Cisco Unified JTAPI Alarms and Services

The Cisco Unified JTAPI alarms and services consists of a set of classes and interfaces that expose the additional functionality not readily exposed in JTAPI 1.2 specification but are available in Cisco Unified Communications Manager. Developers can use the classes and interfaces to create new applications or modify existing classes and interfaces to create new methods.

This chapter describes the alarms and services that are available for implementation in a Cisco Unified Communications Manager.

For information about Cisco Unified JTAPI extensions, see Cisco Unified JTAPI Extensions

- Alarm Class Hierarchy, on page 2
- AlarmManager, on page 2
- AlarmWriter, on page 4
- DefaultAlarm, on page 6
- DefaultAlarmWriter, on page 8
- ParameterList, on page 12
- Alarm Interface Hierarchy, on page 14
- Alarm, on page 14
- AlarmWriter, on page 19
- Services Tracing Class Hierarchy, on page 21
- BaseTraceWriter, on page 21
- ConsoleTraceWriter, on page 25
- LogFileTraceWriter, on page 27
- OutputStreamTraceWriter, on page 33
- SyslogTraceWriter, on page 36
- TraceManagerFactory, on page 38
- Services Tracing Interface Hierarchy, on page 40
- Trace, on page 40
- ConditionalTrace, on page 47
- UnconditionalTrace, on page 48
- TraceManager, on page 49
- TraceModule, on page 53
- TraceWriter, on page 54
- TraceWriterManager, on page 57
- Tracing Implementation Class Hierarchy, on page 58
- TraceImpl, on page 59

• ConditionalTraceImpl, on page 61
• UnconditionalTraceImpl, on page 62
• TraceManagerImpl, on page 63
• TraceWriterManagerImpl, on page 67

Alarm Class Hierarchy

The following class hierarchy is contained in the com.cisco.services.alarm package.

```
java.lang.Object
   com.cisco.services.alarm.AlarmManager, on page 2
com.cisco.services.alarm.DefaultAlarm, on page 6 (implements
                                                  com.cisco.services.alarm.Alarm)
com.cisco.services.alarm.DefaultAlarmWriter, on page 8 (implements
                           com.cisco.services.alarm.AlarmWriter, on page 19)
com.cisco.services.alarm.ParameterList, on page 12
```

AlarmManager

The AlarmManager is used to create Alarm objects. The AlarmManager is created with a facility and AlarmService hostname and port. All alarms created by the factory will be associated with this facility. This class also maintains a reference to a single AlarmWriter that can be used system wide. An application can make use of this AlarmWriter. AlarmManager exposes a default implementation of an AlarmWriter. Applications can override this with a user defined implementation of their own AlarmWriter.

Usage

AlarmManager AlarmManager = new AlarmManager(facilityName, alarmServiceHost, alarmServicePort, debugTrace, errorTrace);

Alarms are created by the factory by supplying the alarmName (mnemonic), subfacility and severity Alarms can be cached for use in different parts of the application. During a send alarm applications can specify the variable parameters that offer specific information to the AlarmService.

Usage

Typically applications will maintain their own AlarmManager instance. Applications will also have to set a debug and error trace to enable the alarm tracing to also be sent to the existing trace destinations.

Setup the manager and writer classes:

AlarmWriter alarmWriter = new DefaultAlarmWriter(port, alarmServiceHost);

AlarmManager alarmManager = new AlarmManager(“AA_IVR”, alarmWriter, debugTrace, errorTrace);

Generating the Alarms:

create an alarm for the subfacility and a default severity.

Alarm alarm = alarmManager.createAlarm(“HTTPSS”, Alarm.INFORMATIONAL);

alarm.send(“090T”) sends the alarm with the mnemonic

alarm.send(“090T”, “Port is stuck”, “CTIPort01”) or with a mnemonic and parameter

Declaration

```
public class AlarmManager

java.lang.Object
 |
 +--com.cisco.services.alarm.AlarmManager
```

Note More than one parameter can be sent by specifying a ParameterList

Member summary	
Constructors	
	AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace), on page 3 Create an instance of the AlarmManager for the facility.
Methods	
`Alarm`	createAlarm(String, int), on page 4 Creates an Alarm of required severity for the subFacility
`AlarmWriter`	getAlarmWriter(), on page 4
`void`	setAlarmWriter(AlarmWriter), on page 4 Allows applications to override the AlarmWriter to be used by this AlarmManager, with a user defined AlarmWriter

Inherited member summary
Methods inherited from class `Object`
`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace)

```
public AlarmManager(java.lang.String facility,
  com.cisco.services.alarm.AlarmWriterwriter,
  com.cisco.services.tracing.TracedebugTrace_,
  com.cisco.services.tracing.UnconditionalTraceerrorTrace_)
```

Create an instance of the AlarmManager for the facility. Applications specify an AlarmWriter to be used by this AlarmManager to send the Alarms to the AlarmService.

Methods

createAlarm(String, int)

```
public com.cisco.services.alarm.Alarm createAlarm
   (java.lang.String subfacility,  intseverity)
```

Creates an Alarm of required severity for the subFacility

Returns:

an object implementing the alarm interface

getAlarmWriter()

```
public com.cisco.services.alarm.AlarmWriter getAlarmWriter()
```

Returns:

an AlarmWriter object

setAlarmWriter(AlarmWriter)

```
public void setAlarmWriter(com.cisco.services.alarm.AlarmWriter writer)
```

Allows applications to override the AlarmWriter to be used by this AlarmManager, with a user defined AlarmWriter

AlarmWriter

An AlarmWriter receives alarm messages and transmits it to the receiving AlarmService on a TCP link. This interface can be used to implement other AlarmWriters to be used with this implementation of com.cisco.service.alarm A DefaultAlarmWriter is provided with this implementation and can be obtained from the AlarmManager.

Declaration

```
public interface AlarmWriter
```

All Known Implementing Classes

DefaultAlarmWriter, on page 8

Member Summary

Member summary	
Methods	
`void`	close(), on page 5 close the AlarmWriter

Member summary	
`java.lang.String`	getDescription(), on page 5
`boolean`	getEnabled(), on page 5
`java.lang.String`	getName(), on page 5
`void`	send(String), on page 5 Send out the alarm message to the AlarmService.
`void`	setEnabled(boolean), on page 6 Enable or disable the AlarmWriter

Methods

close()

```
public void close()
```

close the AlarmWriter

getDescription()

```
public java.lang.String getDescription()
```

Returns:

the AlarmWriter description

getEnabled()

```
public boolean getEnabled()
```

Returns:

the current enabled or disabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Returns:

the AlarmWriter name

send(String)

```
public void send(java.lang.String alarmMessage)
```

Send out the alarm message to the AlarmService.

Parameters:

the - Alarm to be sent

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Enable or disable the AlarmWriter

Parameters:

enable or disable the AlarmWriter

DefaultAlarm

An Implementation of the Alarm interface. The AlarmManager creates these Alarms when the createAlarm() method is called.

Declaration

```
public class DefaultAlarm implements Alarm, on page 14

java.lang.Object
|
+--com.cisco.services.alarm.DefaultAlarm
```

All Implemented Interfaces

Alarm, on page 14

Member Summary

Member summary	
Constructors	
	DefaultAlarm(String, String, int, AlarmWriter), on page 7
Methods	
`java.lang.String`	getFacility(), on page 7
`int`	getSeverity(), on page 7
`java.lang.String`	getSubFacility(), on page 7
`void`	send(String), on page 7 Send the alarm with the specified mnemonic
`void`	send(String, ParameterList), on page 8 Send the alarm with the specified name and list of parameters.
`void`	send(String, String, String), on page 8 Send the alarm with the specified name and parameter

Inherited member summary
Fields inherited from interface Alarm, on page 14
ALERTS, on page 17, CRITICAL, on page 17, DEBUGGING, on page 17, EMERGENCIES, on page 17, ERROR, on page 17, HIGHEST_LEVEL, on page 17, INFORMATIONAL, on page 18, LOWEST_LEVEL, on page 18, NOTIFICATION, on page 18, NO_SEVERITY, on page 18, UNKNOWN_MNEMONIC, on page 18, WARNING, on page 18
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

DefaultAlarm(String, String, int, AlarmWriter)

```
public DefaultAlarm(java.lang.String facility,
   java.lang.StringsubFacility,
   intseverity,
   com.cisco.services.alarm.AlarmWriteralarmWriter)
```

Methods

getFacility()

```
public java.lang.String getFacility()
```

Specified By:

getFacility(), on page 18 in interface Alarm, on page 14

getSeverity()

```
public int getSeverity()
```

Specified By:

getSeverity(), on page 18 in interface Alarm, on page 14

getSubFacility()

```
public java.lang.String getSubFacility()
```

Specified By:

getSubFacility(), on page 19 in interface Alarm, on page 14

send(String)

```
public void send(java.lang.String mnemonic)
```

Send the alarm with the specified mnemonic

Specified By:

send(String), on page 19 in interface Alarm, on page 14

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
   com.cisco.services.alarm.ParameterListparamList)
```

Send the alarm with the specified name and list of parameters.

Specified By:

send(String, ParameterList), on page 19 in interface Alarm, on page 14

send(String, String, String)

```
public void send(java.lang.String mnemonic,
   java.lang.StringparamName,
   java.lang.StringparamValue)
```

Send the alarm with the specified name and parameter

Specified By:

send(String, String, String), on page 19 in interface Alarm, on page 14

DefaultAlarmWriter

`DefaultAlarmWriter` implementation of the AlarmWriter interface.

`DefaultAlarmWriter` maintains a queue of a fixed size to which the alarms are written. The sending of the alarms to the alarm service takes place on a separate thread. The queue is fixed size.

Declaration

```
public class DefaultAlarmWriter implements AlarmWriter, on page 4

java.lang.Object
|
+--com.cisco.services.alarm.DefaultAlarmWriter
```

All Implemented Interfaces

AlarmWriter, on page 4

Member Summary

Member summary	
Constructors	
	DefaultAlarmWriter(int, String), on page 9 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and a queue size of fifty (50).

Member summary	
	DefaultAlarmWriter(int, String, int), on page 10 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size.
	DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace), on page 10 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size.
Methods	
`void`	close(), on page 10 Shutdown the send thread and close the socket
`java.lang.String`	getDescription(), on page 11
`boolean`	getEnabled(), on page 11
`java.lang.String`	getName(), on page 11
`static void`	main(String[]), on page 11
`void`	send(String), on page 11 send the Alarm to the alarm service
`void`	setEnabled(boolean), on page 11 Applications can dynamically enable or disable the AlarmWriter

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

DefaultAlarmWriter(int, String)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and a queue size of fifty (50). The AlarmService is listening on this port for Alarm messages.

Parameters:

port: port on which the alarm service is listening

alarmServiceName: The host name of the machine with the Alarm service

Throws:

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName,
   intqueueSize) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size. The AlarmService is listening on this port for Alarm messages.

Parameters:

port—port on which the alarm service is listening

alarmServiceName—The host name of the machine with the Alarm service

queueSize - the size of the queue to be maintained in the alarm writer

Throws:

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName,
   intqueueSize,
   com.cisco.services.tracing.ConditionalTracedebugTrace_,
   com.cisco.services.tracing.UnconditionalTraceerrorTrace_) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size. The AlarmService is listening on this port for Alarm messages.

Parameters:

port—port on which the alarm service is listening

alarmServiceName—The host name of the machine with the Alarm service

queueSize - the size of the queue to be maintained in the alarm writer

Throws:

java.net.UnknownHostException

Methods

close()

```
public void close()
```

Shutdown the send thread and close the socket

Specified By:

close in interface AlarmWriter, on page 4

getDescription()

```
public java.lang.String getDescription()
```

Specified By:

getDescription in interface AlarmWriter, on page 4

Returns:

a short description of the AlarmWriter

getEnabled()

```
public boolean getEnabled()
```

Specified By:

getEnabled in interface AlarmWriter, on page 4

Returns:

the enabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Specified By:

getName in interface AlarmWriter, on page 4

Returns:

the name of the AlarmWriter

main(String[])

```
public static void main(java.lang.String[] args)
```

send(String)

```
public void send(java.lang.String alarmMessage)
```

send the Alarm to the alarm service

Specified By:

send in interface AlarmWriter, on page 4

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Applications can dynamically enable or disable the AlarmWriter

Specified By:

setEnabled in interface AlarmWriter, on page 4

ParameterList

ParameterList is a list of name value pairs that is used to send additional (and optional) user defined parameters to the AlarmService. These parameters can contain the specifics of an Alarm.

As an example, a LowResourceAlarm can have a parameter that informs the service which particular resource is low:

name = “CPUUsage”

value = “0.9”

These parameters are user definable but must, however, also be pre-defined in the AlarmService catalog.

Declaration

```
public class ParameterList

java.lang.Object
|
+--com.cisco.services.alarm.ParameterList
```

Member Summary

Member summary	
Constructors	
	ParameterList(), on page 13 Default constructor for the ParameterList
	ParameterList(String, String), on page 13 Constructor that takes a name value pair.
Methods	
`void` addParameter(String, String), on page 13	addParameter(String, String), on page 13 method used to add additional name value pairs (parameters) to the list
`java.lang.String[]`	getParameterNames(), on page 13 Get the parameter names in the list
`java.lang.String`	getParameterValue(String), on page 13 get the value for a parameter
`void`	removeAllParameters(), on page 14 remove all the parameters in the list

Member summary	
`void`	removeParameter(String), on page 14 remove a particular parameter if it is in the list
`java.lang.String`	toString(), on page 14

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

ParameterList()

```
public ParameterList()
```

Default constructor for the ParameterList

ParameterList(String, String)

```
public ParameterList(java.lang.String name,
   java.lang.Stringvalue)
```

Constructor that takes a name value pair.

Methods

addParameter(String, String)

```
public void addParameter(java.lang.String name,
   java.lang.Stringvalue)
```

method used to add additional name value pairs (parameters) to the list

getParameterNames()

```
public java.lang.String[] getParameterNames()
```

Get the parameter names in the list

Returns:

array of parameters

getParameterValue(String)

```
public java.lang.String getParameterValue(java.lang.String parameterName)
```

get the value for a parameter

Returns:

value of a parameter

removeAllParameters()

```
public void removeAllParameters()
```

remove all the parameters in the list

removeParameter(String)

```
public void removeParameter(java.lang.String parameterName)
```

remove a particular parameter if it is in the list

toString()

```
public java.lang.String toString()
```

Overrides:

`toString` in class `Object`

Alarm Interface Hierarchy

The following interface hierarchy is contained in the com.cisco.services.alarm package.

com.cisco.services.alarm.Alarm, on page 14

com.cisco.services.alarm.AlarmWriter, on page 19

Alarm

The Alarm interface is used to define Alarms in. An Alarm has an XML representation that it must adhere to in order to be recognized by the Alarm Service, with a DTD as shown below. An application can implement this interface or use the AlarmFactory to generate Alarms of the correct format. The Alarm is the a specification that needs to be sent to an AlarmService that will take some action based on the Alarm. Using this specification the AlarmService will access definitions available in a catalog. This catalog is maintained by the user requiring the Alarm function to effect the appropriate action for the Alarm. The severity specified the Alarm can over-ride the severity associated with this Alarm in the catalog. If no severity is specified in the Alarm the catalog severity is used.

Alarm severities are derived from Syslog and are defined as follows:

0 = EMERGENCIES System unusable

1 = ALERTS Immediate action needed

2 = CRITICAL Critical conditions

3 = ERROR Error conditions

4 = WARNING Warning conditions

5 = NOTIFICATION Normal but significant condition

6 = INFORMATIONAL Informational messages only

7 = DEBUGGING Debugging messages

Declaration

```
public interface Alarm
```

All Known Implementing Classes

DefaultAlarm, on page 6

Member Summary

Member summary	
Fields	
`static int`	ALERTS, on page 17 The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed) Syslog severity level = 1
`static int`	CRITICAL, on page 17 A critical failure, the application cannot accomplish the tasks required due to this failure, for example, the application cannot open the database to read the device list Syslog severity level = 2
`static int`	DEBUGGING, on page 17 Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled Syslog severity level = 7
`static int`	EMERGENCIES, on page 17 Emergency situation, a system shutdown is necessary Syslog severity level = 0
`static int`	ERROR, on page 17 An error condition of some kind has occurred and the user needs to understand the nature of that failure Syslog severity level = 3

Member summary	
`static int`	HIGHEST_LEVEL, on page 17 The highest trace level, currently this is DEBUGGING with a trace level of 7
`static int`	INFORMATIONAL, on page 18 Information of some form not relating to errors, warnings, audit, or debug Syslog severity level = 6
`static int`	LOWEST_LEVEL, on page 18 The lowest trace level, currently this is EMERGENCIES with a trace level of 0
`static int`	NO_SEVERITY, on page 18 Applications can set this level to generate Alarms without a severity.
`static int`	NOTIFICATION, on page 18 Notification denotes a normal but significant condition Syslog severity level = 5
`static java.lang.String`	UNKNOWN_MNEMONIC, on page 18 String used when a mnemonic is not specified during an Alarm send
`static int`	WARNING, on page 18 Warning that a problem of some form exists but is not keeping the application from completing its tasks Syslog severity level = 4
Methods	
`java.lang.String`	getFacility(), on page 18
`int`	getSeverity(), on page 18
`java.lang.String`	getSubFacility(), on page 19
`void`	send(String), on page 19 send the Alarm with the specified mnemonic.
`void`	send(String, ParameterList), on page 19 send an Alarm with the specified mnemonic and supplied parameter list

Member summary	
`void`	send(String, String, String), on page 19 send an Alarm with the specified mnemonic and with one parameter

Fields

ALERTS

```
public static final int ALERTS
```

The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed)

Syslog severity level = 1

CRITICAL

```
public static final int CRITICAL
```

A critical failure, the application cannot accomplish the tasks required due to this failure, for example, the application cannot open the database to read the device list

Syslog severity level = 2

DEBUGGING

```
public static final int DEBUGGING
```

Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled (Syslog severity level = 7).

EMERGENCIES

```
public static final int EMERGENCIES
```

Emergency situation, a system shutdown is necessary

Syslog severity level = 0

ERROR

```
public static final int ERROR
```

An error condition of some kind has occurred and the user needs to understand the nature of that failure

Syslog severity level = 3

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

The highest trace level, currently this is DEBUGGING with a trace level of 7

INFORMATIONAL

```
public static final int INFORMATIONAL
```

Information of some form not relating to errors, warnings, audit, or debug

Syslog severity level = 6

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

The lowest trace level, currently this is EMERGENCIES with a trace level of 0

NO_SEVERITY

```
public static final int NO_SEVERITY
```

Applications can set this level to generate Alarms without a severity. NOTE: This is only intended for cases where an application wants the AlarmService to use the severity associated with the Alarm in the catalog

NOTIFICATION

```
public static final int NOTIFICATION
```

Notification denotes a normal but significant condition (Syslog severity level = 5).

UNKNOWN_MNEMONIC

```
public static final java.lang.String UNKNOWN_MNEMONIC
```

String used when a mnemonic is not specified during an Alarm send

WARNING

```
public static final int WARNING
```

Warning that a problem of some form exists but is not keeping the application from completing its tasks (Syslog severity level = 4).

Methods

getFacility()

```
public java.lang.String getFacility()
```

Returns:

the facility name of this Alarm

getSeverity()

```
public int getSeverity()
```

Returns:

severity of the alarm, an integer in the range [0-7]

getSubFacility()

```
public java.lang.String getSubFacility()
```

Returns:

the subfacility of this Alarm

send(String)

```
public void send(java.lang.String mnemonic)
```

send the Alarm with the specified mnemonic. If a null or empty String is passed a mnemonic UNK is sent

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
   com.cisco.services.alarm.ParameterListparameterList)
```

send an Alarm with the specified mnemonic and supplied parameter list

send(String, String, String)

```
public void send(java.lang.String mnemonic,
   java.lang.StringparameterName,
   java.lang.StringparameterValue)
```

send an Alarm with the specified mnemonic and with one parameter.

AlarmWriter

An AlarmWriter receives alarm messages and transmits it to the receiving AlarmService on a TCP link. This interface can be used to implement other AlarmWriters to be used with this implementation of com.cisco.service.alarm A DefaultAlarmWriter is provided with this implementation and can be obtained from the AlarmManager.

Declaration

```
public interface AlarmWriter
```

All Known Implementing Classes

DefaultAlarmWriter, on page 8

Member Summary

Member summary	
Methods	
`void`	close(), on page 5 close the AlarmWriter

Member summary	
`java.lang.String`	getDescription(), on page 5
`boolean`	getEnabled(), on page 5
`java.lang.String`	getName(), on page 5
`void`	send(String), on page 5 Send out the alarm message to the AlarmService.
`void`	setEnabled(boolean), on page 6 Enable or disable the AlarmWriter

Methods

close()

```
public void close()
```

close the AlarmWriter

getDescription()

```
public java.lang.String getDescription()
```

Returns:

the AlarmWriter description

getEnabled()

```
public boolean getEnabled()
```

Returns:

the current enabled or disabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Returns:

the AlarmWriter name

send(String)

```
public void send(java.lang.String alarmMessage)
```

Send out the alarm message to the AlarmService.

Parameters:

the Alarm to be sent

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Enable or disable the AlarmWriter

Parameters:

enable or disable the AlarmWriter

Services Tracing Class Hierarchy

The following class hierarchy is contained in the com.cisco.services.tracing package.

```
java.lang.Object
   com.cisco.services.tracing.BaseTraceWriter, on page 21 (implements
                                                 com.cisco.services.tracing.TraceWriter)
   com.cisco.services.tracing.ConsoleTraceWriter, on page 25
   com.cisco.services.tracing.LogFileTraceWriter, on page 27
    com.cisco.services.tracing.OutputStreamTraceWriter, on page 33
   com.cisco.services.tracing.SyslogTraceWriter, on page 36
com.cisco.services.tracing.TraceManagerFactory, on page 38
```

BaseTraceWriter

This abstract class is useful for supplying a default, non-printing TraceWriter to a TraceWriterManager This class must be extended to provide the functionality to trace to different streams. The doPrintln() method must be implemented by the extending class.

Declaration

```
public abstract class BaseTraceWriter implements TraceWriter, on page 54

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 54

Direct Known Subclasses

ConsoleTraceWriter, on page 25, LogFileTraceWriter, on page 27, OutputStreamTraceWriter, on page 33, SyslogTraceWriter, on page 36

Member Summary

Member summary	
Constructors	
`protected`	BaseTraceWriter(int[], String, String), on page 23 BaseTraceWriter with trace levels as passed in traceLevels in the array falling outside the range Trace.LOWEST_LEVEL and Trace.HIGHEST_LEVEl are ignored
`protected`	BaseTraceWriter(int, String, String), on page 23 BaseTraceWriter that traces all levels up to the maxTraceLevel The trace level is maintained in the range [Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL]
`protected`	BaseTraceWriter(String, String), on page 23 BaseTraceWriter which only traces the lowest level i.e. severity level, Trace.LOWEST_LEVEL messages
Methods	
`void`	close(), on page 23
`protected void`	doClose(), on page 24
`protected void`	doFlush(), on page 24
`protected abstract void`	doPrintln(String, int), on page 24 Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality
`void`	flush(), on page 24
`java.lang.String`	getDescription(), on page 24
`boolean`	getEnabled(), on page 24
`java.lang.String`	getName(), on page 24
`int[]`	getTraceLevels(), on page 24
`void`	println(String, int), on page 25
`void`	setTraceLevels(int[]), on page 25
`java.lang.String`	toString(), on page 25

Inherited member summary
Methods inherited from class `Object`
clone() , equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

BaseTraceWriter(int[], String, String)

```
protected BaseTraceWriter(int[] traceLevels,
   java.lang.Stringname,
   java.lang.Stringdescription)
```

BaseTraceWriter with trace levels as passed in traceLevels in the array falling outside the range Trace.LOWEST_LEVEL and Trace.HIGHEST_LEVEl are ignored

Parameters:

traceLevels - array of trace levels

See Also:

Trace, on page 40

BaseTraceWriter(int, String, String)

```
protected BaseTraceWriter(int maxTraceLevel,
   java.lang.Stringname,
   java.lang.Stringdescription)
```

BaseTraceWriter that traces all levels up to the maxTraceLevel The trace level is maintained in the range [Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL]

See Also:

Trace, on page 40

BaseTraceWriter(String, String)

```
protected BaseTraceWriter(java.lang.String name,
   java.lang.Stringdescription)
```

BaseTraceWriter which only traces the lowest level i.e. severity level, Trace.LOWEST_LEVEL messages

See Also:

Trace, on page 40

Methods

close()

```
public final void close()
```

Description copied from interface:

com.cisco.services.tracing.TraceWriter

Releases any resources associated by this TraceWriter.

Specified By:

`close` in interface TraceWriter, on page 54

doClose()

```
protected void doClose()
```

doFlush()

```
protected void doFlush()
```

doPrintln(String, int)

```
protected abstract void doPrintln(java.lang.String message,
   intmessageNumber)
```

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

flush()

```
public final void flush()
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

Forces output of any messages that have been printed using the `println` method

Specified By:

`flush` in interface TraceWriter, on page 54

getDescription()

```
public final java.lang.String getDescription()
```

Specified By:

`getDescription` in interface TraceWriter, on page 54

getEnabled()

```
public boolean getEnabled()
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

Returns whether the println method will print anything or not. A closed TraceWriter will always return `false` from this method.

Specified By:

`getEnabled` in interface TraceWriter, on page 54

getName()

```
public final java.lang.String getName()
```

Specified By:

`getName` in interface TraceWriter, on page 54

getTraceLevels()

```
public final int[] getTraceLevels()
```

Specified By:

`getTraceLevels` in interface TraceWriter, on page 54

println(String, int)

```
public final void println(java.lang.String message,
   intseverity)
```

Description copied from interface: `com.cisco.services.tracing.TraceWriter`

Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream. Each trace writer has a notion of the highest level trace it traces.

Specified By:

`println` in interface TraceWriter, on page 54

setTraceLevels(int[])

```
public final void setTraceLevels(int[] levels)
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

set the trace levels that will be traced by this TraceWriter

Specified By:

`setTraceLevels` in interface TraceWriter, on page 54

toString()

```
public final java.lang.String toString()
```

Overrides:

`toString` in class `Object`

ConsoleTraceWriter

Supplies a console TraceWriter to trace to System.out.

See Also:

Trace, on page 40

Declaration

```
public final class ConsoleTraceWriter extends BaseTraceWriter

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.ConsoleTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 54

Member Summary

Member summary	
Constructors	
	ConsoleTraceWriter(), on page 26 Default constructor, traces all severity levels
	ConsoleTraceWriter(int), on page 27 Constructor that sets the maximum level to be traced.
	ConsoleTraceWriter(int[]), on page 27 Construct a ConsoleTraceWriter with an array of trace levels Only traces with the severity in the tracelevel array are traced
Methods	
`protected void`	doFlush(), on page 27
`protected void`	doPrintln(String, int), on page 27
`static void`	main(String[]), on page 27

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 21
close(), on page 23, doClose(), on page 24, flush(), on page 24, getDescription(), on page 24, getEnabled(), on page 24, getName(), on page 24, getTraceLevels(), on page 24, println(String, int), on page 25, setTraceLevels(int[]), on page 25, toString(), on page 25
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

ConsoleTraceWriter()

`public` **`ConsoleTraceWriter`**`()`

Default constructor, traces all severity levels

ConsoleTraceWriter(int)

```
public ConsoleTraceWriter(int maxTraceLevel)
```

Constructor that sets the maximum level to be traced.

See Also:

Trace, on page 40

ConsoleTraceWriter(int[])

```
public ConsoleTraceWriter(int[] traceLevels)
```

Construct a ConsoleTraceWriter with an array of trace levels Only traces with the severity in the tracelevel array are traced

Parameters:

`int` - [] traceLevels

See Also:

Trace, on page 40

Methods

doFlush()

```
protected final void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 21

doPrintln(String, int)

```
protected final void doPrintln(java.lang.String message,
   intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 21

main(String[])

```
public static void main(java.lang.String[] args)
```

LogFileTraceWriter

This class extends the BaseTraceWriter class to implement a TraceWriter that writes to a set of log files, rotating among them as each becomes filled to a specified capacity and stores them in a specified directory.

Each of the log files is named according to a pattern controlled by three properties, CurrentFile, FileNameBase, and FileExtension. The CurrentFile property determines which log file, by ordinal number, is being written at present, the FileNameBase property determines the prefix of each log file name, and the FileExtension property determines the suffix, e.g. “txt”. From these properties, log files are named **FileNameBase LeadingZeroPadding CurrentFile.FileExtension**. The CurrentFile property takes on a value from 1 to the value of the MaxFiles property. Note that the CurrentFile property, when converted to a String, is padded with leading zeroes depending on the values of the MaxFiles and CurrentFile properties. An index file tracks the index of the last file written. If the logFileWriter is recreated (for example if an application is restarted) new files will continue from the last written index.

Where the log files are stored is determined by the path, dirNameBase, useSameDir. If a path is not specified, the current path is used as default. If a dirNameBase is not specified, it write log files in the path. Depending upon whether useSameDir is true or false, files are written to the same directory or a new directory, each time an instance of LogFileTraceWriter is created. In case new directories are being made each time, the directory name will consist of the dirNameBase and a number, separated by an ’_’. The number is one more than the greatest number associated with directories with the same dirNameBame in the path. While specifying the path, you may use either a “/” or “\\”, but not “\”

The LogFileTraceWriter keeps track of how many bytes have been written to the current log file. When that number grows within approximately LogFileTraceWriter.ROLLOVER_THRESHOLD bytes, tracing continues to the next file, which is either CurrentFile + 1 if CurrentFile is not equal to MaxFiles, or 1 if CurrentFile is equal to MaxFiles.

Note All properties of this class are specified in the constructor; there is no way to change them dynamically. **Caveat:** If two instances of LogFileTraceWriter are created with the same path and dirNameBase, and useSameDir is true, they may write to the same file.

Example

The following code instantiates a LogFileTraceWriter that will create log files called “MyLog01.log” through “MyLog12.log”. Each file will grow to approximately 100K bytes in size before the next file is created:

LogFileTraceWriter out = new LogFileTraceWriter (“MyLog”, “log”, 12, 100 * 1024); will create a log file TraceWriter which will rotate traces to 12 files from Mylog01.log and Mylog12.log with a file size of 100 KBytes. By default the tracing is set to the HIGHEST_LEVEL.

The following code constructs a LogFileTraceWriter which stores the log files in the path “c:/LogFiles” in a sub directory, “Run”. The files will be named MyLogXX.log. The number of rotating files will be 12 with a size of 100 KB. The same directory gets used for each instance of the application.

LogFileTraceWriter out = new LogFileTraceWriter (“c:/logFiles”, “Run”, “MyLog”, “log”, 12, 100*1024, true);

See Also

Trace, on page 40

Declaration

```
public final class LogFileTraceWriter extends BaseTraceWriter, on page 21

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.LogFileTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 54

Member Summary

Member summary	
Fields	
`static java.lang.String`	DEFAULT_FILE_NAME_BASE, on page 30
`static java.lang.String`	DEFAULT_FILE_NAME_EXTENSION, on page 31
`static char`	DIR_BASE_NAME_NUM_SEPERATOR, on page 31
`static int`	MIN_FILE_SIZE, on page 31
`static int`	MIN_FILES, on page 31
`static int`	ROLLOVER_THRESHOLD, on page 31
Constructors	
	LogFileTraceWriter(String, String, int, int), on page 31 Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.
	LogFileTraceWriter(String, String, String, String, int, int, boolean), on page 31 Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.
	LogFileTraceWriter(String, String, String, String, int, int, int, boolean), on page 31 Constructs a LogFileTraceWriter that rotates among an arbitrary number of files storing them in a specified directory.
Methods	
`protected void`	doClose(), on page 32 Closes this OutputStream.

Member summary	
`protected void`	doFlush(), on page 32
`protected void`	doPrintln(String, int), on page 32
`int`	getCurrentFile(), on page 32 Returns the CurrentFile property
`java.lang.String`	getFileExtension(), on page 32 Returns the FileExtension property
`java.lang.String`	getFileNameBase(), on page 33 Returns the FileNameBase property
`java.lang.String`	getHeader(), on page 33 Get the header string that will be written at the beginning of each log file.
`int`	getMaxFiles(), on page 33 Returns the MaxFiles property
`int`	getMaxFileSize(), on page 33 Returns the MaxFileSize property
`void`	setHeader(String), on page 33 Set the constant header string that will be written at the beginning of every file, trace writing continues from the next line after the header is written.

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 21
close(), on page 23, doClose(), on page 24, flush(), on page 24, getDescription(), on page 24, getEnabled(), on page 24, getName(), on page 24, getTraceLevels(), on page 24, println(String, int), on page 25, setTraceLevels(int[]), on page 25, toString(), on page 25
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Fields

DEFAULT_FILE_NAME_BASE

```
public static final java.lang.String DEFAULT_FILE_NAME_BASE
```

DEFAULT_FILE_NAME_EXTENSION

```
public static final java.lang.String DEFAULT_FILE_NAME_EXTENSION
```

DIR_BASE_NAME_NUM_SEPERATOR

```
public static final char DIR_BASE_NAME_NUM_SEPERATOR
```

MIN_FILE_SIZE

```
public static final int MIN_FILE_SIZE
```

MIN_FILES

```
public static final int MIN_FILES
```

ROLLOVER_THRESHOLD

```
public static final int ROLLOVER_THRESHOLD
```

Constructors

LogFileTraceWriter(String, String, int, int)

```
public LogFileTraceWriter(java.lang.String fileNameBase,
   java.lang.StringfileNameExtension,
   intmaxFiles,
   intmaxFileSize) throwsIOException
```

Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels. Since a path and Directory Base name is not specified, it writes the files to the current directory without any sub directories.

Throws:

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
   java.lang.StringdirNameBase,
   java.lang.StringfileNameBase,
   java.lang.StringfileNameExtension,
   intmaxFiles,
   intmaxFileSize,
   booleanuseSameDir) throwsIOException
```

Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.

Throws:

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
   java.lang.StringdirNameBase,
   java.lang.StringfileNameBase,
```

```
java.lang.StringfileNameExtension,
intmaxFiles,
intmaxFileSize,
intmaxTraceLevel,
booleanuseSameDir) throwsIOException
```

Constructs a LogFileTraceWriter that rotates among an arbitrary number of files storing them in a specified directory.

Throws:

java.io.IOException

Methods

doClose()

```
protected void doClose()
```

Closes this OutputStream. Any log file that is currently open will be closed as well.

Overrides:

`doClose` in class BaseTraceWriter, on page 21

doFlush()

```
protected void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 21

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
   intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 21

getCurrentFile()

```
public int getCurrentFile()
```

Returns:

the CurrentFile property

getFileExtension()

```
public java.lang.String getFileExtension()
```

Returns:

the FileExtension property

getFileNameBase()

```
public java.lang.String getFileNameBase()
```

Returns:

the FileNameBase property

getHeader()

```
public java.lang.String getHeader()
```

Get the header string that will be written at the beginning of each log file.

Returns:

the Header Property

getMaxFiles()

```
public int getMaxFiles()
```

Returns:

the MaxFiles property

getMaxFileSize()

```
public int getMaxFileSize()
```

Returns:

the MaxFileSize property

setHeader(String)

```
public void setHeader(java.lang.String header)
```

Set the constant header string that will be written at the beginning of every file, trace writing continues from the next line after the header is written. If `setHeader` is called after a file output has started, it will take effect from the next file to be written.

Usage:

```
tm = TraceManagerFactory.registerModule(this);
tw = newLogFileTraceWriter("trace", "log", 10, 1024*1024);
tw.setHeader(header);
tm.getTraceWriterManager().addTraceWriter(tw);
```

OutputStreamTraceWriter

OutputStreamTraceWriter wraps an output stream in a TraceWriter. This simplifies adding custom tracing classes that can co-exist with other TraceWriters.

Declaration

```
public final class OutputStreamTraceWriter extends BaseTraceWriter, on page 21

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.OutputStreamTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 54

Member Summary

Member summary	
Constructors	
	OutputStreamTraceWriter(int, OutputStream), on page 35 Default constructor which is auto-flushing
	OutputStreamTraceWriter(int, OutputStream, boolean), on page 35 Create an OutputStreamTraceWriter
Methods	
`protected void`	doClose(), on page 35
`protected void`	doFlush(), on page 35
`protected void`	doPrintln(String, int), on page 35
`java.io.OutputStream`	getOutputStream(), on page 36

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 21
close(), on page 23, doClose(), on page 24, flush(), on page 24, getDescription(), on page 24, getEnabled(), on page 24, getName(), on page 24, getTraceLevels(), on page 24, println(String, int), on page 25, setTraceLevels(int[]), on page 25, toString(), on page 25
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

OutputStreamTraceWriter(int, OutputStream)

```
public OutputStreamTraceWriter(int maxTraceLevel,
   java.io.OutputStreamoutputStream)
```

Default constructor which is auto-flushing

See Also:

Trace, on page 40

OutputStreamTraceWriter(int, OutputStream, boolean)

```
public OutputStreamTraceWriter(int maxTraceLevel,
   java.io.OutputStreamoutputStream,
   booleanautoFlush)
```

Create an OutputStreamTraceWriter

See Also:

Trace, on page 40

Methods

doClose()

```
protected void doClose()
```

Overrides:

`doClose` in class BaseTraceWriter, on page 21

doFlush()

```
protected void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 21

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,  intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 21

getOutputStream()

```
public java.io.OutputStream getOutputStream()
```

Returns:

the output stream associated with the TraceWriter

SyslogTraceWriter

SyslogTraceWriter refines the BaseTraceWriter to allow tracing to syslog. Cisco syslog specification calls for sending low level traces to a syslog collector in the form of UDP messages. No buffering is done in this TraceWriter. The SyslogTraceWriter makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet.

Declaration

```
public final class SyslogTraceWriter extends BaseTraceWriter, on page 21

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.SyslogTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 54

Member Summary

Member summary	
Constructors	
	SyslogTraceWriter(int, String), on page 37 Default SyslogTraceWriter with a max trace level of INFORMATIONAL
	SyslogTraceWriter(int, String, int), on page 37 SyslogTraceWriter with max trace level specified
	SyslogTraceWriter(int, String, int[]), on page 37 SyslogTraceWriter which takes an array of trace levels.
Methods	
`void`	doClose(), on page 38 Closes the socket

Member summary	
`protected void`	doPrintln(String, int), on page 38 The SyslogTraceWriter makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet.
`static void`	main(String[]), on page 38

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 21
close(), on page 23, doClose(), on page 24, flush(), on page 24, getDescription(), on page 24, getEnabled(), on page 24, getName(), on page 24, getTraceLevels(), on page 24, println(String, int), on page 25, setTraceLevels(int[]), on page 25, toString(), on page 25
Methods inherited from class Object
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

SyslogTraceWriter(int, String)

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector)
```

Default SyslogTraceWriter with a max trace level of INFORMATIONAL

See Also:

Trace, on page 40

SyslogTraceWriter(int, String, int)

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector,
   intmaxTraceLevel)
```

SyslogTraceWriter with max trace level specified

See Also:

Trace, on page 40

SyslogTraceWriter(int, String, int[])

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector,
   int[]traceLevels)
```

SyslogTraceWriter which takes an array of trace levels.

See Also:

Trace, on page 40

Methods

doClose()

```
public void doClose()
```

Closes the socket

Overrides:

`doClose` in class BaseTraceWriter, on page 21

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
   intmessageNumber)
```

The `SyslogTraceWriter` makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet. The portion of the message after a '\r' or '\n' is ignored

Overrides:

`doPrintln` in class BaseTraceWriter, on page 21

main(String[])

```
public static void main(java.lang.String[] args)
```

TraceManagerFactory

The TraceManagerFactory class is a class by which applications obtain a TraceManager object. The TraceModule passed in the constructor is registered in a list. The list can be enumerated using the getModules() method.

Declaration

```
public class TraceManagerFactory

java.lang.Object
|
+--com.cisco.services.tracing.TraceManagerFactory
```

Member Summary

Member summary
Methods

Member summary	
`static java.util.Enumeration`	getModules(), on page 39 Returns an enumeration of the TraceModules registered with this factory.
`static TraceManager`	registerModule(TraceModule), on page 39 Returns an instance of a TraceManager object.
`static TraceManager`	registerModule(TraceModule, String[], TraceWriterManager), on page 39 Returns an instance of a TraceManager object.
`static TraceManager`	registerModule(TraceModule, TraceWriterManager), on page 40 Returns an instance of a TraceManager object.

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods

getModules()

```
public static java.util.Enumeration getModules()
```

Returns an enumeration of the TraceModules registered with this factory.

registerModule(TraceModule)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module)
```

Returns an instance of a TraceManager object. The contained TraceWriterManager will not have any default TraceWriters.

registerModule(TraceModule, String[], TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module,
   java.lang.String[]subFacilities,
   com.cisco.services.tracing.TraceWriterManagertraceWriterManager)
```

Returns an instance of a TraceManager object. Trace output will be redirected to the TraceWriterManager object specified.

registerModule(TraceModule, TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module,
   com.cisco.services.tracing.TraceWriterManagertraceWriterManager)
```

Returns an instance of a TraceManager object. Trace output will be redirected to the TraceWriterManager object specified.

Services Tracing Interface Hierarchy

The following interface hierarchy is contained in the com.cisco.services.tracing package.

com.cisco.services.tracing.Trace, on page 40
   com.cisco.services.tracing.ConditionalTrace, on page 47
com.cisco.services.tracing.UnconditionalTrace, on page 48

com.cisco.services.tracing.TraceManager, on page 49

com.cisco.services.tracing.TraceModule, on page 53

com.cisco.services.tracing.TraceWriter, on page 54

com.cisco.services.tracing.TraceWriterManager, on page 57

Trace

The Trace interface defines the methods that allow application tracing. Trace also defines the standard trace types as specified by Syslog Trace Logging.Syslog currently defines 8 levels of trace. The severity of the message is indicated in the trace as a number ranging between [0-7] (0 and 7 included). Currently 7 is HIGHEST_LEVEL and 0 is the LOWEST_LEVEL trace. All 8 levels are predefined here as static int types for reference in tracing sub-system implementations.

The severities traced are as follows:

0 = EMERGENCIES System unusable

1 = ALERTS Immediate action needed

2 = CRITICAL Critical conditions

3 = ERROR Error conditions

4 = WARNING Warning conditions

5 = NOTIFICATION Normal but significant condition

6 = INFORMATIONAL Informational messages only

7 = DEBUGGING Debugging messages

Declaration

```
public interface Trace
```

All Known Subinterfaces

ConditionalTrace, on page 47UnconditionalTrace, on page 48

Member Summary

Member summary	
Fields	
`static int`	ALERTS, on page 43 The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed) Syslog severity level = 1
`static java.lang.String`	ALERTS_TRACE_NAME, on page 43 String descriptor for ALERTS trace level
`static int`	CRITICAL, on page 17 A critical failure, the application cannot accomplish the tasks required due to this failure, e.g.: the application cant open the database to read the device list Syslog severity level = 2
`static java.lang.String`	CRITICAL_TRACE_NAME, on page 44 String descriptor for CRITICAL trace level
`static int`	DEBUGGING, on page 44 Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled Syslog severity level = 7
`static java.lang.DEBUGGING_TRACE_NAME, on page 44String`	DEBUGGING_TRACE_NAME, on page 44 String descriptor for the DEBUGGING trace level
`static int`	EMERGENCIES, on page 44 Emergency situation, a system shutdown is necessary Syslog severity level = 0
`static java.lang.String`	EMERGENCIES_TRACE_NAME, on page 44 String descriptor for EMERGENCIES trace level

Member summary	
`static int`	ERROR, on page 44 An error condition of some kind has occurred and the user needs to understand the nature of that failure Syslog severity level = 3
`static java.lang.String`	ERROR_TRACE_NAME, on page 44 String descriptor for ERROR trace level
`static int`	HIGHEST_LEVEL, on page 44 The highest trace level, currently this is DEBUGGING with a trace level of 7
`static int`	INFORMATIONAL, on page 45 Information of some form not relating to errors, warnings, audit, or debug Syslog severity level = 6
`static java.lang.String`	INFORMATIONAL_TRACE_NAME, on page 45 String descriptor for INFORMATIONAL trace level
`static int`	LOWEST_LEVEL, on page 45 The lowest trace level, currently this is EMERGENCIES with a trace level of 0
`static int`	NOTIFICATION, on page 45 Notification denotes a normal but significant condition Syslog severity level = 5
`static java.lang.String`	NOTIFICATION_TRACE_NAME, on page 45 String descriptor for NOTIFICATION trace level
`static int`	WARNING, on page 45 Warning that a problem of some form exists but is not keeping the application from completing its tasks Syslog severity level = 4
`static java.lang.String`	WARNING_TRACE_NAME, on page 45 String descriptor for WARNING trace level
Methods	
`java.lang.String`	getName(), on page 45 Returns the name of this Trace object.

Member summary	
`java.lang.String`	getSubFacility(), on page 46 Returns the subFacility of trace
`int`	getType(), on page 46 Returns the type of trace.
`boolean`	isEnabled(), on page 46 Returns the state of this Trace object.
`void`	println(Object), on page 46 Prints the string returned by the Object.toString() method and terminates the line as defined by the system.
`void`	println(String), on page 46 Prints a message in the same format as Trace.print() and terminates the line as defined by the system.
`void`	println(String, Object), on page 46 Prints the string returned by the Object.toString() method and terminates the line as defined by the system.
`void`	println(String, String), on page 47 Prints a message in the same format as Trace.print() and terminates the line as defined by the system.
`void`	setDefaultMnemonic(String), on page 47 Sets a default mnemonic for all messages printed out to this trace.

Fields

ALERTS

```
public static final int ALERTS
```

The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed)

Syslog severity level = 1

ALERTS_TRACE_NAME

```
public static final java.lang.String ALERTS_TRACE_NAME
```

String descriptor for ALERTS trace level

CRITICAL

```
public static final int CRITICAL
```

A critical failure, the application cannot accomplish the tasks required due to this failure, e.g.: the application cant open the database to read the device list

Syslog severity level = 2

CRITICAL_TRACE_NAME

```
public static final java.lang.String CRITICAL_TRACE_NAME
```

String descriptor for CRITICAL trace level

DEBUGGING

```
public static final int DEBUGGING
```

Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled

Syslog severity level = 7

DEBUGGING_TRACE_NAME

```
public static final java.lang.String DEBUGGING_TRACE_NAME
```

String descriptor for the DEBUGGING trace level

EMERGENCIES

```
public static final int EMERGENCIES
```

Emergency situation, a system shutdown is necessary

Syslog severity level = 0

EMERGENCIES_TRACE_NAME

```
public static final java.lang.String EMERGENCIES_TRACE_NAME
```

String descriptor for EMERGENCIES trace level

ERROR

```
public static final int ERROR
```

An error condition of some kind has occurred and the user needs to understand the nature of that failure

Syslog severity level = 3

ERROR_TRACE_NAME

```
public static final java.lang.String ERROR_TRACE_NAME
```

String descriptor for ERROR trace level

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

The highest trace level, currently this is DEBUGGING with a trace level of 7

INFORMATIONAL

```
public static final int INFORMATIONAL
```

Information of some form not relating to errors, warnings, audit, or debug

Syslog severity level = 6

INFORMATIONAL_TRACE_NAME

```
public static final java.lang.String INFORMATIONAL_TRACE_NAME
```

String descriptor for INFORMATIONAL trace level

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

The lowest trace level, currently this is EMERGENCIES with a trace level of 0

NOTIFICATION

```
public static final int NOTIFICATION
```

Notification denotes a normal but significant condition

Syslog severity level = 5

NOTIFICATION_TRACE_NAME

```
public static final java.lang.String NOTIFICATION_TRACE_NAME
```

String descriptor for NOTIFICATION trace level

WARNING

```
public static final int WARNING
```

Warning that a problem of some form exists but is not keeping the application from completing its tasks

Syslog severity level = 4

WARNING_TRACE_NAME

```
public static final java.lang.String WARNING_TRACE_NAME
```

String descriptor for WARNING trace level

Methods

getName()

```
public java.lang.String getName()
```

Returns:

the name of this Trace object

getSubFacility()

```
public java.lang.String getSubFacility()
```

Returns:

the trace subFacility type

getType()

```
public int getType()
```

Returns:

the type of trace as specified in Syslog. DEBUGGING, INFORMATIONAL, WARNING, etc.

isEnabled()

```
public boolean isEnabled()
```

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Returns:

true if tracing is enabled, false otherwise

See Also

ConditionalTrace, on page 47

println(Object)

```
public void println(java.lang.Object object)
```

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Parameters:

`object` - the object to be printed

println(String)

```
public void println(java.lang.String message)
```

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Parameters:

`message` - the message to be printed

println(String, Object)

```
public void println(java.lang.String mnemonic,
   java.lang.Objectobject)
```

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Parameters:

`object` - the object to be printed

`mnemonic` - the mnemonic mapped to message to be printed

println(String, String)

```
public void println(java.lang.String mnemonic,
   java.lang.Stringmessage)
```

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Parameters:

`message` - the message to be printed

`mnemonic` - the mnemonic mapped to message to be printed

setDefaultMnemonic(String)

```
public void setDefaultMnemonic(java.lang.String mnemonic)
```

Sets a default mnemonic for all messages printed out to this trace.

Parameters:

`mnemonic,` - a mnemonic string

ConditionalTrace

The ConditionalTrace interface extends the Trace interface and defines the methods that allow enabling and disabling of tracing for this particular condition.

Typically, applications obtain one ConditionalTrace object for each condition that they need to trace under certain circumstances but not always (for example, AUDIT, INFO, and so on).

Declaration

```
public interface ConditionalTrace extends Trace, on page 40
```

All Superinterfaces

Trace, on page 40

Member Summary

Member summary	
Methods	
`void`	disable(), on page 48 Disables this condition for tracing.
`void`	enable(), on page 48 Enables this condition for tracing.

Inherited member summary
Fields inherited from interface Trace, on page 40
ALERTS, on page 43, ALERTS_TRACE_NAME, on page 43, CRITICAL, on page 17, CRITICAL_TRACE_NAME, on page 44, DEBUGGING, on page 44, DEBUGGING_TRACE_NAME, on page 44, EMERGENCIES, on page 44, EMERGENCIES_TRACE_NAME, on page 44, ERROR, on page 44, ERROR_TRACE_NAME, on page 44, HIGHEST_LEVEL, on page 44, INFORMATIONAL, on page 45, INFORMATIONAL_TRACE_NAME, on page 45, LOWEST_LEVEL, on page 45, NOTIFICATION, on page 45, NOTIFICATION_TRACE_NAME, on page 45, WARNING, on page 45, WARNING_TRACE_NAME, on page 45
Methods inherited from interface Trace, on page 40
getName(), on page 45, getSubFacility(), on page 46, getType(), on page 46, isEnabled(), on page 46, println(Object), on page 46, println(String), on page 46, println(String, Object), on page 46, println(String, String), on page 47, setDefaultMnemonic(String), on page 47

Methods

disable()

```
public void disable()
```

Disables this condition for tracing.

enable()

```
public void enable()
```

Enables this condition for tracing.

UnconditionalTrace

The UnconditionalTrace interface extends the Trace interface. Note that because this object extends Trace, its state is enabled by default and it may not be changed.

Typically, applications would obtain one UnconditionalTrace object per each condition that they need to trace always under any circumstances (such as, ERROR, FATAL, and so on).

Declaration

```
public interface UnconditionalTrace extends Trace, on page 40
```

All Superinterfaces

Trace, on page 40

Member Summary

Inherited Member summary
Fields inherited from interface Trace, on page 40
ALERTS, on page 43, ALERTS_TRACE_NAME, on page 43, CRITICAL, on page 17, CRITICAL_TRACE_NAME, on page 44, DEBUGGING, on page 44, DEBUGGING_TRACE_NAME, on page 44, EMERGENCIES, on page 44, EMERGENCIES_TRACE_NAME, on page 44, ERROR, on page 44, ERROR_TRACE_NAME, on page 44, HIGHEST_LEVEL, on page 44, INFORMATIONAL, on page 45, INFORMATIONAL_TRACE_NAME, on page 45, LOWEST_LEVEL, on page 45, NOTIFICATION, on page 45, NOTIFICATION_TRACE_NAME, on page 45, WARNING, on page 45, WARNING_TRACE_NAME, on page 45
Methods inherited from interface Trace, on page 40
getName(), on page 45, getSubFacility(), on page 46, getType(), on page 46, isEnabled(), on page 46, println(Object), on page 46, println(String), on page 46, println(String, Object), on page 46, println(String, String), on page 47, setDefaultMnemonic(String), on page 47

TraceManager

The TraceManager interface defines the methods that allow applications trace management.

Typically, an application obtains only one TraceManager object. All Trace objects are created by default: Predefined Trace in accordance with Syslog definitions are:

```
ConditionalTraces:INFORMATIONAL, DEBUGGING, NOTIFICATION, WARNING
UnconditionalTraces:ERROR, CRITICAL, ALERTS, EMERGENCIES
```

Facilities/Sub-Facilities:

- **Facility**—A code consisting of two or more uppercase letters that indicate the facility to which the message refers. A facility can be a hardware device, a protocol, or a module of the system software.
- **SubFacility**—A code consisting of two or more uppercase letters that indicate the sub-facility to which the message refers. A sub-facility can be a hardware device component, a protocol unit, or a sub-module of the system software.

By default all 8 Conditional and UnConditional Traces are created for the Facility and 8 for each of the subFacilities In order to use the DEBUGGING trace for the parent FACILITY, for example, the application needs to use the getConditionalTrace(“DEBUGGING”) method of this object.

In order to use the DEBUGGING trace for the SUBFACILITY, for example, the application needs to use the getConditionalTrace(SUBFACILITY + “_” + “DEBUGGING”) method of this object or use the getConditionalTrace(SUBFACILITY, “DEBUGGING”) method.

System wide TraceWriterManager is set through the setTraceWriterManager method provided by this interface.

The Trace Manager object also allows the application to enable or disable tracing for all trace through the enableAll() and disableAll() methods.

Declaration

```
public interface TraceManager
```

Member Summary

Member summary	
Methods	
`void`	addSubFacilities(String[]), on page 51 Sets a set of subFacilities for this TraceManager/Facility.
`void`	addSubFacility(String), on page 51 Adds a single subFacility for this TraceManager/Facility.
`void`	disableAll(), on page 51 Disables tracing for all Trace objects managed by this TraceManager.
`void`	disableTimeStamp(), on page 52 Disables prefixing a time stamp for every message printed by this TraceManager.
`void`	enableAll(), on page 52 Enables tracing for all Trace objects managed by this TraceManager.
`void`	enableTimeStamp(), on page 52 Enables prefixing a time stamp for every message printed by this TraceManager.
`ConditionalTrace`	getConditionalTrace(int), on page 52 Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.
`ConditionalTrace`	getConditionalTrace(String, int), on page 52 Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility
`java.lang.String`	getName(), on page 52 Returns the Facility name for this TraceManager.
`java.lang.String[]`	getSubFacilities(), on page 52 Returns the subFacility names for this TraceManager/Facility.

Member summary	
`java.util.Enumeration`	getTraces(), on page 52 Returns an enumeration of the Trace objects managed by this TraceManager.
`TraceWriterManager`	getTraceWriterManager(), on page 52 Returns the TraceWriter used by this TraceManager.
`UnconditionalTrace`	getUnconditionalTrace(int), on page 52 Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.
`UnconditionalTrace`	getUnconditionalTrace(String, int), on page 53 Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility
`void`	removeTrace(Trace), on page 53 Removes a Trace object given an object.
`void`	setSubFacilities(String[]), on page 53 Sets a set of subFacilities for this TraceManager/Facility.
`void`	setSubFacility(String), on page 53 Adds a single subFacility for this TraceManager/Facility.
`void`	setTraceWriterManager(TraceWriterManager), on page 53 Sets the TraceWriter to be used by this TraceManager.

Methods

addSubFacilities(String[])

```
public void addSubFacilities(java.lang.String[] names)
```

Sets a set of subFacilities for this TraceManager/Facility.

addSubFacility(String)

```
public void addSubFacility(java.lang.String name)
```

Adds a single subFacility for this TraceManager/Facility.

disableAll()

```
public void disableAll()
```

Disables tracing for all Trace objects managed by this TraceManager.

disableTimeStamp()

```
public void disableTimeStamp()
```

Disables prefixing a time stamp for every message printed by this TraceManager.

enableAll()

```
public void enableAll()
```

Enables tracing for all Trace objects managed by this TraceManager.

enableTimeStamp()

```
public void enableTimeStamp()
```

Enables prefixing a time stamp for every message printed by this TraceManager.

getConditionalTrace(int)

```
public com.cisco.services.tracing.ConditionalTrace
   getConditionalTrace(int severity)
```

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.

getConditionalTrace(String, int)

```
public com.cisco.services.tracing.ConditionalTrace
   getConditionalTrace(java.lang.String subFacility,
   intseverity)
```

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility

getName()

```
public java.lang.String getName()
```

Returns the Facility name for this TraceManager.

getSubFacilities()

```
public java.lang.String[] getSubFacilities()
```

Returns the subFacility names for this TraceManager/Facility.

getTraces()

```
public java.util.Enumeration getTraces()
```

Returns an enumeration of the Trace objects managed by this TraceManager.

getTraceWriterManager()

```
public com.cisco.services.tracing.TraceWriterManager getTraceWriterManager()
```

Returns the TraceWriter used by this TraceManager.

getUnconditionalTrace(int)

```
public com.cisco.services.tracing.UnconditionalTrace getUnconditionalTrace(int severity)
```

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.

getUnconditionalTrace(String, int)

```
public com.cisco.services.tracing.UnconditionalTrace
   getUnconditionalTrace(java.lang.String subFacility,
   intseverity)
```

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility

removeTrace(Trace)

```
public void removeTrace(com.cisco.services.tracing.Trace tc)
```

Removes a Trace object given an object.

setSubFacilities(String[])

```
public void setSubFacilities(java.lang.String[] names)
```

Deprecated and replaced with `TraceManager.addSubFacilities` method

Sets a set of subFacilities for this TraceManager/Facility.

setSubFacility(String)

```
public void setSubFacility(java.lang.String name)
```

Deprecated and replaced with `TraceManager.addSubFacility` method

Adds a single subFacility for this TraceManager/Facility.

setTraceWriterManager(TraceWriterManager)

```
public void setTraceWriterManager(com.cisco.services.tracing.TraceWriterManager twm)
```

Sets the TraceWriter to be used by this TraceManager.

TraceModule

The TraceModule interface serves two purposes. First, it allows applications to discover the TraceManager object used by other packages that they use. Second, applications that register with the TraceManagerFactory must identify themselves by implementing this interface.

Declaration

```
public interface TraceModule
```

All Known Subinterfaces

com.cisco.jtapi.extensions.CiscoJtapiPeer

Member Summary

Member summary	
Methods	
`TraceManager`	getTraceManager(), on page 54 Returns the TraceManager that an object is using for tracing.
`java.lang.String`	getTraceModuleName(), on page 54 Returns the module name.

Methods

getTraceManager()

```
public com.cisco.services.tracing.TraceManager getTraceManager()
```

Returns the TraceManager that an object is using for tracing.

getTraceModuleName()

```
public java.lang.String getTraceModuleName()
```

Returns the module name.

TraceWriter

The TraceWriter interface abstracts the details of trace message output. The TraceWriter uses its enabled method to advertise whether or not the print and println methods will have any effect. Users of TraceWriter should use the value returned by the getEnabled method as an indication of whether they should invoke the print and println methods at all.

Declaration

```
public interface TraceWriter
```

All Known Subinterfaces

TraceWriterManager, on page 57

All Known Implementing Classes

BaseTraceWriter, on page 21

Member Summary

Member summary	
Methods	
`void`	close(), on page 55 Releases any resources associated by this TraceWriter .
`void`	flush(), on page 55 Forces output of any messages that have been printed using the println method
`java.lang.String`	getDescription(), on page 55
`boolean`	getEnabled(), on page 56 Returns whether the println method will print anything or not.
`java.lang.String`	getName(), on page 56
`int[]`	getTraceLevels(), on page 56
`void`	println(String, int), on page 56 Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream.
`void`	setTraceLevels(int[]), on page 56 set the trace levels that will be traced by this TraceWriter

Methods

close()

```
public void close()
```

Releases any resources associated by this TraceWriter.

flush()

```
public void flush()
```

Forces output of any messages that have been printed using the println method

getDescription()

```
public java.lang.String getDescription()
```

Returns:

a short description of this TraceWriter

getEnabled()

```
public boolean getEnabled()
```

Returns whether the println method will print anything or not. A closed `TraceWriter` will always return `false` from this method.

Returns:

true if this TraceWriter is enabled, false if not

getName()

```
public java.lang.String getName()
```

Returns:

the name of this TraceWriter

getTraceLevels()

```
public int[] getTraceLevels()
```

Returns:

the array of trace levels that will be traced by this TraceWriter

println(String, int)

```
public void println(java.lang.String message,
   intseverity)
```

Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream. Each trace writer has a notion of the highest level trace it traces

Parameters:

`message` - the string to print

`severity` - of the trace.

See Also

Trace, on page 40

setTraceLevels(int[])

```
public void setTraceLevels(int[] levels)
```

set the trace levels that will be traced by this TraceWriter

Parameters:

`int[]` - levels

See Also

Trace, on page 40

TraceWriterManager

TraceWriterManager contains the list of TraceWriter objects that are used to implement the tracing. The list is populated at startup from the switches in a .ini file. A LogFileTraceWriter, a ConsoleTraceWriter, and a SyslogTraceWriter are available. Users can override the existing TraceWriters by setting a user implemented TraceWriter[] or adding to the existing TraceWriters. This makes it possible to add other TraceWriters that can function along with existing trace writers.

Declaration

```
public interface TraceWriterManager extends TraceWriter, on page 54
```

All Superinterfaces

TraceWriter, on page 54

Member Summary

Member summary	
Methods	
`void`	addTraceWriter(TraceWriter), on page 58 Add another TraceWriter to the array
`TraceWriter[]`	getTraceWriters(), on page 58
`void`	removeTraceWriter(TraceWriter), on page 58 Remove the TraceWriter from the array in the manager
`void`	setTraceWriters(TraceWriter[]), on page 58 Implementations can use this method to override or enhance the provided TraceWriters

Inherited member summary
Methods inherited from interface TraceWriter, on page 54
close(), on page 55, flush(), on page 55, getDescription(), on page 55, getEnabled(), on page 56, getName(), on page 56, getTraceLevels(), on page 56, println(String, int), on page 56, setTraceLevels(int[]), on page 56

Methods

addTraceWriter(TraceWriter)

```
public voidaddTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Add another TraceWriter to the array

Parameters:

`TraceWriter` - to be added to the list

getTraceWriters()

```
public com.cisco.services.tracing.TraceWriter[] getTraceWriters()
```

Returns:

the array of TraceWriters in the manager

removeTraceWriter(TraceWriter)

```
public voidremoveTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Remove the TraceWriter from the array in the manager

setTraceWriters(TraceWriter[])

```
public voidsetTraceWriters(com.cisco.services.tracing.TraceWriter[] traceWriters)
```

Implementations can use this method to override or enhance the provided TraceWriters

Parameters:

`set` - the array of TraceWriters.

Tracing Implementation Class Hierarchy

The following tracing implementation class hierarchy is contained in the com.cisco.services.tracing.implementation package.

```
java.lang.Object
   com.cisco.services.tracing.implementation.TraceImpl, on page 59 (implements
                                                 com.cisco.services.tracing.Trace)
   com.cisco.services.tracing.implementation.ConditionalTraceImpl, on page 61 (implements
                           com.cisco.services.tracing.ConditionalTrace)
  com.cisco.services.tracing.implementation.UnconditionalTraceImpl, on page 62 (implements

                         com.cisco.services.tracing.UnconditionalTrace)
com.cisco.services.tracing.implementation.TraceManagerImpl, on page 63 (implements
                                    com.cisco.services.tracing.TraceManager)
com.cisco.services.tracing.implementation.TraceWriterManagerImpl, on page 67 (implements
                               com.cisco.services.tracing.TraceWriterManager)
```

TraceImpl

Declaration

public abstract class **TraceImpl**

extends java.lang.Object

implements Trace

All Implemented Interfaces

Trace, on page 40

Methods

println

public final void **println**(java.lang.String message)

Description copied from interface: Trace

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

message - the message to be printed

println

public final void println(java.lang.String mnemonic, java.lang.String message)

Description copied from interface: Trace

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

mnemonic - the mnemonic mapped to message to be printed

message - the message to be printed

println

public final void **println**(java.lang.Object object)

Description copied from interface: Trace

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

object - the object to be printed

println

public final void **println**(java.lang.String mnemonic, java.lang.Object object)

Description copied from interface: Trace

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

mnemonic - the mnemonic mapped to message to be printed

object - the object to be printed

getName

public final java.lang.String **getName**()

Description copied from interface: Trace

Returns the name of this Trace object.

Specified by:

getName in interface Trace

Returns:

the name of this Trace object

setDefaultMnemonic

public final void **setDefaultMnemonic**(java.lang.String mnemonic)

Description copied from interface: Trace

Sets a default mnemonic for all messages printed out to this trace.

Specified by:

setDefaultMnemonic in interface Trace

Parameters:

mnemonic - a mnemonic string

getType

public int **getType**()

Description copied from interface: Trace

Returns the type of trace.

Specified by:

getType in interface Trace

Returns:

the trace severity as specified in Syslog. DEBUGGING, INFORMATIONAL, WARNING, etc.

getSubFacility

public java.lang.String **getSubFacility**()

Description copied from interface: Trace

Returns the subFacility of trace

Specified by:

getSubFacility in interface Trace

Returns:

the trace subFacility type

Inherited Methods

isEnabled

ConditionalTraceImpl

Declaration

public final class ConditionalTraceImpl

extends TraceImpl

implements ConditionalTrace

All Implemented Interfaces

ConditionalTrace, Trace

Methods

enable

public void **enable**()

Description copied from interface: ConditionalTrace

Enables this condition for tracing.

Specified by:

enable in interface ConditionalTrace

disable

public void **disable**()

Description copied from interface: ConditionalTrace

Disables this condition for tracing.

Specified by:

disable in interface ConditionalTrace

isEnabled

public boolean **isEnabled**()

Description copied from interface: Trace

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Specified by:

isEnabled in interface Trace

Returns:

true if tracing is enabled, false otherwise

See Also:

ConditionalTrace

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

UnconditionalTraceImpl

Declaration

public final class UnconditionalTraceImpl

extends TraceImpl

implements UnconditionalTrace

All Implemented Interfaces

Trace, UnconditionalTrace

Methods

isEnabled

public boolean **isEnabled**()

Description copied from interface: Trace

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Specified by:

isEnabled in interface Trace

Returns:

true if tracing is enabled, false otherwise

See Also:

ConditionalTrace

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

TraceManagerImpl

The TraceManagerImpl class implements the TraceManager interface.

Declaration

public class **TraceManagerImpl** extends java.lang.Object

java.lang.Object

|

+--com.cisco.services.tracing.implementation.TraceManagerImpl

All Implemented Interfaces

TraceManager, on page 49

Constructors

public **TraceManagerImpl**(java.lang.StringmoduleName, java.lang.String[]subFacilities, TraceWriterManagertraceWriterManager)

public **TraceManagerImpl**(java.lang.StringmoduleName, TraceWriterManagertraceWriterManager)

Methods

getConditionalTrace

```
public ConditionalTrace getConditionalTrace(intseverity)
```

Description copied from interface: TraceManager

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.

Specified by:

`getConditionalTrace` in interface `TraceManager`

getConditionalTrace

```
public ConditionalTrace getConditionalTrace(java.lang.StringsubFacility,
   intseverity)
```

Description copied from interface: TraceManager

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility

Specified by:

`getConditionalTrace` in interface `TraceManager`

getUnconditionalTrace

```
public UnconditionalTrace getUnconditionalTrace(intseverity)
```

Description copied from interface: TraceManager

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.

Specified by:

`getUnconditionalTrace` in interface `TraceManager`

getUnconditionalTrace

```
public UnconditionalTrace getUnconditionalTrace(java.lang.StringsubFacility,
   intseverity)
```

Description copied from interface: TraceManager

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility

Specified by:

`getUnconditionalTrace` in interface `TraceManager`

getTraceWriterManager

```
public TraceWriterManager getTraceWriterManager()
```

Description copied from interface: TraceManager

Returns the TraceWriter used by this TraceManager.

Specified by:

`getTraceWriterManager` in interface `TraceManager`

setTraceWriterManager

```
public void setTraceWriterManager(TraceWriterManagerout)
```

Description copied from interface: TraceManager

Sets the TraceWriter to be used by this TraceManager.

Specified by:

`setTraceWriterManager` in interface `TraceManager`

removeTrace

```
public void removeTrace(Tracetc)
```

Description copied from interface: TraceManager

Removes a Trace object given an object.

Specified by:

`removeTrace` in interface `TraceManager`

getTraces

```
public java.util.Enumeration getTraces()
```

Description copied from interface: TraceManager

Returns an enumeration of the Trace objects managed by this TraceManager.

Specified by:

`getTraces` in interface `TraceManager`

enableAll

```
public void enableAll()
```

Description copied from interface: TraceManager

Enables tracing for all Trace objects managed by this TraceManager.

Specified by:

`enableAll` in interface `TraceManager`

disableAll

```
public void disableAll()
```

Description copied from interface: TraceManager

Disables tracing for all Trace objects managed by this TraceManager.

Specified by:

`disableAll` in interface `TraceManager`

getName

```
public java.lang.String getName()
```

Description copied from interface: TraceManager

Returns the Facility name for this TraceManager.

Specified by:

`getName` in interface `TraceManager`

enableTimeStamp

```
public void enableTimeStamp()
```

Description copied from interface: TraceManager

Enables prefixing a time stamp for every message printed by this TraceManager.

Specified by:

`enableTimeStamp` in interface `TraceManager`

disableTimeStamp

```
public void disableTimeStamp()
```

Description copied from interface: TraceManager

Disables prefixing a time stamp for every message printed by this TraceManager.

Specified by:

`disableTimeStamp` in interface `TraceManager`

getSubFacilities

```
public java.lang.String[] getSubFacilities()
```

Returns the subFacility names for this TraceManager/Facility.

Specified by:

`getSubFacilities` in interface `TraceManager`

addSubFacilities

```
public void addSubFacilities(java.lang.String[]names)
```

Adds subFacilities for this TraceManager/Facility.

Specified by:

`addSubFacilities` in interface `TraceManager`

addSubFacility

`public void addSubFacility(java.lang.Stringname)`

Adds a subFacility for this TraceManager/Facility.

Specified by:

`addSubFacility` in interface `TraceManager`

Deprecated

getSubFacilities(java.lang.String[]names)

Replaced by addSubFacilties(String[]).

setSubFacility(java.lang.Stringname)

Replaced by addSubFacility(String).

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

TraceWriterManagerImpl

TraceWriterManager contains the list of TraceWriter objects that are used to implement the tracing. The list is populated at startup from the switches in a .ini file. A LogFileTraceWriter, a ConsoleTraceWriter, and a SyslogTraceWriter are available. Users can override the existing TraceWriters by setting a user implemented TraceWriter[] or adding to the existing TraceWriters. This makes it possible to add other traceWriters that can function along with exisiting trace writers.

Note Methods inherited from class java.lang.Object are clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

Declaration

public class TraceWriterManagerImpl extends java.lang.Object implements TraceWriterManager

java.lang.Object

com.cisco.services.tracing.implementation.TraceWriterManagerImpl

All Implemented Interfaces

TraceWriter, TraceWriterManager

Constructors

TraceWriterManagerImpl

```
public TraceWriterManagerImpl()
```

Creates a TraceWriterManagerImpl with a zero length TraceWriter array .

Methods

setTraceWriters

```
public void setTraceWriters(TraceWriter[]traceWriters)
```

Overrides the existing TraceWriters with a new user supplied set .

Specified by:

`setTraceWriters` in interface `TraceWriterManager`

Parameters:

traceWriters - An array of TraceWriters.

getTraceWriters

```
public TraceWriter[] getTraceWriters()
```

Returns the array of TraceWriters currently in use .

Specified by:

`getTraceWriters` in interface `TraceWriterManager`

Returns:

The array of TraceWriters in the manager.

addTraceWriter

```
public void addTraceWriter(TraceWritertw)
```

Add this TraceWriter to the array of trace writers

Specified by:

`addTraceWriter` in interface `TraceWriterManager`

Parameters:

tw - TraceWriter to be added to the list

removeTraceWriter

```
public void removeTraceWriter(TraceWritertw)
```

Remove the Tracewriter from the array of trace writers.

Specified by:

`removeTraceWriter` in interface `TraceWriterManager`

println

```
public void println(java.lang.Stringmessage,  intseverity)
```

All traces invoke this method. A trace supplies its severity along with the message. Traces below the threshold severity of the TraceWriter are allowed. Eg. If the Threshhold severity is set to INFORMATIONAL (level = 6) DEBUG traces will not be passed by the TraceWriter. The severity level is set in the constructor of the TraceWriter

Specified by:

println in interface TraceWriter

Parameters:

`message` - The string to print

`severity` - The severity of the trace.

See Also:

Trace

Flush

public void flush()

Description copied from interface: TraceWriter

Forces output of any messages that have been printed using the println method

Specified by:

`flush` in interface `TraceWriter`

close

```
public void close()
```

Description copied from interface: TraceWriter

Releases any resources associated by this TraceWriter.

Specified by:

`close` in interface `TraceWriter`

getEnabled

```
public boolean getEnabled()
```

Returns true if any one of the underlying TraceWriter is enabled, else returns false.

Specified by:

`getEnabled` in interface `TraceWriter`

Returns:

`True` if this TraceWriter is enabled, `false` if not.

getName

```
public java.lang.String getName()
```

Specified by:

`getName` in interface `TraceWriter`

Returns:

The name of this TraceWriter.

getDescription

```
public java.lang.String getDescription()
```

Specified by:

`getDescription` in interface `TraceWriter`

Returns:

A short description of this TraceWriter.

setTraceLevels

```
public void setTraceLevels(int[]levels)
```

The TraceWriterManager does nothing for this method .

Specified by:

`setTraceLevels` in interface `TraceWriter`

Parameters:

Levels - Array of trace levels.

See Also:

Trace

getTraceLevels

```
public int[] getTraceLevels()
```

The TraceWriterManager returns a null, as the traceLevel is maintained at the individual TraceWriter .

Specified by:

`getTraceLevels` in interface `TraceWriter`

Returns:

`null`