOMServerAppender" on page 46.

RollingFileAppender

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. See "RollingFileAppender" on page 48.

sLogAppender

sLogAppender is a reserved appender. You should not define new instances of this appender. See "sLogAppender" on page 56.

UNXFacilityAppender

UNXFacilityAppender writes messages to the syslogd logging facility in UNIX operating systems. See "UNXFacilityAppender" on page 56.

WindowsEventAppender

WindowsEventAppender writes messages to the Windows Event log. See "WindowsEventAppender" on page 59.

ZOSFacilityAppender

ZOSFacilityAppender enables multiple instances of SAS in the z/OS operating system to write messages to a common location. See "ZOSFacilityAppender" on page 60.

ZOSWtoAppender

ZOSWtoAppender directs SAS application messages to the z/OS operating system console. See "ZOSWtoAppender" on page 63.

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 90.

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 7, "Filters," on page 77.

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 might allow the application to continue running.
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: The conversion patterns that are used in the SAS logging facility and in the C language PRINTF statement are similar.

In addition to the set of pattern layouts that are used by the appender classes in the SAS logging facility, a different set of pattern layouts is used by ARMAppender. For more information, see ARMAppender Pattern Layouts 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; (%F:%L); %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, the log file, the line number in the calling program that specified the logging request, 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 6, “Pattern Layout,” on page 65.

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
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 that 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 7, “Filters,” on page 77.

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 an alternative logging configuration file called **logconfig_trace.xml** that can be used for troubleshooting.

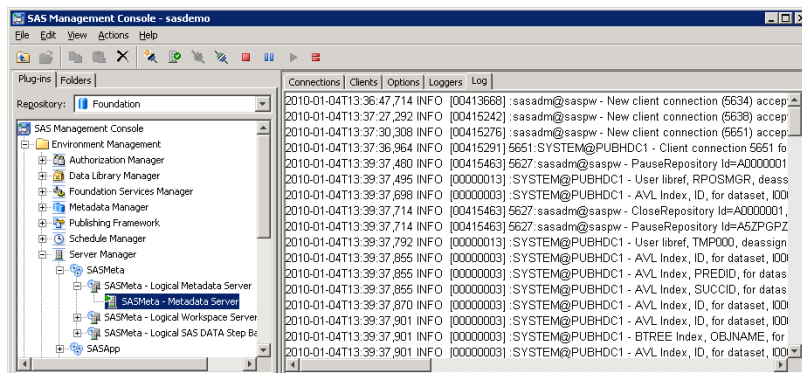
For more information, see the following topics in the *SAS Intelligence Platform: System Administration Guide*:

- “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:

Connections Clients Options Loggers Log		
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>.

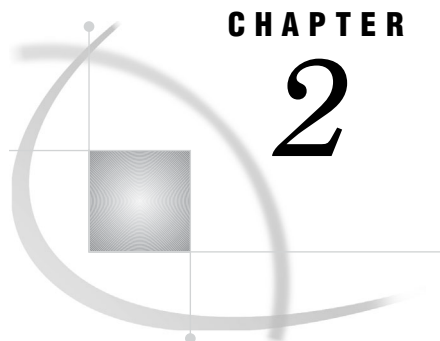
Best Practices for SAS Server Logging

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*.
- *SAS Interface to Application Response Measurement (ARM): Reference*, which 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 19

Enabling the Logging Facility in SAS Programs 19

Naming a SAS Session 19

LOGAPPLNAME= System Option 20

LOGCONFIGLOC= System Option 20

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 20.

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 macros 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 20 and the S conversion character in “Conversion Characters” on page 68.

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

“Conversion Characters” on page 68

SYSLOGAPPLNAME Automatic Macro Variable in *SAS Macro Language: Reference*

LOGCONFIGLOC= System Option

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

Alias: LOGCFGLOC=

Valid in: configuration file, SAS invocation, spawner invocation

Category: Environment control: Initialization and operation

PROC OPTIONS GROUP= EXECMODES

Syntax

LOGCONFIGLOC=*file-specification*

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.

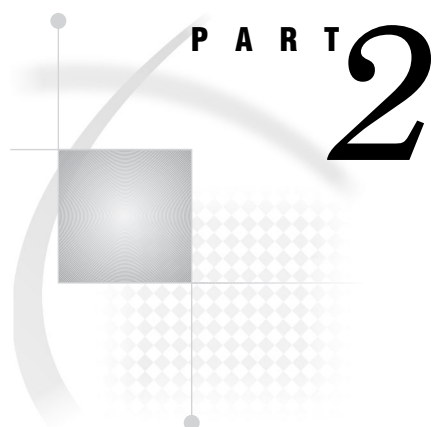
Details

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.

Examples

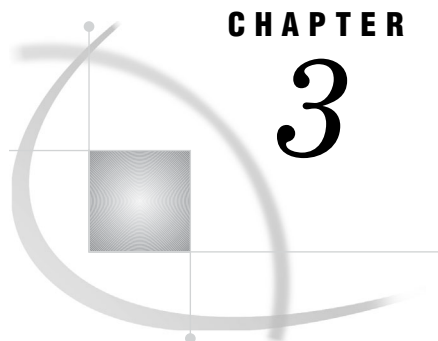
The following example shows the use of the LOGCONFIGLOC= system option:

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

XML Configuration Overview and Reference

<i>Chapter 3</i>	Overview of the Logging Configuration File	25
<i>Chapter 4</i>	Logger Reference	31
<i>Chapter 5</i>	Appender Reference	33
<i>Chapter 6</i>	Pattern Layout	65
<i>Chapter 7</i>	Filters	77



CHAPTER

3

Overview of the Logging Configuration File

<i>Typographical Conventions</i>	25
<i>Syntax Conventions</i>	25
<i>XML Elements for Configuring SAS Logging</i>	26
<i>Structure of the Logging Configuration File</i>	28
<i>Sample Configuration Files</i>	29

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:

<i>italics</i>	identifies arguments or values that you supply. Items in italics can represent user-supplied values that are either one of the following: <ul style="list-style-type: none"> □ nonliteral values that are assigned to an argument (for example, value="<i>column-parameter</i>") □ nonliteral arguments (for example, <i><filter-definitions></i>)
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	For repeated elements, the -1 after an element indicates the first element.
-n	The ellipsis before an element and the -n after the same element indicates that the element can be repeated any number of times.
...	
" "	Single 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 3.1 SAS Logging XML Configuration Elements

Element Name and Description	Element Characteristics
<code><?xml ?></code> is the first XML element in the configuration file	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: one <input type="checkbox"/> Required attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>version="1.0"</code> <input type="checkbox"/> Optional attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>encoding= "encoding-specification "</code> specifies a language encoding. For more information, see the <i>SAS National Language Support (NLS): Reference Guide</i>.
<code><logging:configuration></code> defines the logging message category and the message category default values <code></logging:configuration></code>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: one <input type="checkbox"/> Required attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>xmlns:logging="http://support.sas.com/xml/logging/1.0/"</code> <input type="checkbox"/> Optional attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>threshold="level"</code> 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 14. <input type="checkbox"/> <code>debug="TRUE FALSE"</code> 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. <input type="checkbox"/> Optional child elements: number of instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><appender></code>: zero or more <input type="checkbox"/> <code><logger></code>: zero or more <input type="checkbox"/> <code><root></code>: zero or one

<p><code><appender></code></p> <p>defines a destination for log messages, filters to limit log messages, and the format of the log message</p> <p><code></appender></code></p>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: zero or multiple <input type="checkbox"/> Required attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>name="appender-name"</code> is a user-defined name for the appender. <input type="checkbox"/> <code>class="class-name"</code> specifies the type of appender (For more information, see “SAS Appenders for Server Logging” on page 13). <input type="checkbox"/> Optional child elements: number of instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><param></code>: zero or multiple <input type="checkbox"/> <code><layout></code>: zero or one <input type="checkbox"/> <code><filter></code>: zero or multiple <input type="checkbox"/> <code><appender-ref></code>: zero or multiple <input type="checkbox"/> <code><rollingPolicy></code>: zero or one; required when the appender class is RollingFileAppender; otherwise, do not include this element.
<p><code><logger></code></p> <p>names a message category, references an appender, and defines a message threshold</p> <p><code></logger></code></p>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: zero or multiple <input type="checkbox"/> Required attribute: <ul style="list-style-type: none"> <input type="checkbox"/> <code>name="logger-name"</code> is a user-defined logger name. <input type="checkbox"/> Optional attribute: <ul style="list-style-type: none"> <input type="checkbox"/> <code>additivity="TRUE FALSE"</code> specifies whether to pass the log event to loggers in a hierarchy. <input type="checkbox"/> Optional child elements: number of instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><level></code>: zero or multiple <input type="checkbox"/> <code><appender-ref></code>: zero or multiple
<p><code><root></code></p> <p>a fixed SAS logger and the highest logger in any logging hierarchy</p> <p><code></root></code></p>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: zero or one <input type="checkbox"/> Attributes: none <input type="checkbox"/> optional child elements: number of instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><level></code>: zero or one <input type="checkbox"/> <code><appender-ref></code>: zero or multiple
<p><code><param /></code></p> <p>defines an appender or a filter parameter and a parameter value</p>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: zero or multiple <input type="checkbox"/> Required attributes: <ul style="list-style-type: none"> <input type="checkbox"/> <code>name="attribute-name"</code> specifies the literal name of an attribute. <input type="checkbox"/> <code>value="attribute-value"</code> specifies either a literal or user-supplied value.
<p><code><layout></code></p> <p>specifies conversion-pattern layouts</p> <p><code></layout></code></p>	<ul style="list-style-type: none"> <input type="checkbox"/> Number of instances: zero or one <input type="checkbox"/> Optional child element: number of instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><param></code>: zero or multiple

<code><level></code>	<input type="checkbox"/> Number of instances: zero or one
defines the message threshold that a logger accepts	<input type="checkbox"/> Required attribute: <ul style="list-style-type: none"> <input type="checkbox"/> <code>value="level"</code> 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 14).

<code><filter></code>	<input type="checkbox"/> Number of instances: zero or one
specifies message-filtering values	<input type="checkbox"/> Required attribute: <ul style="list-style-type: none"> <input type="checkbox"/> <code>class="filter-class"</code> specifies the name of the filter (see “Message Filtering” on page 16).
<code></filter></code>	<input type="checkbox"/> Optional child element: instances <ul style="list-style-type: none"> <input type="checkbox"/> <code><param></code>: zero or multiple

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

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. When you add appenders and loggers, the elements must appear in this order:

- ☐ `<?xml>`
- ☐ `<logging:configuration>`
- ☐ `<appender>`
- ☐ `<logger>`
- ☐ `<root>`

See “XML Elements for Configuring SAS Logging” on page 26 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>
```

```

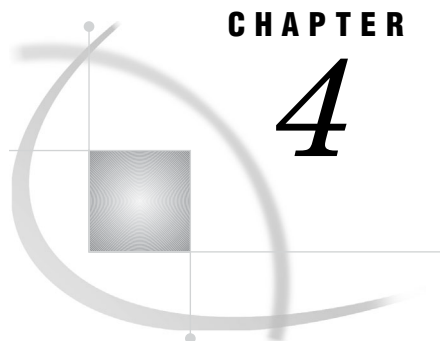
    <layout>
      <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
    </layout>
  </appender>
  <appender class="FileAppender" name="rootAppender">
    <param name="Append" value="true"/>
    <param name="ImmediateFlush" value="true"/>
    <param name="File" value="c:\logs\root\root1.log"/>
  </appender>
  <logger name="log4WServer">
    <level value="info"/>
    <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.



CHAPTER

4

Logger Reference

SAS Logger Overview and Syntax 31

Logger Overview 31

Logger Syntax 31

Logger Syntax Description 31

Logger Examples 32

Example 1: Define a Logger to Log Error Messages for an Application in Production 32

Example 2: Define Loggers That Inherit the Level 32

SAS Logger Overview and Syntax

Logger Overview

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.

In a logging facility XML configuration file and in a SAS program, an appender that is specified as an appender reference in a logger must be defined before the logger is defined.

For more information about loggers, see “Loggers” on page 7.

Logger Syntax

XML Configuration

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

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

Required: Yes

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.

Required: No

Default: None

See: “Logging Thresholds” on page 14

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.

Logger Examples

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>
```

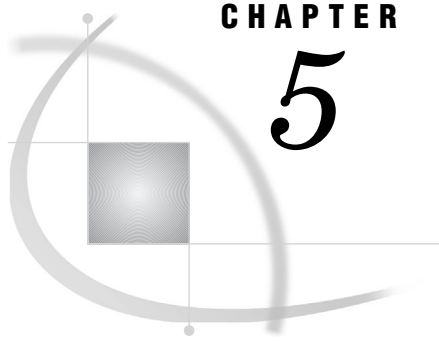
Example 2: Define Loggers That Inherit the Level

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

5

Appender Reference

<i>ARMAppender</i>	34
<i>ARMAppender Overview</i>	34
<i>ARMAppender Syntax</i>	34
<i>ARMAppender Syntax Description</i>	35
<i>ARMAppender Example</i>	37
<i>ARMAppender Usage and Best Practices</i>	38
<i>ConsoleAppender</i>	38
<i>ConsoleAppender Overview</i>	38
<i>ConsoleAppender Syntax</i>	38
<i>ConsoleAppender Syntax Description</i>	39
<i>ConsoleAppender Examples</i>	39
<i>FileAppender</i>	39
<i>FileAppender Overview</i>	40
<i>FileAppender Syntax</i>	40
<i>FileAppender Syntax Description</i>	40
<i>FileAppender Examples</i>	43
<i>Example 1: Appending Messages to a File</i>	43
<i>FileAppender Usage and Best Practices</i>	43
<i>FilteringAppender</i>	43
<i>FilteringAppender Overview</i>	43
<i>FilteringAppender Syntax</i>	44
<i>FilteringAppender Syntax Description</i>	44
<i>FilteringAppender Example</i>	45
<i>FilteringAppender Usage and Best Practices</i>	46
<i>IOServerAppender</i>	46
<i>IOServerAppender Overview</i>	46
<i>IOServerAppender Syntax</i>	47
<i>IOServerAppender Syntax Description</i>	47
<i>IOServerAppender Example</i>	47
<i>IOServerAppender Usage and Best Practices</i>	48
<i>RollingFileAppender</i>	48
<i>RollingFileAppender Overview</i>	48
<i>RollingFileAppender Syntax</i>	48
<i>RollingFileAppender Syntax Description</i>	49
<i>RollingFileAppender Configuration Examples</i>	54
<i>Example 1: Roll Over to a New Log File Every Day</i>	54
<i>Example 2: Roll Over to a New Log When a New Session Begins</i>	55
<i>Example 3: Roll Over to a New Log File When the File Reaches a Specified Size</i>	55
<i>RollingFileAppender Usage and Best Practices</i>	56
<i>sLogAppender</i>	56
<i>UNXFacilityAppender</i>	56

UNXFacilityAppender Overview	57
UNXFacilityAppender Syntax	57
UNXFacilityAppender Syntax Description	57
UNXFacilityAppender Example	57
UNXFacilityAppender Diagnostic Levels	58
UNXFacilityAppender Usage and Best Practices	58
WindowsEventAppender	59
WindowsEventAppender Overview	59
WindowsEventAppender Syntax	59
WindowsEventAppender Syntax Description	59
WindowsEventAppender Example	59
WindowsEventAppender Usage	60
ZOSFacilityAppender	60
ZOSFacilityAppender Overview	60
ZOSFacilityAppender Syntax	60
ZOSFacilityAppender Syntax Description	61
ZOSFacilityAppender Examples	61
Configuring ZOSFacility Appender	61
Defining the Logstream	61
Using the SUBSYS DD Statement	62
Limiting the Output	62
Deleting the Logstream Contents	62
ZOSFacilityAppender Usage and Best Practices	63
ZOSWtoAppender	63
ZOSWtoAppender Overview	63
ZOSWtoAppender Syntax	63
ZOSWtoAppender Syntax Description	63
ZOSWtoAppender Example	64
ZOSWtoAppender Usage and Best Practices	64

ARMAppender

ARMAppender Overview

The Application Response Measurement (ARM) appender 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.

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*.

ARMAppender Syntax

ARMAppender syntax is case sensitive.

XML Configuration

```
<appender class="FileAppender" name="ARM-log-name">
  <param name="File" value="file-name"/>
```

```

<layout>
<param name="ConversionPattern"
    value="%d,
    %X{App.Name},
    %X{ARM.Id},
    %X{ARM.GroupName},
    %X{ARM.TranName},
    %X{ARM.TranState},
    %X{ARM.TranId},
    %X{ARM.TranHandle},
    %X{ARM.ParentCorrelator},
    %X{ARM.CurrentCorrelator},
    %X{ARM.TranStatus},
    %X{ARM.TranStart.Time},
    %X{ARM.TranStop.Time},
    %X{ARM.TranBlocked.Time},
    %X{ARM.TranResp.Time}
    "/>
</layout>
</appender>
<appender class="ARMAppender" name="ARM">
    <param name="Agent" value="libarm4"/>

    <param name="Encoding" value="encoding-value"/>

    <param name="GetTimes" value="TRUE | FALSE"/>

    <param name="ManageCorrelators" value="TRUE | FALSE"/>

    <param name="AppName" value="application-name"/>

    <param name="GroupName" value="group-name"/>

    <appender-ref ref="ARM-log-names"/>

</appender>

```

ARMAppender Syntax Description

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

Default: None

Required: Yes. ARM must be the name of the ARMAppender.

Restriction: Only a single instance of an ARMAppender per process.

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

value=" "

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

value="*library-name*"

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

Default: Output is sent to any referenced appenders.

Required: No

name="AppName" value="*application-name*"

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

Default: SAS

Required: No

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

specifies how the log event is written to the ARM log.

Required: No

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

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

specifies the type of character set encoding that is used for strings that are sent to and calls that are received by the ARM 4.0 agent library.

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

Required: No

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

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

Default: None

Required: Yes

name="GetTimes" value="TRUE | FALSE"

enables the ARM appender to compute transaction response time metrics.

TRUE

enables the appender to compute transaction response times.

FALSE

disables the appender to compute transaction response times.

Default: FALSE

Required: No

name="ManageCorrelators" value="TRUE | FALSE"

specifies whether ARMAppender manages transaction correlation.

TRUE

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

FALSE

enables the application to manage transaction correlation.

Default: true

Required: No

name="GroupName" value="*group-name*"

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

Default: current user ID if available, otherwise NULL

Required: No

ARMAppender Example

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

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

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

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

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

  <appender class="FileAppender" name="LOG">
    <param name="File" value="root.log"/>
```

```

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

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

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

</logging:configuration>

```

ARMAppender Usage and Best Practices

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*.

ConsoleAppender

ConsoleAppender Overview

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

ConsoleAppender Syntax

```

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

```

ConsoleAppender Syntax Description

For information about the elements of the appender syntax, see “General Appender Syntax” on page 11.

ConsoleAppender Examples

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>

  <!-- Administration message logger -->
  <logger name="Admin">
    <level value="Error">
  </logger>

  <!-- Application message logger -->
  <logger name="App">
    <level value="Error">
  </logger>

  <!-- Audit message logger -->
  <logger name="Audit">
    <level value="Error">
  </logger>

  <!-- IOM protocol message logger -->
  <logger name="IOM">
    <level value="Error">
  </logger>

  <!-- Root logger -->
  <root>
    <level value="Error"/>
    <appender-ref ref="console"/>
  </root>

</logging:configuration>
```

FileAppender

FileAppender Overview

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

FileAppender Syntax

XML Configuration

```
<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>
```

FileAppender Syntax Description

`class="FileAppender" name="appender-name"`
 specifies the user-assigned name for this instance of FileAppender.

Default: None

Required: Yes

`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

Required: No

Interaction: If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. For details, see the Unique parameter.

`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.

Required: No

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

Required: Yes, if *path-and-filename-pattern* is not specified.

Interaction: *path and filename* overwrites any value that you specify for *path-and-filename-pattern*.

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 *yyyymmdd*, or if you want to include additional information such as the hour.

Main discussion: "Conversion Characters" on page 68

%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.

Main discussion: "Conversion Characters" on page 68

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

Required: Yes, if *path-and-filename* is not specified.

Interaction: *path and filename* is overwritten by any value that you specify for *path-and-filename*.

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.

Required: No

Default: true

name="Locale" value="locale"

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

Required: No

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.

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.

Required: No

Default: None

See: “Logging Thresholds” on page 14

`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`.

Required: No

Default: FALSE

Interaction: 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

Required: No

Main discussion: Chapter 7, “Filters,” on page 77

`name="ConversionPattern" value="conversion-pattern"`
specifies how the log message is written to the log.

Required: No

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

Main discussion: Chapter 6, “Pattern Layout,” on page 65

FileAppender Examples

Example 1: 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>
```

FileAppender Usage and Best Practices

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}**).

FilteringAppender

FilteringAppender Overview

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.

FilteringAppender Syntax

XML Configuration

```
<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>
```

FilteringAppender Syntax Description

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

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

Default: None

Required: Yes

`ref="referenced-appender-name"`

specifies the appender that events are to be passed to.

Required: Yes

`filter-definitions`

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

Default: None

Required: No

Main discussion: Chapter 7, “Filters,” on page 77

`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.

Required: No

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.

Main discussion: Chapter 6, “Pattern Layout,” on page 65

`name="Locale" value="locale"`

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

Required: No

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.

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.

Required: No

Default: TRUE

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.

Required: No

Default: None

See: “Logging Thresholds” on page 14

FilteringAppender 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.#{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>

```

FilteringAppender Usage and Best Practices

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.

IOMServerAppender

IOMServerAppender Overview

IOMServerAppender 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.

IOMServerAppender Syntax

XML Configuration

```
<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>
```

IOMServerAppender 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.

Required: No

Default: 10000

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.

Required: No

Default: None

See: “Logging Thresholds” on page 14

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

specifies how the log message is written to the log.

Required: No

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

Main discussion: Chapter 6, “Pattern Layout,” on page 65

IOMServerAppender 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>
```

IOMServerAppender Usage and Best Practices

IOMServerAppender makes server log messages 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)
```

- On OpenVMS or HP Integrity Servers, define a `sys$scratch` logical name.

△

RollingFileAppender

RollingFileAppender Overview

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. For example, messages can roll over to a new file every hour, every day, when the file grows to a specified size, or when filtering criteria are met.

RollingFileAppender Syntax

XML Configuration

```
<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"/>
</appender>
```



```

<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>

```

RollingFileAppender Syntax Description

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

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

Default: None

Required: Yes

`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

Required: No

Interaction: If both the Unique parameter and the Append parameter are specified, then the Unique parameter takes precedence. For details, see the Unique parameter.

`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.

Required: No

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

Required: No

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.

Required: No

Default: TRUE

name="Locale" value="*locale*"

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

Required: No

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.

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.

Required: No

Default: None

See: "Logging Thresholds" on page 14

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**.

Required: No

Default: FALSE

Interaction:

- 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 description for FixedWindowRollingPolicy.

filter-definitions

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

Default: None

Required: No

Main discussion: Chapter 7, “Filters,” on page 77

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

Required: No

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

Main discussion: Chapter 6, “Pattern Layout,” on page 65

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
- TimeBasedRollingPolicy

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.

Main discussion: “Conversion Characters” on page 68

For 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.

Default: None

Required: No

Interaction: *path and filename* is overwritten by any value that you specify for *path-and-filename*.

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.

Range: 1 - 14

Required: No

Default: 7

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.

Acceptable values: Any integer from 1 to 14

Required: No

Default: 1

Interaction: 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 *yyyymmdd*, or if you want to include additional information such as the hour.

Main discussion: “Conversion Characters” on page 68

%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.

Main discussion: “Conversion Characters” on page 68

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

Required: Yes

Interaction: *path and filename* is overwritten by any value that you specify for *path-and-filename*.

`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

Required: No

triggeringPolicy class="SizeBasedTriggeringPolicy | FilterBasedTriggeringPolicy" specifies the policy that determines when a new log file is created. SAS provides the following instances of triggeringPolicy:

- SizeBasedTriggeringPolicy
- FilterBasedTriggeringPolicy

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.

Main discussion: Chapter 7, “Filters,” on page 77

RollingFileAppender Configuration 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_20080301.log**). When the system date changes, messages roll over to a new log file whose name contains the new date (for example, **MetadataServer_20080302.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_20080301.log**). When a new session begins, messages roll over to a new log file whose name contains the current date (for example, **MetadataServer_20080302.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>
  <layout>
    <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
  </layout>
</appender>
```

```

    </rollingPolicy>
    <triggeringPolicy class="SizeBasedTriggeringPolicy">
        <param name="MaxFileSize" value="100KB"/>
    </triggeringPolicy>
    <layout>
        <param name="ConversionPattern" value="%d %-5p [%t] %u - %m"/>
    </layout>
</appender>

```

RollingFileAppender Usage and Best Practices

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“FileAppender” on page 39.

Note: On OpenVMS systems, uppercase letters are used in the filename for the first log file that is created. When logging rolls over to a new file, the filename is formed, using the case that is specified in the File parameter of the appender configuration. △

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}).

sLogAppender

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

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 will result in unpredictable behavior. △

UNXFacilityAppender

UNXFacilityAppender Overview

UNXFacilityAppender is a logging facility appender that supports event logging on UNIX operating systems. UNXFacilityAppender sends event messages to the **syslogd** logging facility.

UNXFacilityAppender Syntax

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

UNXFacilityAppender 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.

For information about the generic elements of the appender syntax, see “General Appender Syntax” on page 11.

UNXFacilityAppender 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>
```

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 5.1 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.

UNXFacilityAppender Usage and Best Practices

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.

WindowsEventAppender

WindowsEventAppender Overview

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

WindowsEventAppender Syntax

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

WindowsEventAppender 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

For information about the generic elements of the appender syntax, see “General Appender Syntax” on page 11.

WindowsEventAppender 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>

```

Note: The install and configuration process usually create a configuration file automatically. △

WindowsEventAppender Usage

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.

ZOSFacilityAppender

ZOSFacilityAppender Overview

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.

ZOSFacilityAppender Syntax

```

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

```

ZOSFacilityAppender Syntax Description

name="stream" value="value"
 specifies the logstream for ZOSFacilityAppender.

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

For information about the generic elements of the appender syntax, see “General Appender Syntax” on page 11.

ZOSFacilityAppender Examples

Configuring ZOSFacility Appender

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

```

<appender class="ZOSFacilityAppender" name="LOG">
  <param name="stream" value="USERNAME.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>

```

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)

```

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)
/*

```

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'

```

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

```

ZOSFacilityAppender Usage and Best Practices

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.

ZOSWtoAppender

ZOSWtoAppender Overview

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.

ZOSWtoAppender 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>
```

ZOSWtoAppender 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.

For more information about the WTO service macro and the parameters listed here, see IBM's *z/OS V1R9.0 MVS Assembler Services Reference*.

For information about the generic elements of the appender syntax, see “General Appender Syntax” on page 11.

ZOSWtoAppender Example

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

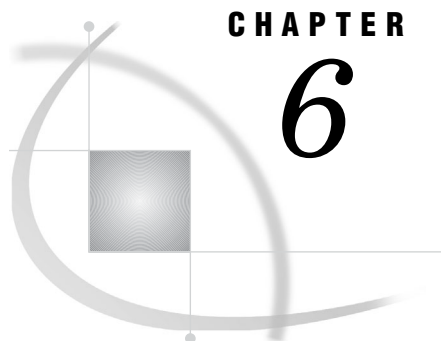
```
<appender name="WTO" class="ZOSWtoAppender">
  <layout>
    <param name=ConversionPattern"
      value="%d %-5p [%t] %c (%F:%L) --- %m"/>
    </layout>

    <param name="routecode" value="11"/>
    <param name="decode" value="7"/>
    <param name="mcsflag" value="HRDCPY"/>
  </appender>

<logger name="ADMIN.OPERATIONS">
  <level value="trace"/>
  <appender-ref ref="WTO"/>
</logger>
```

ZOSWtoAppender Usage and Best Practices

The “routecode”, “decode”, 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.



CHAPTER

6

Pattern Layout

<i>Overview of Pattern Layouts</i>	65
<i>Syntax for a Pattern Layout</i>	65
<i>Pattern Layout Syntax Description</i>	66
<i>Conversion Characters</i>	68
<i>Format Modifiers</i>	73
<i>Examples</i>	73
<i>Examples of Format Modifiers</i>	73
<i>Example of a Pattern Layout</i>	74
<i>Example of a Formatted Log Event</i>	75

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 ARMApender Pattern Layouts 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. △

Syntax for a Pattern Layout

XML Configuration

```

<appender class="AppenderName" name="log-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>

```

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. △

Pattern Layout Syntax Description

`class= "AppenderName" name="log-name"`

specifies the name of the log for the specified appender that the formatted log events are directed to.

See Also: For the appender class names, see Chapter 5, “Appender Reference,” on page 33.

`name="Header" value="header-text"`

specifies the header text that the appender uses when it starts a new log.

`name="HeaderPattern" value="conversion-pattern"`

specifies the pattern layout that is used to identify, order, and format information in a header for a log event. 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*. 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] system-conversion-character-1 [... [literal-text]
%[format-modifier-n] system-conversion-character-n] />

```

Valid in: XML configuration files for any appender class.

Restriction: The header pattern is limited to the S conversion character in a logging configuration file.

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 conversion character.

Example: `<param name="HeaderPattern" value="%S{os_name} %S{jobid} %S{host_name} %S{user_name}"/>`

See: S Conversion Character on page 70
 “Format Modifiers” on page 73

`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: `"%d %-5p [%t] %X{Client.ID}:%u - %m"`

Restriction: Conversion specifiers are case sensitive.

Do not use these problematic characters, known as *variants*, in pattern layouts in EBCDIC encoding environments:

`! # $ % & \ [] ^ { } | ~ \n`

Note: `\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.

Example: `<param name="ConversionPattern" value="%d; %-5p; %t; %c; (%F:%L); %m"/>`

See: “Conversion Characters” on page 68
 “Format Modifiers” on page 73

See Also: For more information about variant characters, see the *SAS National Language Support (NLS): Reference Guide*.

`name="Footer" value="footer-text"`
 specifies the footer text that the appender uses when it starts a new log.

`name="FooterPattern" value="conversion-pattern"`
 specifies the pattern layout that is used to identify, order, and format information in a footer for a log event. 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*. 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] system-conversion-character-1 [... [literal-text]
%[format-modifier-n] system-conversion-character-n] />
```

Valid in: XML configuration files for any appender class.

Restriction: The footer pattern is limited to the S conversion character in a logging configuration file.

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 conversion character.

Example: `<param name="FooterPattern" value="%S{host_name}"/>`

See: S Conversion Character on page 70
“Format Modifiers” on page 73

`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 ">,"

"'" is converted to "","

"'" is converted to "',"

"&" is converted to "&,"

FALSE

specifies that no character conversion to entity representations is performed.

Conversion Characters

A conversion character defines a data item to include in the message. Some conversion characters also contain optional specifiers. Here are the supported conversion characters:

c [*{precision-specifier}*]

reports the name of the logger that generates the log event.

Alias: logger

Default: Complete logger name; for example,
"Logging.Appender.IOMCallContext".

Requirement: If the precision specifier is used, it must be surrounded with a pair of braces.

The precision specifier is represented as a decimal constant.

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 [*{date conversion specifier}*]

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 surrounded with a pair of braces.

Interaction: Here are the supported *date conversion specifiers*:

ABSOLUTE

specifies the time in this format: *HH:mm:ss,SSS*. An example is 15:49:37,459

DATE

specifies the date and time in this format: *dd MMM yyyy HH:mm:ss,SSS*. An example is 06 Nov 2008 15:49:37,459

ISO8601

specifies the date and time in this format: *yyyy-MM-dd HH:mm:ss,SSS*. An example is 1999-11-27 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: AM/PM marker (string)	FF: Day of week in month (numeric)
dd: Day in month (numeric)	GG: Era designator (string "AD")
hh: Hour in AM/PM (numeric 1-12)	HH: Hour in day (numeric 0-23)
mm: Minute in hour (numeric)	KK: Hour in AM/PM (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)	

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.

Examples: d{ABSOLUTE}
d{EEEE MMMM yyyy HH:mm:ss,SSS 'Ship date'}

F

reports the name of the file in the application that generated the log event.

Alias: file

L

reports the line number in the application that generated the log event.

Alias: line

m [{*prefix-identifier*}]

writes the messages that are associated with the log event. 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.

Alias: message

Default: None

Valid values: Here are the supported *prefix-identifiers*:

HYPHEN

inserts a hyphen (-) before each message.

PLUS

inserts a plus sign (+) before each message.

Requirement: If the prefix identifier is used, it must be surrounded with a pair of braces.

n [{*newline-prefix-identifier*}]

enables you to supply discretionary newline characters among the data items that compose the log event.

Default: None

Valid values: Here are the supported *prefix-identifiers*:

HYPHEN

inserts a hyphen (-) before each message.

PLUS

inserts a plus sign (+) before each message.

Requirement: If the prefix identifier is used, it must be surrounded with a pair of braces.

p

reports the level of the log event.

Alias: level

Valid values: Here are the supported levels:

TRACE

DEBUG

INFO

WARN

ERROR

FATAL

r

reports the number of milliseconds that elapsed between the start of the application and the creation of the log event.

Alias: relative

S {*key*}

delivers various system information to the log event.

Alias: systemInfo

Requirement: The S conversion character must be followed by the specified value, which is also referred to as a *key*, and must be surrounded with a pair of braces.

Valid values: Here are the supported *keys* and values:

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. Δ

For more information, see the u conversion character on page 72.

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: %S{os_family}

sn

reports the sequence number of the log event.

Alias: sequenceNumber

t

reports the identifier of the thread that generated the log event.

Alias: thread

u

reports the client identity that is associated with the current thread or task. 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.

Alias: username

See Also: S {user_name} on page 71

x

reports the NDC (nested diagnostic context) that is associated with the thread that generated the log event.

Alias: ndc

X {key}

reports the MDC (mapped diagnostic context) that is associated with the thread that generated the log event. 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.

Alias: properties

Requirement: The key must be surrounded with a pair of braces. An example is %X{clientNumber}, where clientNumber is the key.

%%

enables you to specify a literal percent sign symbol in a text string of a conversion pattern. 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: <param name="ConversionPattern" value="%d;text%%text;%m"/>

Here is sample output:

2008-06-25-10:24:22,234; text;text;Numeric maximum was larger than 8,
am setting to 8.

Format Modifiers

A *format modifier* is an optional set of characters that controls the field width, padding, and justification of the specified data item in log output. Here are the supported format modifiers:

- (hyphen)

specifies left-justification of the data item that is defined by the conversion character.

Example: %-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

specified as 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)

Example: %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.

Tip: The value is never truncated. To specify a maximum width, use the *maximum-field-width-modifier*.

maximum-field-width-modifier

specified as 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.

Example: %.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".

Restriction: The behaviors of the *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

Examples of Format Modifiers

Here are examples of format modifiers that are used with the c conversion character. The c conversion character reports the name of the logger.

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.

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 (%F:%L) %m"/>
</layout>
```

Here is an explanation:

Pattern Layout	Explanation	Example
%d	reports the date of the log event and formats the date using the default format, ISO8601.	2008-06-25 10:24:22,234; <i>Note:</i> The semicolon (;) is literal text. △
%-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 ; <i>Note:</i> The semicolon (;) is literal text. △
%t	reports the identifier of the thread that generated the log event.	3; <i>Note:</i> The semicolon (;) is literal text. △

Pattern Layout	Explanation	Example
<code>%c{2}</code>	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; <i>Note:</i> The semicolon (;) is literal text. △
<code>(%F:%L)</code>	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); <i>Note:</i> The parentheses ()), colon (:), and semicolon (;) are literal text. △
<code>%m</code>	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.

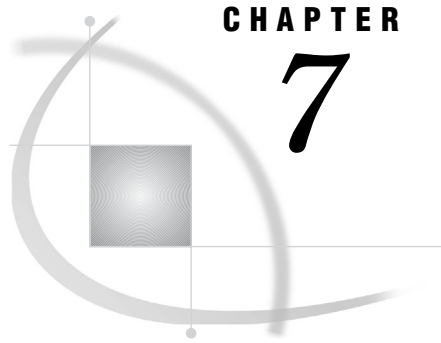
Example of a Formatted Log Event

Here is an example of a formatted log event that is delivered to the appropriate output device:

Date	Level	Thread ID	Logger	Filename/Line Number	Message
20008-06-25 10:24:22,234;	WARN;	3;	Appender. IOMCallContext;	(ynl4.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; (ynl4.sas.c:149);
      Numeric maximum was larger than 8, am setting to 8.
```

CHAPTER

7

Filters

<i>Overview of Filters</i>	77
<i>Syntax for Filters</i>	79
<i>AndFilter</i>	79
<i>AndFilter Overview</i>	79
<i>AndFilter Syntax</i>	79
<i>AndFilter Syntax Description</i>	80
<i>AndFilter Examples</i>	81
<i>DenyAllFilter</i>	81
<i>DenyAllFilter Overview</i>	81
<i>DenyAllFilter Syntax</i>	81
<i>DenyAllFilter Syntax Description</i>	81
<i>LevelMatchFilter</i>	81
<i>LevelMatchFilter Overview</i>	81
<i>LevelMatchFilter Syntax</i>	82
<i>LevelMatchFilter Syntax Description</i>	82
<i>LevelMatchFilter Example</i>	82
<i>LevelRangeFilter</i>	82
<i>LevelRangeFilter Overview</i>	82
<i>LevelRangeFilter Syntax</i>	83
<i>LevelRangeFilter Syntax Description</i>	83
<i>LevelRangeFilter Example</i>	83
<i>StringMatchFilter</i>	83
<i>When to Use the StringMatchFilter</i>	84
<i>StringMatchFilter Syntax</i>	84
<i>StringMatchFilter Syntax Description</i>	84
<i>StringMatchFilter Example</i>	84
<i>Filter Examples</i>	84
<i>Example 1: Filter for a Specific User's Error Messages</i>	84
<i>Example 2: Filter for a Specific Date</i>	85

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
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 that 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 `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:

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="StringToMatch" value="Perf.ARM"/>
  <param="AcceptOnMatch" value="true"/>
</filter>
<filter class="DenyAllFilter">
</filter>
```

Note: Filter definitions are not available in the logging facility language elements for SAS programs. △

Syntax for Filters

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>
```

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. △

AndFilter

AndFilter Overview

You use AndFilter when you want to log messages that meet multiple criteria. 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.

AndFilter 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>

```

AndFilter 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	DEBUG TRACE INFO WARN ERROR FATAL
	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.

AndFilter Examples

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>
```

DenyAllFilter

DenyAllFilter Overview

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.

DenyAllFilter Syntax

```
<filter class="DenyAllFilter">
```

DenyAllFilter Syntax Description

class="DenyAllFilter"

specifies to deny all messages that do not meet the filter chain criteria.

LevelMatchFilter

LevelMatchFilter Overview

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.

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.

LevelMatchFilter Syntax

```
<filter class="LevelMatchFilter">
  <param name="LevelToMatch" value="DEBUG | TRACE | INFO | WARN |
    ERROR | FATAL"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

LevelMatchFilter 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: Chapter 6, “Pattern Layout,” on page 65

LevelMatchFilter 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

LevelRangeFilter Overview

Use LevelRangeFilter when you want to filter log event messages whose message threshold falls within a range of message thresholds. 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 TRUE, messages that have the thresholds TRACE and FATAL are filtered.

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.

LevelRangeFilter Syntax

```
<filter class="LevelRangeFilter">
  <param name="LevelMax" value="threshold"/>
  <param name="LevelMin" value="threshold"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

LevelRangeFilter 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.

LevelRangeFilter 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>
```

StringMatchFilter

When to Use the StringMatchFilter

Use StringMatchFilter when you want to filter messages based on a string in the log event message.

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.

StringMatchFilter Syntax

```
<filter class="StringMatchFilter">
  <param name="StringToMatch" value="character-string"/>
  <param name="AcceptOnMatch" value="TRUE | FALSE"/>
</filter>
```

StringMatchFilter 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.

StringMatchFilter 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="false"/>
</filter>
```

Filter Examples

Example 1: Filter for a Specific User's Error Messages

In this example, the filtering policy 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="AndFilter">
      <param="AcceptOnMatch" value="true"/>
      <filter class="LevelMatchFilter">
        <param="LevelToMatch" value="error"/>
        <param="AcceptOnMatch" value="true"/>
      </filter>
      <filter class="StringMatchFilter">
        <param="StringToMatch" value="sasuser1"/>
        <param="AcceptOnMatch" value="true"/>
      </filter>
    </filter>
    <filter class="DenyAllFilter">
    </filter>
  </appender>
  <root>
    <level value="trace"/>
    <appender-ref ref="eventLog"/>
  </root>
</logging:configuration>

```

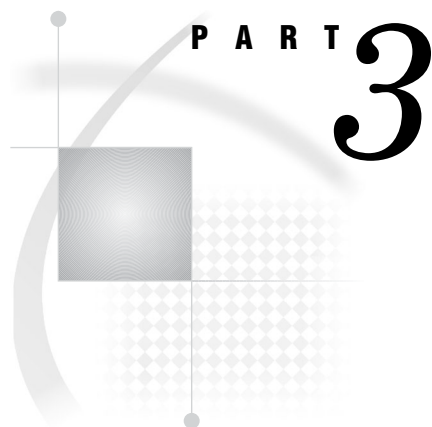
Example 2: Filter for a Specific Date

The following filtering policy denies log events that were sent on 2008-09-22:

```

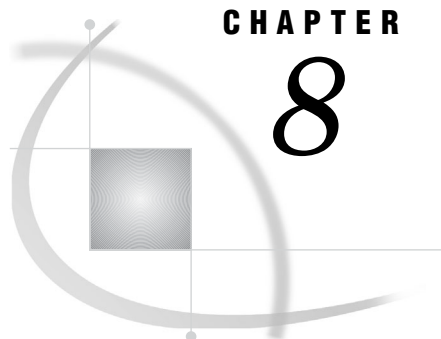
<filter class="StringMatchFilter">
  <param="StringToMatch" value="2008-09-22"/>
  <param="AcceptOnMatch" value="false"/>
</filter>
<filter class="DenyAllFilter">
</filter>

```

The Logging Facility for SAS Programs

<i>Chapter 8</i>	The SAS Logging Facility in the SAS Language	<i>89</i>
<i>Chapter 9</i>	Autocall Macro Reference	<i>97</i>
<i>Chapter 10</i>	Function Reference	<i>111</i>
<i>Chapter 11</i>	Logger and Appender Object Language Reference	<i>121</i>
<i>Appendix 1</i>	Recommended Reading	<i>137</i>



CHAPTER

8

The SAS Logging Facility in the SAS Language

<i>Overview of the SAS Logging Facility in the SAS Language</i>	89
<i>Initializing the SAS Logging Facility for SAS Programs</i>	90
<i>Which Language Elements Need Initializing?</i>	90
<i>Initializing the Logging Facility Autocall Macros</i>	90
<i>The LOGCONFIGLOC= System Option</i>	90
<i>Creating and Using Appenders in a SAS Program</i>	90
<i>Creating Appenders</i>	90
<i>Associating Appenders with Loggers</i>	91
<i>Creating Loggers in a SAS Program</i>	91
<i>Using SAS Language Elements to Create Loggers</i>	91
<i>Updating Logger Attributes</i>	92
<i>Message Categories in the SAS Language</i>	92
<i>Creating Log Events in a SAS Program</i>	94
<i>Example of Creating Logger and Appender Categories</i>	94

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 and DATA step objects for DATA step programming. The elements are also autocall macros 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 8 and “LOGCONFIGLOC= System Option” on page 20.

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 20 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 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=myfiles");
    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 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 99
- “LOG4SAS_APPENDER Function” on page 112
- “DECLARE Statement, Appender Object” on page 126

Associating Appendors 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. 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 101
- “LOG4SAS_LOGGER Function” on page 113
- “DECLARE Statement, Logger Object” on page 128

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=(functionAppender) 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 94
- “%LOG4SAS_LOGGER Autocall Macro” on page 101
- “LOG4SAS_LOGGER Function” on page 113
- “DECLARE Statement, Logger Object” on page 128

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.

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 94.

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, 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 14
- “%LOG4SAS_TRACE Autocall Macro” on page 102
- “%LOG4SAS_DEBUG Autocall Macro” on page 103
- “%LOG4SAS_INFO Autocall Macro” on page 104
- “%LOG4SAS_WARN Autocall Macro” on page 105
- “%LOG4SAS_ERROR Autocall Macro” on page 106
- “%LOG4SAS_FATAL Autocall Macro” on page 107
- “LOG4SAS_LOGEVENT Function” on page 115
- “TRACE Method” on page 134
- “DEBUG Method” on page 125
- “INFO Method” on page 132
- “WARN Method” on page 135
- “ERROR Method” on page 130
- “FATAL Method” on page 131

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;

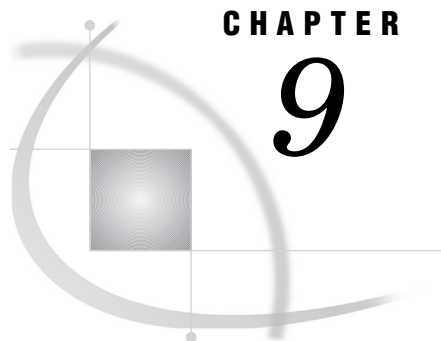
/* Create the context for logging regression messages. */
/* Regression log events of level info or higher are written * /
/* to the destination specified by the appender defined as myPgmRegression. */

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;

/* 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;

```

CHAPTER

9

Autocall Macro Reference

<i>Using Autocall Macros to Log Messages</i>	97
<i>Dictionary</i>	98
%LOG4SAS Autocall Macro	98
%LOG4SAS_APPENDER Autocall Macro	99
%LOG4SAS_LOGGER Autocall Macro	101
%LOG4SAS_TRACE Autocall Macro	102
%LOG4SAS_DEBUG Autocall Macro	103
%LOG4SAS_INFO Autocall Macro	104
%LOG4SAS_WARN Autocall Macro	105
%LOG4SAS_ERROR Autocall Macro	106
%LOG4SAS_FATAL Autocall Macro	107
<i>Example of Using Autocall Macros to Log Messages</i>	108

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

initializes the autocall macro logging environment.

%LOG4SAS_APPENDER

defines an appender, which names the destination of the log message, a message layout, and a message threshold.

%LOG4SAS_LOGGER

defines a logger, which defines a message category for log messages and the appenders that are associated with the message category.

%LOG4SAS_DEBUG

is the log event that you use to write debug messages.

%LOG4SAS_TRACE

is the log event that you use to write trace messages.

%LOG4SAS_WARN

is the log event that you use to write warning messages.

%LOG4SAS_INFO

is the log event that you use to write informational messages.

`%LOG4SAS_ERROR`

is the log event that you use to write error messages.

`%LOG4SAS_FATAL`

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 “Selected Autocall Macros Provided with SAS Software” and “MAUTOSOURCE System Option” in *SAS Macro Language: Reference*.

Dictionary

%LOG4SAS Autocall Macro

Initializes the logging environment to use autocall macros.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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

Autocall macros

“`%LOG4SAS_APPENDER` Autocall Macro” on page 99

“`%LOG4SAS_LOGGER` Autocall Macro” on page 101

“`%LOG4SAS_DEBUG` Autocall Macro” on page 103

- “%LOG4SAS_TRACE Autocall Macro” on page 102
- “%LOG4SAS_WARN Autocall Macro” on page 105
- “%LOG4SAS_INFO Autocall Macro” on page 104
- “%LOG4SAS_ERROR Autocall Macro” on page 106
- “%LOG4SAS_FATAL Autocall Macro” on page 107
- “Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_APPENDER Autocall Macro

Defines an appender.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: The %LOG4SAS autocall macro must be invoked before this macro is invoked to initialize the logging facility autocall macros.

Requirement: 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. △

Requirement: Appender classes are case sensitive. In this instance, the classname must be **FileRefAppender**.

Requirement: 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 6, “Pattern Layout,” on page 65.

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 14.

Examples

The following appender definition names the appender **debugMyProgram**, names a log message destination using the fileref debugOutput, and specifies a pattern that reports the filename and the line number in 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

Autocall macros

“%LOG4SAS Autocall Macro” on page 98

“%LOG4SAS_LOGGER Autocall Macro” on page 101

“%LOG4SAS_DEBUG Autocall Macro” on page 103

“%LOG4SAS_TRACE Autocall Macro” on page 102

“%LOG4SAS_WARN Autocall Macro” on page 105

“%LOG4SAS_INFO Autocall Macro” on page 104

“%LOG4SAS_ERROR Autocall Macro” on page 106

“%LOG4SAS_FATAL Autocall Macro” on page 107

“Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_LOGGER Autocall Macro

Defines a logger.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

APPENDER-REF=(*appender-list*)

specifies one or more appender names to which log events are passed.

Requirement: The appender names must already exist. Appender names are created by the %LOG4SAS_APPENDER autocall macro.

Requirement: 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 applied by the logger. The following are the valid level values, from lowest to highest: TRACE, DEBUG, INFO, WARN, ERROR, FATAL.

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 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, assume you have a logger, `CONSOLE`. The output of a log event for the logger `CONSOLE` goes to all of the appenders in `CONSOLE` and its ancestors. However, if an ancestor of logger `CONSOLE`, say `FILE`, has the additivity argument set to `FALSE`, then `CONSOLE`’s output is directed to all of the appenders in `CONSOLE` and its ancestors up to and including `FILE`, but not to the appenders in any of the ancestors of `FILE`.

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 14.

See Also

Autocall macros

- “%LOG4SAS Autocall Macro” on page 98
- “%LOG4SAS_APPENDER Autocall Macro” on page 99
- “%LOG4SAS_DEBUG Autocall Macro” on page 103
- “%LOG4SAS_TRACE Autocall Macro” on page 102
- “%LOG4SAS_WARN Autocall Macro” on page 105
- “%LOG4SAS_INFO Autocall Macro” on page 104
- “%LOG4SAS_ERROR Autocall Macro” on page 106
- “%LOG4SAS_FATAL Autocall Macro” on page 107
- “Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_TRACE Autocall Macro

Logs a TRACE message if the specified logger accepts TRACE messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

“%LOG4SAS Autocall Macro” on page 98

“%LOG4SAS_APPENDER Autocall Macro” on page 99

“%LOG4SAS_LOGGER Autocall Macro” on page 101

“%LOG4SAS_DEBUG Autocall Macro” on page 103

“%LOG4SAS_WARN Autocall Macro” on page 105

“%LOG4SAS_INFO Autocall Macro” on page 104

“%LOG4SAS_ERROR Autocall Macro” on page 106

“%LOG4SAS_FATAL Autocall Macro” on page 107

“Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_DEBUG Autocall Macro

Logs a DEBUG message if the specified logger accepts DEBUG messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

“%LOG4SAS Autocall Macro” on page 98

“%LOG4SAS_APPENDER Autocall Macro” on page 99

“%LOG4SAS_LOGGER Autocall Macro” on page 101

“%LOG4SAS_TRACE Autocall Macro” on page 102

“%LOG4SAS_WARN Autocall Macro” on page 105

“%LOG4SAS_INFO Autocall Macro” on page 104

“%LOG4SAS_ERROR Autocall Macro” on page 106

“%LOG4SAS_FATAL Autocall Macro” on page 107

“Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_INFO Autocall Macro

Logs an INFO message if the specified logger accepts INFO messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

- “%LOG4SAS Autocall Macro” on page 98
- “%LOG4SAS_APPENDER Autocall Macro” on page 99
- “%LOG4SAS_LOGGER Autocall Macro” on page 101
- “%LOG4SAS_DEBUG Autocall Macro” on page 103
- “%LOG4SAS_TRACE Autocall Macro” on page 102
- “%LOG4SAS_WARN Autocall Macro” on page 105
- “%LOG4SAS_ERROR Autocall Macro” on page 106
- “%LOG4SAS_FATAL Autocall Macro” on page 107
- “Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_WARN Autocall Macro

Logs a WARN message if the specified logger accepts WARN messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

“%LOG4SAS Autocall Macro” on page 98

“%LOG4SAS_APPENDER Autocall Macro” on page 99

“%LOG4SAS_LOGGER Autocall Macro” on page 101

“%LOG4SAS_DEBUG Autocall Macro” on page 103

“%LOG4SAS_WARN Autocall Macro” on page 105

“%LOG4SAS_INFO Autocall Macro” on page 104

“%LOG4SAS_ERROR Autocall Macro” on page 106

“%LOG4SAS_FATAL Autocall Macro” on page 107

“Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_ERROR Autocall Macro

Logs an ERROR message if the specified logger accepts ERROR messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

- “%LOG4SAS Autocall Macro” on page 98
- “%LOG4SAS_APPENDER Autocall Macro” on page 99
- “%LOG4SAS_LOGGER Autocall Macro” on page 101
- “%LOG4SAS_DEBUG Autocall Macro” on page 103
- “%LOG4SAS_TRACE Autocall Macro” on page 102
- “%LOG4SAS_WARN Autocall Macro” on page 105
- “%LOG4SAS_INFO Autocall Macro” on page 104
- “%LOG4SAS_FATAL Autocall Macro” on page 107
- “Example of Using Autocall Macros to Log Messages” on page 108

%LOG4SAS_FATAL Autocall Macro

Logs a FATAL message if the specified logger accepts FATAL messages.

Category: Logging

Requirement: The MAUTOSOURCE system option must be set.

Requirement: 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.

See Also

Autocall macros

- “%LOG4SAS Autocall Macro” on page 98
- “%LOG4SAS_APPENDER Autocall Macro” on page 99
- “%LOG4SAS_LOGGER Autocall Macro” on page 101
- “%LOG4SAS_DEBUG Autocall Macro” on page 103
- “%LOG4SAS_TRACE Autocall Macro” on page 102

“%LOG4SAS_WARN Autocall Macro” on page 105

“%LOG4SAS_INFO Autocall Macro” on page 104

“%LOG4SAS_ERROR Autocall Macro” on page 106

“Example of Using Autocall Macros to Log Messages” on page 108

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 9.1 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:

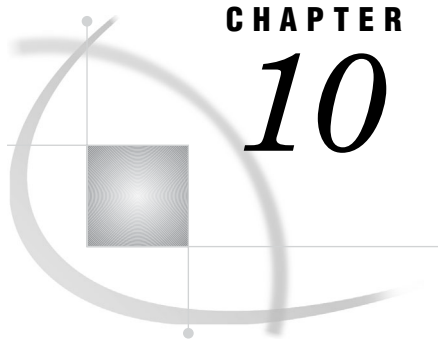
```

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

```



CHAPTER

10

Function Reference

<i>Using the Logging Facility Functions in the DATA Step</i>	111
<i>Dictionary</i>	112
<i>LOG4SAS_APPENDER Function</i>	112
<i>LOG4SAS_LOGGER Function</i>	113
<i>LOG4SAS_LOGEVENT Function</i>	115
<i>Logging Example Using Functions</i>	117

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

creates an appender. FileRefAppender is the only type of appender that can be created by using the SAS language.

LOG4SAS_LOGGER

logs a message by using a specific logger.

LOG4SAS_LOGEVENT

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=myfiles");
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;
    end;

```

Dictionary

LOG4SAS_APPENDER Function

Creates a fileref appender that can be referenced by a logger.

Category: Logging

Featured in: “Logging Example Using Functions” on page 117

Syntax

LOG4SAS_APPENDER(*"name"*, "FileRefAppender", *'options'*)

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.

Required: Yes

PATTERN=*"pattern"*

specifies one or more message layout patterns that are used to format the log message.

See: Chapter 6, “Pattern Layout,” on page 65

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 to 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 pattern contains just the log message.

For more information, see Chapter 6, “Pattern Layout,” on page 65.

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 14.

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 113

“LOG4SAS_LOGEVENT Function” on page 115

LOG4SAS_LOGGER Function

Creates a logger.

Category: Logging

Featured in: “Logging Example Using Functions” on page 117

Syntax

```
LOG4SAS_LOGGER("name", <"options">)
```

Arguments

"name"

specifies a name for the logger.

Requirements: The name must be enclosed in double quotation marks.

Tip: Requests to create a logger are ignored if they use the name of an existing 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 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

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 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.

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 8.

Appender Reference and Additivity The appenders that are in *appender_name_list* must be defined by using the LOG4SAS_APPENDER function 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*. If ADDITIVITY=FALSE, the log event is processed only by the logger.

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 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 14.

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;
```

See Also

Functions:

“LOG4SAS_APPENDER Function” on page 112

“LOG4SAS_LOGEVENT Function” on page 115

LOG4SAS_LOGEVENT Function

Logs a message by using a specific logger.

Category: Logging

Featured in: “Logging Example Using Functions” on page 117

Syntax

Log4SAS_logevent(*name*, *level*, *message*)

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 might allow the application to continue running.
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.

Interaction: The only variables that the message can resolve are macro variables. DATA step variables do not resolve in the message.

Requirement: The message must be enclosed in quotation marks.

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 8.

See Also

Functions:

“LOG4SAS_APPENDER Function” on page 112

“LOG4SAS_LOGGER Function” on page 113

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 mmddyy10.;
  format dob tod mmddyy10.;

  /* 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=myfiles");
    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;

```

```

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 MYFILES fileref contains the following logging facility messages:

```

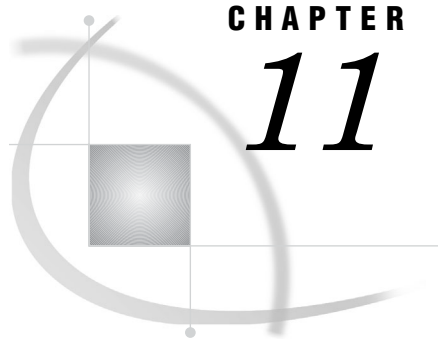
Obtained today's date.
Determined the number of business days.
Found date differences.
Found date differences.
Made adjustments in days.
Found date differences.
Made adjustments in months.
Found date differences.
Made adjustments in days.
Found date differences.
Made adjustments in days.
Found date differences.
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 10.1 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



CHAPTER

11

Logger and Appender Object Language Reference

<i>The Logger and Appender Component Object Interface</i>	121
<i>Dot Notation and DATA Step Component Objects</i>	122
<i>Definition</i>	122
<i>Syntax</i>	122
<i>Dictionary</i>	123
<i>ADDITIVITY Attribute</i>	123
<i>APPENDERREF Attribute</i>	124
<i>DEBUG Method</i>	125
<i>DECLARE Statement, Appender Object</i>	126
<i>DECLARE Statement, Logger Object</i>	128
<i>ERROR Method</i>	130
<i>FATAL Method</i>	131
<i>INFO Method</i>	132
<i>LEVEL Attribute</i>	133
<i>TRACE Method</i>	134
<i>WARN Method</i>	135

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. For more information about SAS logging, see “The SAS Log” in *SAS Language Reference: Concepts* and Chapter 1, “The SAS Logging Facility,” on page 3.

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 the method is success or failure. A return code of zero indicates success; a non-zero 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";

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, assume that you have a logger, CONSOLE. The output of a log event of logger CONSOLE goes to all of the appenders that are in CONSOLE and its ancestors. However, if an ancestor of logger CONSOLE, say FILE, has the additivity flag set to FALSE, then CONSOLE's output is directed to all of the appenders that are in CONSOLE and its ancestors up to and including FILE, but not to the appenders that are in any of the ancestors of FILE.

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 128. △

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

Statement:

“DECLARE Statement, Logger Object” on page 128

APPENDERREF Attribute

Specifies the name of the appender to which log events are passed.

Applies to: logger object

Syntax

object.APPENDERREF = "appender-name";

Arguments

object

specifies the name of the logger object.

appendername

specifies the name of the appender to which the log events for the specified logger are passed.

Interaction: 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.

Interaction: The appender name must already exist. Appender names are created by using the DECLARE statement. For more information, see “DECLARE Statement, Appender Object” on page 126.

Details

You can specify more than one appender for each logger.

Example

The logger object in the following example references the myappd appender.

```
data _null_;
  filename myfref "my.log";
  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

Attribute:

“ADDITIVITY Attribute” on page 123

Statement:

“DECLARE Statement, Appender Object” on page 126

DEBUG Method

Logs a DEBUG message if the specified logger accepts DEBUG messages.

Applies to: logger object

Syntax

object.**DEBUG** ("message");

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 14.

Examples

The following example creates a debugging message for the logger.

```

data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
  end;
  logobj.debug("Test debug message");
run;

```

See Also

Method:

“ERROR Method” on page 130

“FATAL Method” on page 131

“INFO Method” on page 132

“TRACE Method” on page 134

“WARN Method” on page 135

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 ("name", "FileRefAppender", "FILEREf =fileref"
  <, PATTERN: "pattern"> <, THRESHOLD: "threshold">);

```

Arguments

name

specifies the name of the appender object.

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.

Note: This is the only supported appender class. More appender classes might be supported in the future. △

Requirement: Appender class names are case sensitive. In this instance, the name must be “FileRefAppender”.

Requirement: FileRefAppender must be enclosed in double quotation marks.

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 log event produces an empty string.

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 124. 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 contains the log message only.
For more information, see “Overview of Pattern Layouts” on page 65.

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 14.

Examples

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

Attribute:

“APPENDERREF Attribute” on page 124

Statement:

“DECLARE Statement, Logger Object” on page 128

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 ("name" <, ADDITIVITY: TRUE | FALSE><, LEVEL: "level">
  <,, APPENDERREF: "appender-name"<..., APPENDERREF: "appender-name">>);
```

Arguments

name

specifies the name of the logger object.

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.

Tip: You can also specify this optional argument by using the “ADDITIVITY Attribute” on page 123 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” on page 126.

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.

Tip: You can specify more than one appender for each logger.

Tip: 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 124.

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.

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 133 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 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, assume that you have a logger, CONSOLE. The output of a log event of logger CONSOLE goes to all of the appenders that are in CONSOLE and its ancestors. However, if an ancestor of logger CONSOLE, say FILE, has the additivity flag set to FALSE, then CONSOLE’s output is directed to all of the appenders that are in CONSOLE and its ancestors up to and including FILE, but not the appenders in any of the ancestors of FILE.

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 14.

Examples

The following example creates a logger object, mylog.

```

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

Attribute:

“ADDITIVITY Attribute” on page 123

“APPENDERREF Attribute” on page 124

“LEVEL Attribute” on page 133

Statement:

“DECLARE Statement, Appender Object” on page 126

ERROR Method

Logs an ERROR message if the specified logger accepts ERROR messages.

Applies to: logger object

Syntax

object.**ERROR** ("message");

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 that might allow the application to continue running. For more information about the logging levels, see “Logging Thresholds” on page 14.

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

Method:

“DEBUG Method” on page 125

“FATAL Method” on page 131

“INFO Method” on page 132

“TRACE Method” on page 134

“WARN Method” on page 135

FATAL Method

Logs a FATAL message if the specified logger accepts FATAL messages.

Applies to: logger object

Syntax

object.**FATAL** ("message");

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 will probably cause the application to end abruptly. For more information about logging levels, see “Logging Thresholds” on page 14.

Example

The following example creates an error message for the logger.

```
data _null_;
  if _n_ = 1 then do;
    declare logger logobj("testlog");
  end;
  logobj.fatal("Test fatal message");
run;
```

See Also

Method:

- “DEBUG Method” on page 125
- “ERROR Method” on page 130
- “INFO Method” on page 132
- “TRACE Method” on page 134
- “WARN Method” on page 135

INFO Method

Logs an INFO message if the specified logger accepts INFO messages.

Applies to: logger object

Syntax

object.**INFO** ("message");

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 14.

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

Method:

“DEBUG Method” on page 125

“ERROR Method” on page 130

“FATAL Method” on page 131

“TRACE Method” on page 134

“WARN Method” on page 135

LEVEL Attribute

Defines the level of message that is accepted by the specified logger.

Applies to: logger object

Syntax

object.**LEVEL**= "level";

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 14.

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 128. △

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

Statement:

“DECLARE Statement, Logger Object” on page 128

TRACE Method

Logs a TRACE message if the specified logger accepts TRACE messages.

Applies to: logger object

Syntax

object.TRACE ("message");

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 14.

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

Method:

“DEBUG Method” on page 125

“ERROR Method” on page 130

“FATAL Method” on page 131

“INFO Method” on page 132

“WARN Method” on page 135

WARN Method

Logs a WARN message if the specified logger accepts WARN messages.

Applies to: logger object

Syntax

object.**WARN** ("message");

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 14.

Example

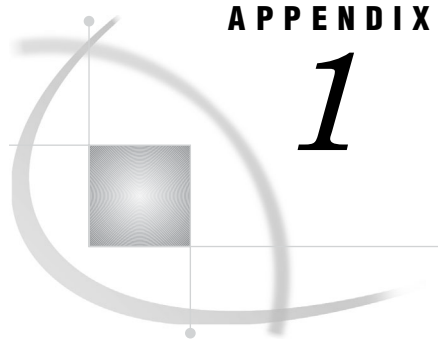
The following example creates a warn message for a logger.

```
data _null_;  
  if _n_ = 1 then do;  
    declare logger logobj("testlog");  
  end;  
  logobj.warn("Test warn message");  
run;
```

See Also

Method:

- “DEBUG Method” on page 125
- “ERROR Method” on page 130
- “FATAL Method” on page 131
- “INFO Method” on page 132
- “TRACE Method” on page 134



APPENDIX

1

Recommended Reading

Recommended Reading 137

Recommended Reading

Here is the recommended reading list for this title:

- *SAS Intelligence Platform: System Administration Guide*
- *SAS Interface to Application Response Measurement (ARM): Reference*
- *SAS Language Reference: Dictionary*
- *SAS Macro Language: Reference*
- *SAS OLAP Server: User's Guide*

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

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: 1-800-727-3228
Fax: 1-919-531-9439
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/bookstore**

Customers outside the United States and Canada, please contact your local SAS office for assistance.

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. See also appender additivity.

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. See also additivity flag.

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. See also conversion specifier, format modifier, and pattern layout.

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. See also conversion pattern.

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.

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

in the logging facility, a set of character strings, 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.

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

a SAS object server that is launched in order to fulfill client requests for IOM services. Short form: IOM server.

IOM

See Integrated Object Model.

IOM server

See Integrated Object Model server.

level

the diagnostic level that is associated with a log event. Examples of levels 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. 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.

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 an installation and configuration tool called 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. See also SAS log.

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 is designed to prompt the customer for all the remaining input at the start of the session so that the customer does not have to monitor an 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. In some cases, the SAS log can also contain output from the DATA step and from certain procedures. See also SAS console log.

threshold

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

Index

A

accessibility 3
 Action Response Measurement 34
 additivity
 See appender additivity
 ADDITIVITY attribute 123
 AndFilter 79
 appender additivity 102, 115
 logger objects and 129
 passing log events and 123
 appender categories
 creating 94
 appender classes
 formatting log messages for 65
 appender objects
 creating an instance of 126
 declaring 126
 FileRefAppender and 127
 initializing data for 126
 patterns and 127
 thresholds and 127
 APPENDERREF attribute 124
 appenders 10
 ARMAppender 34
 associating with loggers 91
 ConsoleAppender 38
 creating and using in SAS programs 90
 creating fileref appenders 112
 defining 99
 definition 5
 FileAppender 40
 FileRefAppender 97, 113, 127
 FilteringAppender 43
 in SAS language 13
 IOMServerAppender 46
 names of 100, 112, 127
 names of, for passing log events 124
 passing log events to 123
 patterns and 100
 processing in DATA step 113
 referencing in loggers 14
 RollingFileAppender 48
 SAS appenders for server logging 13
 sLogAppender 56
 syntax 11
 UNXFacilityAppender 57
 WindowsEventAppender 59
 XML elements for configuring 11
 ZOSFacilityAppender 60

 ZOSWtoAppender 63
 application messages
 directing to z/OS consoles 63
 ARM appender
 example 37
 syntax 34
 syntax description 35
 ARM transaction events 34
 ARMAppender 34
 attributes 92, 121
 autocall macros 97
 examples 108
 FileRefAppender and 97
 initializing 90
 initializing logging environment for 98

B

best practices 18

C

client applications
 adjusting logging levels in 17
 enabling 56
 viewing logging messages in 17
 Component Interface 121
 component objects 121
 dot notation and 122
 configuration
 changing 18
 for SAS server logging 17
 logging configuration 5
 XML elements for configuring appenders 11
 configuration file 28
 sample files 29
 structure of 28
 syntax conventions 25
 typographical conventions 25
 XML 21
 XML elements for 26
 console
 directing application messages to 63
 writing messages to 38
 ConsoleAppender 38
 conversion characters 68
 conversion patterns 65
 conversion specifiers 65

current console
writing messages to 38

D

DATA step
logging facility in 111
processing appenders in 113
processing loggers in 115
DATA step Component Interface 121
DATA step component objects 121
dot notation and 122
dates
filtering for specific date 85
DEBUG level 15, 18
DEBUG messages 103, 125
DEBUG method 125
DECLARE statement, Appender object 126
DECLARE statement, Logger object 128
DenyAllFilter 81
diagnostic levels
UNIX 58
dot notation
component objects and 122
definition 122
syntax 122

E

environment
initializing for autocall macros 98
setting up 6
ERROR level 15
ERROR messages 106, 130
ERROR method 130
event logging
UNIX 57
Windows 59
z/OS 60

F

FATAL level 15
FATAL messages 107, 131
FATAL method 131
FileAppender 40
fileref appenders 112
patterns and 113
thresholds and 113
FileRefAppender
appender objects and 127
autocall macros and 97
functions and 113
files
writing messages to 40
writing messages to rolling files 48
filter classes 16, 77
filtering policy 16, 78
FilteringAppender 43
filters 5, 14
AndFilter 79
based on a string in log event message 84
DenyAllFilter 81
denying messages not meeting specifications 81
examples 84
for specific date 85

for specific user's error messages 84
LevelMatchFilter 81
LevelRangeFilter 82
log events for a single threshold 81
message filtering 16
messages meeting multiple criteria 79
overview 77
range of message thresholds 82
StringMatchFilter 84
subfilters 79
syntax 79
format modifiers 73
examples 73
formatted log event 75
formatting messages 15, 65
functions 111
examples 117
FileRefAppender and 113

H

hierarchical logger names 8

I

INFO level 15
INFO messages 104, 132
INFO method 132
IOM servers
writing messages to 46
IOMServerAppender 46

L

LEVEL attribute 133
LevelMatchFilter 81
LevelRangeFilter 82
levels 5, 14, 18
adjusting in client applications 17
adjusting temporarily 18
creating loggers and 115
defining 133
defining loggers, to inherit the level 32
defining loggers and 102
logger objects and 129
UNIX diagnostic levels 58
log events 5
additivity and passing events 123
creating in SAS programs 94
filtering for a single threshold 81
formatted 75
passing 123, 124
%LOG4SAS autocall macro 98
%LOG4SAS_APPENDER autocall macro 99
LOG4SAS_APPENDER function 112
%LOG4SAS_DEBUG autocall macro 103
%LOG4SAS_ERROR autocall macro 106
%LOG4SAS_FATAL autocall macro 107
%LOG4SAS_INFO autocall macro 104
LOG4SAS_LOGEVENT function 116
%LOG4SAS_LOGGER autocall macro 101
LOG4SAS_LOGGER function 114
%LOG4SAS_TRACE autocall macro 103
%LOG4SAS_WARN autocall macro 105
LOGAPPLNAME= system option 20
LOGCONFIGLOC= system option 21, 90

- logconfig_trace.xml file 18
- logger categories 94
- logger objects
 - additivity and 129
 - creating an instance of 128
 - declaring 128
 - initializing data for 128
 - levels and 129
- loggers 7
 - associating appenders with 91
 - creating 114
 - creating in SAS programs 91
 - defining 101
 - defining to inherit the level 32
 - defining to log error messages for application in production 32
 - definition 5
 - examples 32
 - hierarchical logger names 8
 - in SAS language 10
 - logging messages with a specific logger 116
 - names of 101, 114, 116, 129
 - overview 31
 - processing in DATA step 115
 - referencing appenders in 14
 - SAS server logger names 9
 - syntax 31
 - syntax description 31
 - updating attributes 92
 - XML elements for configuring 7
- logging configuration 5
- logging configuration file
 - See* configuration file
- logging facility 4
 - compared with SAS log 4
 - enabling for SAS server logging 19
 - enabling in SAS programs 19
 - in DATA step 111
 - in SAS language 89
 - initializing autocall macros 90
 - initializing for SAS programs 90
 - 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) 63
- messages
 - directing to z/OS consoles 63
 - error messages for applications in production 32
 - filtering 16
 - formatting 15
 - formatting for appender classes 65
 - logging with a specific logger 116
 - viewing in client applications 17
 - writing to current console 38
 - writing to IOM servers 46
 - writing to rolling files 48
 - writing to specified file 40
- methods 121

- middle tier applications 56

N

- names
 - appender names 100, 112, 124, 127
 - hierarchical logger names 8
 - logger names 101, 114, 116, 129
 - SAS server logger names 9
 - SAS session names 19, 20

O

- operating system consoles
 - directing application messages to 63

P

- passing log events 123, 124
- pattern layouts 15
 - appender objects and 127
 - conversion characters and 68
 - defining appenders and 100
 - definition 5
 - example 74
 - fileref appenders and 113
 - format modifiers and 73
 - overview 65
 - syntax 66
 - syntax description 66
- performance data
 - based on ARM standards 34

R

- referencing appenders in loggers 14
- reserved class 56
- rolling files
 - writing messages to 48
- RollingFileAppender 48

S

- SAS Data Integration Studio
 - logging messages and levels 18
- SAS language
 - appenders in 13
 - loggers in 10
 - logging facility in 89
 - message categories in 92
- SAS log
 - compared with logging facility 4
- SAS Management Console
 - logging messages and levels 17
- SAS programs
 - creating and using appenders in 90
 - creating log events in 94
 - creating loggers in 91
 - enabling logging facility in 19
 - initializing logging facility for 90
- SAS server logging
 - adjusting logging levels in client applications 17
 - best practices for 18
 - enabling logging facility for 19
 - initial configuration for 17

- viewing logging messages in client applications 17
- SAS servers
 - appenders for 13
 - logger names 9
- SAS session names 19, 20
- sLogAppender 56
- StringMatchFilter 84
- subfilters 79
- syntax conventions 25

T

- terminology 5
- thresholds 14
 - appender objects and 127
 - defining appenders and 100
 - definition 5
 - fileref appenders and 113
 - filtering log events for a single threshold 81
 - filters for a range of 82
- TRACE level 15, 18
- TRACE messages 103, 134
- TRACE method 134
- typographical conventions 25

U

- UNIX
 - diagnostic levels 58
 - event logging on 57
 - writing messages to current console 38

- UNXFacilityAppender 57
- updates
 - updating logger attributes 92
- users 4
 - filter for specific user's error messages 84

W

- WARN level 15
- WARN messages 105, 135
- WARN method 135
- Windows
 - event logging on 59
 - writing messages to current console 38
- WindowsEventAppender 59
- write-to-operator (WTO) service macro 63

X

- XML configuration file 21
- XML elements
 - for configuration file 26
 - for configuring appenders 11

Z

- z/OS
 - directing application messages to console 63
 - event logging on 60
- ZOSFacilityAppender 60
- ZOSWtoAppender 63

Your Turn

We welcome your feedback.

- ☐ If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- ☐ If you have comments about the software, please send them to **suggest@sas.com**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn



**THE
POWER
TO KNOW®**