CTI OS Logging

CTI OS Client Logs (COM and C++)

If you install the tracing mechanism, the COM and C++ CILs automatically create a log file and trace to it. The trace log file name and location for client processes is found under the following Windows registry key:

```
HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc.\CTIOS Tracing
```

The default filename is CtiosClientLog. Log files are created using the convention <TraceFileName>.<Windows user name>.mmdd.hhmmss.log. The files are created in the current directory of the executing program, such as the directory into which you install the Agent Desktop. You can provide a fully qualified path for the TraceFileName if you wish to store the files in a different location. For example, setting the following value stores the log files in the directory C:\Temp, using the naming convention CtiosClientLog.<Windows user name>.mmdd.hhmmss.log.

```
C:\Temp\CtiosClientLog
```

Client trace files are formatted in ASCII text that you can open them with a conventional text editor such as Notepad.

Install Tracing Mechanism (COM and C++)

To install the tracing mechanism:

Procedure

Step 1 Copy the tracing executable, ctiostracetext.exe, from the distribution media to the folder in which your application is located.

Step 2 Open a command window and register the tracing mechanism:

Step 3 ctiostracetext.exe /regserver

Set Trace Levels (COM and C++)

You must set the trace level in the registry by creating a TraceMask registry value within the HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc.\CTIOS Tracing key and setting its value to 0x40000307.

```
[HKEY_CURRENT_USER\Software\Cisco Systems, Inc.\CTIOS Tracing]"TraceMask"=dword:40000307
```

Trace levels for client processes, such as the Agent Desktop phone, are stored under the following registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems, Inc.\CTIOS Tracing]
"TraceFileName"="%HOMEPATH%\\CtiOsClientLog" "TraceMask"=dword:00000000
"MaxDaysBeforeExpire"=dword:00000007 "MaxFiles"=dword:00000005 "MaxFileSizeKb"=dword:00000800
"FlushIntervalSeconds"=dword:0000001e "TraceServer"="C:\\Program Files\\Cisco Systems\\CTIOS
Client\\CTIOS Toolkit\\Win32 CIL\\Trace\\CTIOSTraceText.exe"
```

For CTIOS server versions 7.5(10), 8.0(3) and later the default trace level will not print the call variable in CTIOS sever logs. This has been done as an enhancement to reduce the log size in these two versions and above. To get the call variable in CTIOS logs you need to set the trace level to 0x400000.

Note You can configure CTI OS Tracing globally for the entire machine (using the TraceMask setting on HKLM) and per user (using the TraceMask setting on HKCU).

Warning If the TraceMask is not set or if it is set incorrectly, the application's performance can be negatively affected. The preferred setting for normal operation is 0x40000307.

Trace Configuration (COM and C++)

You can set C++ and COM client trace configuration parameters in the Windows registry at the following key. For more information about configuring tracing for the Java CIL, see Java CIL Logging Utilities, on page 3. For more information about configuring tracing for the .NET CIL, see Logging and tracing (.NET), on page 6.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\CTIOS Tracing
```

These settings are defined as follows:

Table 1: Configuring Tracing Settings

Parameter	Description	Optimal Value
FlushIntervalSeconds	Maximum number of seconds before the trace mechanism transfers data to the log file.	30
MaxDaysBeforeExpire	Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.	7
MaxFiles	Maximum number of log files that can exist in the log file directory before the logging mechanism starts overwriting old files.	5
MaxFileSizeKb	Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.	2048
TraceMask	Bit mask that determines the categories of events that are traced.	0x40000307

Java CIL Logging Utilities

The Java CIL provides a different logging facility than the C++ CIL. This gives the customer application more flexibility in how trace messages are handled. It also limits the number of special privileges the browser would need to give the applet using the CIL; the Java CIL only needs to access the network and not the file system. For that reason, the Java CIL does its tracing through the firing of special events called “LogEvents” that the custom application can trap and handle in however way it sees fit.

The Java CIL provides the following objects for logging as part of the utilities package.

ILogEvents

This interface must be implemented by a class interested in receiving Java CIL LogEvents. It has only one method.

```
void processLogEvent(LogEvent event)
```

LogEvent

A custom application that is interested in receiving LogEvents receives an object of this type whenever a log message is generated. This class extends the Java “EventObject”, and has one public method.

Method	Description

getDescription	Returns the text description to write somewhere.

Syntax

```
String getDescription()
```

Logger

A custom application that is interested in firing or handling its own LogEvents can create an instance of this class.

Method	Description
Logger	Public constructor of the Logger object.
Trace	Lets the custom app fire a LogEvent.
GetTraceMask	Gets the trace mask.
IsTraceMaskEnabled	Determines if a certain trace mask is set.
addLogListener	Subscribes to receive LogEvents.
removeLogListener	Unsubscribes from receiving LogEvents.

Syntax

```
Logger()
   int Trace(long nMsgTraceMask, String message)
   long GetTraceMask()
   boolean IsTraceMaskEnabled(long nMsgTraceMask)
   void addLogListener(ILogEvents logEvents
```

where logEvents implements the ILogEvents interface.

```
void removeLogListener(ILogEvents logEvents)
```

where logEvents implements the ILogEvents interface.

LogEventsAdapter

This is a wrapper class around the Logger facility. A custom application that is interested in tracing but does not want to implement its own ILogEvents interface can create an instance of this class. The adapter class provides two constructors, a default one that automatically logs to the Java console and one that takes in an output filename.

Method	Description
LogEventsAdapter	Public constructor.
startLogging	Starts receiving LogEvents.
stopLogging	Stops receiving LogEvents.
processLogEvent	Handles a LogEvent.

finalize	Does some cleanup.

Syntax

```
LogEventsAdapter()LogEventsAdapter(String fileName)
void startLogging()
void stopLogging()
void processLogEvent(LogEvent e)
void finalize()
```

Logging and tracing (Java)

The Java CIL tracing mechanism behaves differently from that of the COM and C++ CILs. The Java CIL does not automatically create a log file and trace to it. You must develop the custom application to create and maintain the log file.

The Java CIL provides classes that allow you to write tracing messages from CTI applications. You can create a class that implements ILogListener, register it with the LogManager, and write the trace events to a log file.

The Java CIL also includes the LogWrapper class, which implements the ILogListener interface and provides a default logging mechanism.

The LogWrapper class has three constructors:

- LogWrapper() - Creates a new LogWrapper object that writes tracing messages to System.out.
- LogWrapper(string sFileName) - Creates a new LogWrapper object that writes trace messages to the file specified in sFileName.
- LogWrapper(string sFileName, long iMaxSize, int iArchives, int iExpires, int iFlushIntervalMs) - Creates a new LogWrapper object that traces to the file specified in sFileName and sets all the tracing properties provided:
    - The maximum size of a single trace file (the default is 2048 Kb).
    - The maximum number of trace files before LoggerManager deletes the oldest file (the default is 4).

If a developer deploys an application and then wants to debug it in the field, they need a way to change the trace mask from the default level if necessary to provide more information for debugging.

Note You also need to provide a way to adjust the trace mask at runtime. If you encounter problems, Cisco personnel need to see this log file to assist you with your problem.

For more information about the LogWrapper class and its associated methods, see the Java CIL Javadoc file.

Logging and tracing (.NET)

The .NET CIL tracing mechanism behaves differently from that of the COM and C++ CILs. The .NET CIL does not automatically create a log file and trace to it. You must develop the custom application to create and maintain the log file.

The .NET CIL provides classes that allow you to write tracing messages from CTI applications. Custom applications can either create their own logging mechanism or use the default logging mechanism provided in the .NET CIL.

Default Logging Mechanism

You can use the .NET CIL LogWrapper class to implement logging to the system console or to a file. The LogWrapper class registers itself as an event listener and creates a log file.

Log Trace Events with LogWrapper Class

To log trace events using the LogWrapper class:

Procedure

Step 1 Create an instance of the LogWrapper class, passing the following arguments:

- logFileName - Name of file in which to write trace events.
- fileMaxSize - The maximum size of the log file.
- numberArchivesFiles - Maximum number of log files that can exist in the log file directory before the logging mechanism starts overwriting old files.
- numberDaysBeforeFileExpired - Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.

The following code snippet creates an instance of the LogWrapper class that writes trace events to MyLogFile.txt. When MyLogFile.txt reaches 2048 KB, a new log file is created. The Logger creates a maximum of 20 log files in the log file directory before overwriting existing files. After 10 days, the log file is rolled over into a new log file regardless of its size.

```
// Create a LogWrapper. This will create a file and start // listening for log
events to write to the file.
String logFileName                 = "MyLogFile.txt";
int    fileMaxSize                 = 2048;
int    numberArchivesFiles         = 20;
int    numberDaysBeforeFileExpired = 10;
m_logWrapper = new LogWrapper(logFileName, fileMaxSize, numberArchivesFiles,
numberDaysBeforeFileExpired);
```

Step 2 In your application, write trace events. The following example traces a message at the given trace level for the given method. Set the trace level to the desired trace mask. Trace masks are defined in the Logger class. For more information about available trace mask values, see the following table.

```
protected internal static void MyTrace (int traceLevel,
string methodName,
string msg)
{
if ( m_logger.IsTraceMaskEnabled(traceLevel) )
{
string tracsMsg = string.Format("{0}: {1}", methodName,
msg) ;
m_logger.Trace(traceLevel, msg) ;
}
}
```

The CTI Toolkit Combo Desktop .NET sample application included with the CTI OS toolkit shows how to use the CIL's LogWrapper class in the context of a complex softphone application.

The following table lists the trace masks available in the .NET CIL.

Table 2: Trace Masks in .NET CIL

TraceMask Bit	Value	Purpose
TRACE_LEVEL_MAJOR	0x000000ff	Mask for major events.
TRACE_LEVEL_EVENT_REQ	0x0000ff00	Mask for general events and requests.
TRACE_LEVEL_METHOD	0x00ff0000	Mask for method entry and exit.
TRACE_LEVEL_MEMORY	0xff000000	Mask for very low level operations.
Individual Trace Mask		
Lowest Order Byte Mask: Events		
TRACE_MASK_ALWAYS	0x00	Always print.
TRACE_MASK_CRITICAL	0x01	Critical error.
TRACE_MASK_WARNING	0x02	Warning.
TRACE_MASK_EVT_REQ_HIGH	0x04	High important events/requests.
TRACE_MASK_EVT_REQ_HIGH_PARM	0x08	High important events/requests.
TRACE_MASK_EVT_REQ_AVG	0x10	Average important events/requests.
TRACE_MASK_EVT_REQ_AVG_PARM	0x20	Average important events/requests.
TRACE_MASK_EVT_REQ_LOW	0x40	Low important events/requests.
TRACE_MASK_EVT_REQ_LOW_PARM	0x80	Low important events/requests.
Second Lowest Order Byte: Method Tracing		
TRACE_MASK_METHOD_HIGH	0x0100	High visibility method entry/exit trace.
TRACE_MASK_METHOD_HIGH_LOGIC	0x0200	High visibility method logic trace.

TraceMask Bit	Value	Purpose
TRACE_MASK_METHOD_HIGH_LOGIC	0x0400	Internal visibility method entry/exit trace.
TRACE_MASK_METHOD_AVG_LOGIC	0x0800	Internal visibility method logic trace.
TRACE_MASK_METHOD_LOW	0x1000	Helper object visibility method entry/exit trace.
TRACE_MASK_METHOD_LOW_LOGIC	0x2000	Helper object visibility method logic trace.
TRACE_MASK_METHOD_MAP	0x4000	Map access.
Highest Order Byte: Communications and Processing		
TRACE_MASK_ARGS_METHODS	0x01000000	Method entry/exit for Arguments objects.
TRACE_MASK_ARGS_LOGIC	0x02000000	Logic trace for Arguments objects.
TRACE_MASK_PACKETS_METHODS	0x04000000	Method entry/exit for packets objects.
TRACE_MASK_PACKETS_LOGIC	0x08000000	Logic trace for packets objects.
TRACE_MASK_SERIALIZE_DUMP	0x10000000	Memory dump of serialize buffer.
TRACE_MASK_SOCKETS_DUMP	0x20000000	Memory dump of sockets buffer.
TRACE_MASK_THREADING	0x40000000	Threading tracing on or off.
TRACE_MASK_CONNECTION	0x80000000	Connection tracing on or off.

Custom Logging Mechanism

The LogManager class within the .NET CIL implements all CIL logging functions. This singleton class has only one instance of LogManager, which provides a global point of access. The LogManager object defines a LogEventHandler delegate that custom applications must implement:

```
public delegate void LogEventHandler(object eventSender, LogEventArgs args);
```

Log Trace Events with Logger Class

To log trace events from a custom application to a file, perform the following steps:

Procedure

Step 1 Create a Logger object. For example:

```
m_log = new Logger();
```

Step 2 Write a method to handle log events. This method can trace the log events to a file, if desired. For example:

```
public virtual void ProcessLogEvent(Object eventSender, LogEventArgs Evt){
// Output the trace
String traceLine = Evt.Description;
// Check that tracing is enabled for this tracelevel
if ( m_logger.IsTraceMaskEnabled(traceLevel) )
{
WriteTraceLineToFile(traceLine);
}
}
```

Step 3 Create a log listener to handle trace events. In the following example, the AddLogListener method registers the LogEventHandler delegate as a listener for trace events. The LogManager sends trace events to the method that you pass to the LogEventHandler.

In the following example, the LogManager sends trace events to the ProcessLogEvent method created in Step 2.

```
m_log.AddLogListener(new LogManager.LogEventHandler(ProcessLogEvent));
```

Note The LogManager only calls the method passed as a parameter to the LogEventHandler for a particular trace if the trace level for that trace is enabled. You can use the IsTraceMaskEnabled method in the Logger class to determine whether or not a trace level is enabled.

Trace Configuration (Java and .NET)

For the Java and .NET CILs, you can configure tracing either programmatically by using the LogWrapper class or by editing the TraceConfig.cfg file. Settings in TraceConfig.cfg do not take effect until LogWrapper.ProcessConfigFile is called. Your application must call ProcessConfigFile if you have edited the configuration settings in the TraceConfig.cfg file.

The All Agents Sample .NET code in the .NET CIL includes a sample TraceConfig.cfg file and shows you how to process that file.

Log file configuration settings are defined as follows:

Table 3: Configuration Settings

Parameter	Description	Optimal Value
NumberDaysBeforeFileExpired	Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.	1
NumberArchivesFiles	Maximum number of log files that may exist in the log file directory before the logging mechanism starts overwriting old files.	5
FileMaxSize	Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.	2048
TraceMask	Bit mask that determines the categories of events that are traced.	0x40000307