CM-06 ログ出力機能

■ 概要

本機能は、アプリケーションでログを出力する統一的な仕組みを提供する。個別のロギングパッケージ[1]をラップした標準 API を提供するため、ロギングパッケージを変更する場合の、ソースコードの修正を不要とすることができる。。また、Jakarta Commons Logging[2]と同等のログレベルを備えているため、サーバ側が Java、クライアント側が.NET という組み合わせの開発においても、アプリケーション全体で統一的なログポリシーを定めることができる。

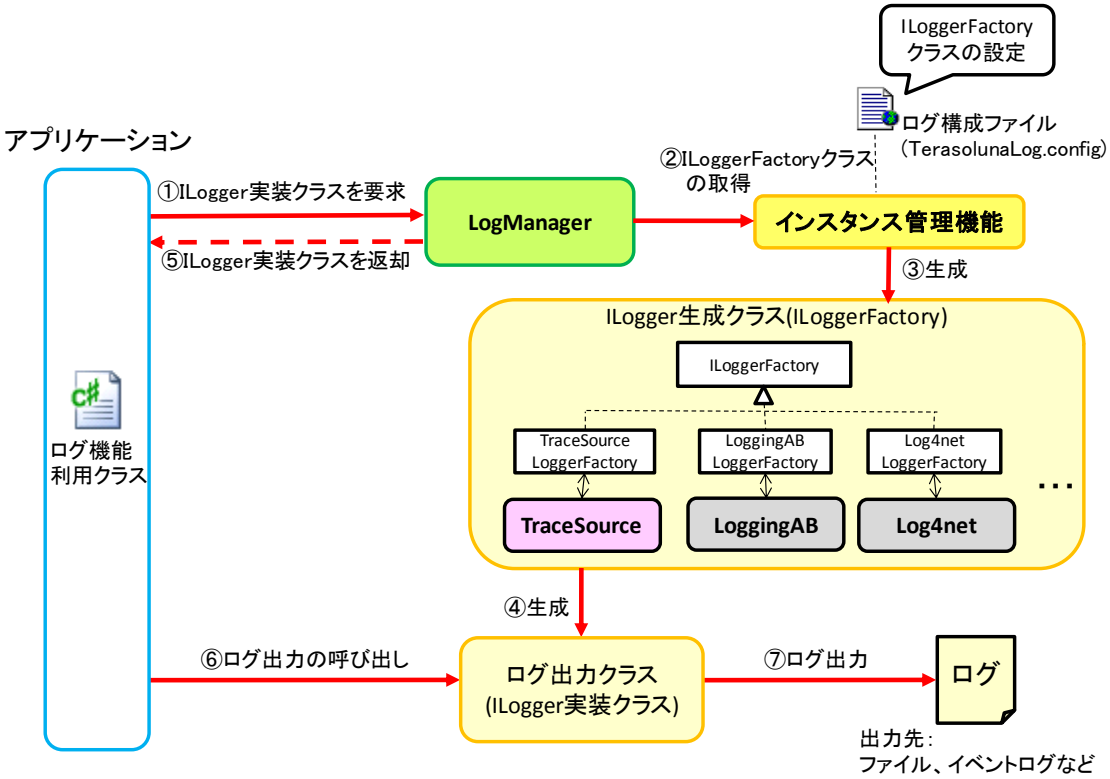


図 1 ログ出力機能の動作概念図

ログ出力機能を利用するクラスは、LogManager クラスより、ログ出力クラス (ILogger 実装クラス)を取得する。LogManager クラスの内部では、「CM-02 インスタンス管理機能」を利用しており、ログ構成ファイル(TerasolunaLog.config)をもとに、ILoggerFactory 実装クラスのインスタンスを生成してログ出力クラスの実装を決定する。

---

[1] Logging AB ：マイクロソフトが推進するオープンソース・ライブラリ「Enterprise Library」に含まれる、ログ出力用の Application Block [http://www.codeplex.com/entlib]
log4net ：Apache プロジェクトでオープンソースとして開発されているロギングライブラリ。「log4j」を.NET に移植したもの [http://logging.apache.org/log4net/index.html]

[2] Jakarta Commons Logging ：Apache プロジェクトでオープンソースとして開発されているロギングパッケージ間の互換性問題解決するためのコンポーネント[http://commons.apache.org/logging/]

本機能は API の標準実装として、.NET の標準クラスである TraceSource を利用したログ出力機能（TraceSourceLogger）を提供する。
標準実装のログ出力クラス（TraceSourceLogger）を使用する場合、TerasolunaLog.config を作成する必要はない。
ログを出力する際は、その優先度や重要度を考慮して「ログレベル」を設定する。本機能では、表 1 に示す 6 つのログレベルを用意している。
ログ出力クラスを取得後、ログ出力用メソッドを呼び出す。メソッドはログレベルごとに用意されている。

表 1 ログレベルの種類

項番	種類	ログレベル	説明
1	致命的なエラー	FATAL	致命的な障害が発生した場合に利用する。
2	エラー	ERROR	予期せぬ動作などにより、正しく処理できない場合に利用する。通常は処理の続行が困難な例外が発生した場合に利用する。
3	警告	WARN	エラーやバグの原因となりそうな場合に利用する。例外をやむを得ず捕捉して、処理を業務フローに戻す場合などに出力する。
4	情報	INFO	システムの動作を通知する際に利用する。起動時や処理の開始時など、重要な動作（外部仕様にかかわるもの）をトレースしたい場合に出力する。
5	デバッグ	DEBUG	起動時・メソッド呼び出し時・インスタンス生成時など、内部の動作をトレースしたい場合（主に開発時）に利用する。
6	トレース	TRACE	モジュール内部の情報、ループの繰り返しで大量に出力される情報など、詳細なデバッグ情報を出力する場合（主に開発時）利用する。

■ 使用方法

◆ TraceSource の設定

TraceSourceLogger を利用する場合は、通常の.NET 開発で TraceSource を使用する場合と同様に、アプリケーション構成ファイル(App.config)や Web 構成ファイル(Web.config)の/configuration/system.diagnostics/タグで、TraceSource の設定をする。
構成ファイルに設定する TraceSource のスイッチレベル(ログレベルの閾値)について、TERASOLUNA フレームワークのログレベルと TraceSource のログレベルに違いがあるため注意が必要である。構成ファイルに設定する TraceSource のスイッチレベルと、有効な TERASOLUNA フレームワークのログレベルの対応を表 2 に示す。
この表では、構成ファイルに設定した TraceSource のスイッチレベルに対して、TERASOLUNA フレームワークのログレベルで出力されるものが「○」になっている。

表 2 TraceSource のスイッチレベルと、TERASOLUA ログレベルの対応表

TERASOLUNA フレームワークのログレベル	構成ファイルに設定するスイッチレベル(SourceLevel 列挙体)						
	Off	Critical	Error	Warning	Information	Verbose	All
FATAL		○	○	○	○	○	○
ERROR			○	○	○	○	○
WARN				○	○	○	○
INFO					○	○	○
DEBUG						○	○
TRACE							○(注1)

(注1)TraceEventType.Verbose でログ出力する。

以下に、表 3 に示す関心の異なるログを出力する場合の構成ファイルの設定例をに示す。なお、カテゴリ名とは、/configuration/system.diagnostics/sources/source/name タグで定義した値のことである。TraceSource の詳細な設定方法は MSDN を参照のこと。

表 3 サンプルで出力するログの種類

項番	ログの種類	カテゴリ名	出力先	ログレベルの閾値
1	運用管理ソフトウェアの監視対象として利用するログ	FatalLog	Windows イベントログ	FATAL (Critical)
2	ユーザによるボタン押下を監視対象としたとした統計／監査ログ	SubmitLog	カンマ区切りのテキストログファイル DelemitedLog.txt	TRACE (All)
3	Terasoluna.Labratories.exe(または dll)に存在するクラスのデフォルトログ	Terasoluna. Laboratories	テキストログファイル Log.txt	INFO (Information)
4			XML 形式のログファイル XMLLog.svclog	WARN (Warning)
5			コンソール	DEBUG (Verbose)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- 運用管理ソフトウェアの監視対象として利用するログ -->
      <source name="FatalLog" switchName="FatalLogSwitch"
        switchType="System.Diagnostics.SourceSwitch">
        <listeners>
          <add name="EventLog"/>
          <remove name="Default"/>
        </listeners>
      </source>
      <!-- 統計用のユーザによるボタン押下を単位としたログ -->
      <source name="SubmitLog" switchName="SubmitLogSwitch"
        switchType="System.Diagnostics.SourceSwitch">
        <listeners>
          <add name="DelimitedLogFile"/>
          <remove name="Default"/>
        </listeners>
      </source>
      <!-- Terasoluna.Labratories.exe（またはdll）に存在するクラスのデフォルトログ -->
      <source name="Terasoluna.Laboratories" switchName="DefaultLogSwitch"
        switchType="System.Diagnostics.SourceSwitch">
        <listeners>
          <!-- レベル（フィルタ）が異なるも出力形式も異なるリスナを追加 -->
          <!-- ログファイル -->
          <add name="LogFile"
               type="System.Diagnostics.TextWriterTraceListener" initializeData="Log.txt"
               traceOutputOptions="DateTime, ProcessId, ThreadId">
            <filter type="System.Diagnostics.EventTypeFilter"   initializeData="Information"/>
          </add>
          <add name="XMLLogFile"/>
          <add name="ConsoleLog"/>
          <remove name="Default"/>
        </listeners>
      </source>
    </sources>
  <!-- TraceSourceに対する閾値（スイッチ）の設定 -->
    <switches>
      <add name="FatalLogSwitch" value="Critical"/>
      <add name="SubmitLogSwitch" value="All"/>
      <add name="DefaultLogSwitch" value="All"/>
    </switches>

・・・
```

```
・・・
    <sharedListeners>
      <!-- イベントログ -->
      <add name="EventLog"
           type="System.Diagnostics.EventLogTraceListener"
           initializeData="sample application" />
      <!-- コンソールログ -->
      <add name="ConsoleLog"
           type="System.Diagnostics.ConsoleTraceListener">
        <filter type="System.Diagnostics.EventTypeFilter"
           initializeData="Verbose"/>
      </add>
      <!-- カンマ区切り形式ログファイル -->
      <add name="DelimitedLogFile"
           type="System.Diagnostics.DelimitedListTraceListener"
           delimiter =","
           initializeData="DelemitedLog.txt"
           traceOutputOptions="DateTime, ProcessId, ThreadId"/>
      <!-- XML形式のログファイル(Warningレベル) -->
      <add name="XMLLogFile"
           type="System.Diagnostics.XmlWriterTraceListener"
           initializeData="XMLLog.svclog"
           traceOutputOptions="DateTime, ProcessId, ThreadId">
        <filter type="System.Diagnostics.EventTypeFilter"
           initializeData="Warning"/>
      </add>
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

リスト 1 構成ファイルの記述例

◆ 実装方法

- ログ出力クラスの取得方法
  ログ出力クラスを取得するには LogManager クラスの GetLogger メソッドを呼び出す。
  GetLogger メソッドの一覧を表 4 に示す。

表 4　LogManager クラスの GetLogger メソッド一覧

項番	メソッドの種類	第 1 引数	第 2 引数	説明
1	public static ILogger GetLogger()	-	-	呼び出し元クラスに関連するログ出力クラスを返却する。デフォルト実装（TraceSourceFactory）では、カテゴリ名として、呼び出し元クラスの所属するアセンブリ名が利用される。
2	public static ILogger GetLogger(Type type, params object[] args)	ログ出力対象クラスの Type	ログ出力クラスの生成に必要なパラメータ	指定したクラスに関連するログ出力クラスを返却する。デフォルト実装（TraceSourceFactory）では、カテゴリ名として、呼び出し元クラスの所属するアセンブリ名が利用される
3	public static ILogger GetLogger(string name, params object[] args)	カテゴリ名	ログ出力クラスの生成に必要なパラメータ	指定したカテゴリ名に対応するログ出力クラスを返却する

LogManager によるログ出力クラス取得の実装例をリスト 2 に示す。
この例では、呼び出し元クラス TergetClass は、Terasoluna.Laboratories.exe に存在する。

```
public class TargetClass
{
    /// 運用管理ソフトウェアの監視対象として利用するログ（カテゴリ名）
    private ILogger _fatalLog = LogManager.GetLogger("FatalLog");

    /// 統計用のユーザによるボタン押下を単位としたログ
    private ILogger _submitLog = LogManager.GetLogger("SubmitLog");

    /// カテゴリ名は対象クラスのアセンブリ名になる。
    /// 例えば、このクラスはTerasoluna.Laboratories.exeに存在するので
    /// カテゴリ名は「Terasoluna.Laboratories」になる。
    private ILogger _defaultLog = LogManager.GetLogger();

    public void Execute()
    {
        // 処理
    }
}
```

リスト 2　ログ出力クラスを取得する実装例

- ログの出力
  ログを出力するには、ログ出力クラスのログ出力用メソッドを呼び出す。メソッドはログレベルごとに用意されている。ログレベルと対応するメソッドの一覧を表 5 に示す。
  引数が 1 つのメソッドでは、”message”に指定された引数をログメッセージとして出力する。
  引数が 2 つのメソッドでは、”message”に加えて”ex”に指定された例外を共に出力する。
  出力ログレベル確認用のプロパティは、そのログレベルが現在有効かどうかをチェックし、有効なら true、無効なら false を返す。このプロパティをチェックすることで、そのログレベルが有効でない場合に、メソッドの引数に与える文字列オブジェクトを作成するためのオーバーヘッドを減らすことができる。

表 5 ログレベルと対応するメソッド

項番	種類	ログレベル	ログ出力用メソッド	出力ログレベル確認用プロパティ
1	致命的なエラー	FATAL	Fatal(object message) Fatal(object message, Exception ex)	IsFatalEnabled
2	エラー	ERROR	Error(object message) Error(object message, Exception ex)	IsErrorEnabled
3	警告	WARN	Warn(object message) Warn(object message, Exception ex)	IsWarnEnabled
4	情報	INFO	Info(object message) Info(object message, Exception ex)	IsInfoEnabled
5	デバッグ	DEBUG	Debug(object message) Debug(object message, Exception ex)	IsDebugEnabled
6	トレース	TRACE	Trace(object message) Trace(object message, Exception ex)	IsTraceEnabled

  なお、TraceSourceLogger の場合、ログ出力メソッドを呼び出す度に即時にフラッシュ（TraceSource.Flush）をし、ログへの出力処理を確定している。

- イベント ID の出力
  TraceSource ではイベント ID をログに出力可能である。本機能で、TraceSourceLogger を利用する場合、デフォルトでイベント ID は「0」で出力されるようになっている。
  ILogger の API は、ロギングパッケージの実装に依存しないように作られているため、TERASOLUNA フレームワークでは、TraceSourceLogger に固有な処理である、イベント ID 設定機能を、同クラスの staic メソッド（TraceSourceLogger.SetEventId(int eventId)メソッド）として、提供している。
  TrasceSourceLogger クラスは出力するイベント ID を ThreadStatic（スレッドごとのデータ保管場所）で管理している。このためマルチスレッド処理により誤ってイベント ID が上書きされることはない。しかし、1スレッド内の一連の処理の中で複数種類のカテゴリのログ出力クラスから複数回ログ出力メソッドが呼び出されることがあるため、出力メソッド呼び出し直後に毎回イベント ID をクリアする必要がある。このため、SetEventId メソッドは、ログメソッド出力メソッドの直前で呼び出す必要があるので注意すること。

- 実装例

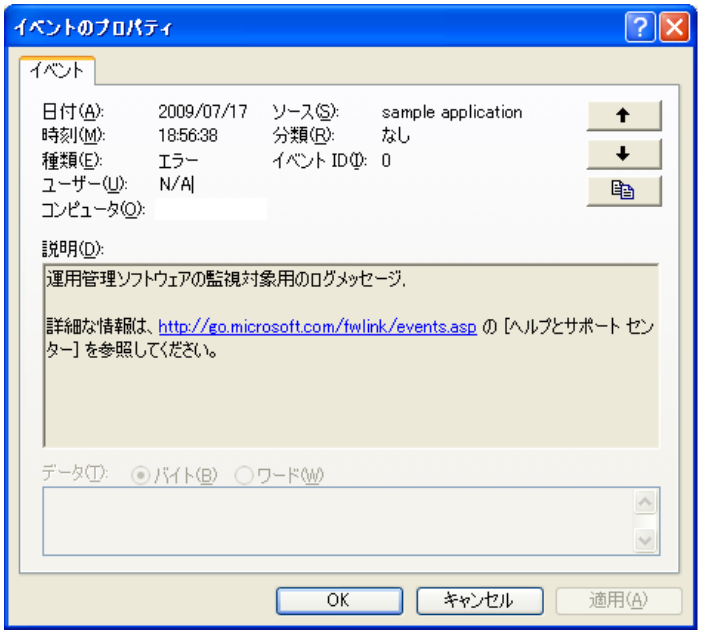
  実装例として、表 3 に示したログを出力するためのコードをリスト 3 に示す。なお、カテゴリの定義、カテゴリごとのログレベルや出力先の定義はリスト 1 のアプリケーション構成ファイルの通りに設定されているものとする。

```
public class TargetClass
{
    /// 運用管理ソフトウェアの監視対象として利用するログ
    private ILogger _fatalLog = LogManager.GetLogger("FatalLog");
    /// 統計用のユーザによるボタン押下を単位としたログ
    private ILogger _submitLog = LogManager.GetLogger("SubmitLog");
    /// 例えば、このクラスはTerasoluna.Laboratories.exeに存在するので
    /// カテゴリ名は「Terasoluna.Laboratories」になる。
    private ILogger _defaultLog = LogManager.GetLogger();
    /// ログの出力実行例
    public void Execute()
    {
        if (_fatalLog.IsFatalEnabled)
        {
            _fatalLog.Fatal("運用管理ソフトウェアの監視対象用のログメッセージ");
        }
        if (_submitLog.IsTraceEnabled)
        {
            _submitLog.Trace("統計用のログメッセージ");
        }
        if (_defaultLog.IsInfoEnabled)
        {
            _defaultLog.Debug("デバッグ");
        }
        if (_defaultLog.IsInfoEnabled)
        {
            _defaultLog.Info("情報");
        }
        if (_defaultLog.IsWarnEnabled)
        {
            ///TraceSourceLoggerのイベントIDの設定例
            ///出力メソッド（この場合Warn）の呼び出し後に
            ///イベントIDはクリアされるので出力する直前に設定すること
            TraceSourceLogger.SetEventId(100);
            _defaultLog.Warn("警告");
        }
         try
            {
                throw new Exception();
            }
            catch (Exception ex)
            {
                if (_defaultLog.IsErrorEnabled)
                {
                    _defaultLog.Error("エラー", ex);
                }
            }
        }
    }
```

リスト 3　ログ出力の実装例

- 出力結果
  リスト 10 を実行した際のログの出力結果を以下に示す。

  ✧ Windows イベント(EventLogTraceListener による出力)
  イベントビューアに、運用監視用のログ(FatalLog カテゴリ)が FATAL レベル(Critical)で出力されている。



リスト 4 EventLogTraceListener による出力例

✧ DelimitedLog.txt(DelimitedListTraceListener による出力)
統計用のログ(SubmitLog カテゴリ)が Trace レベル(Verbose)以上で出力されている。

```
"SubmitLog",Verbose,0,,"統計用のログメッセージ",4376,,"1","2009-07-17T11:38:27.3976932Z",,
```

リスト 5 DelimitedListTraceListener による出力例

✧ Log.txt(TextWriterTraceListener による出力)
対象クラスのデフォルトログ(Terasoluna.Laboratories カテゴリ)が、INFO レベル(Information)以上で出力されている。
また警告ログの EventID が 100 で設定されているのが分かる。

```
Terasoluna.Laboratories Information: 0 : 情報,
      ProcessId=5564
      ThreadId=1
      DateTime=2009-07-17T11:35:05.1260320Z
Terasoluna.Laboratories Warning: 100 : 警告,
      ProcessId=5564
      ThreadId=1
      DateTime=2009-07-17T11:35:05.1260320Z
Terasoluna.Laboratories Error: 0 : エラー, System.Exception: 種類 'System.Exception' の例外がスローされました。
    場所 Terasoluna.Laboratories.Logging.TargetClass.Execute()
    場所 D:¥teraDotNet¥src¥Terasoluna3.5¥Terasoluna.Laboratories¥Logging¥TargetClass.cs:行 52
      ProcessId=5564
      ThreadId=1
      DateTime=2009-07-17T11:35:05.1260320Z
```

リスト 6　TextWriterTraceListener による出力例

✧ XMLLog.svclog（XMLWriterTraceListener による出力）
対象クラスのデフォルトログ（Terasoluna.Laboratories カテゴリ）が、WARN レベル（Warning）以上で出力され、警告ログの EventID が 100 で設定されているのが分かる。
XMLWriterTraceListener が出力する XML は通常のテキストエディタでは見づらいため、サービストレースビューア(SvcTraceViewer.exe)で表示した例を示す。

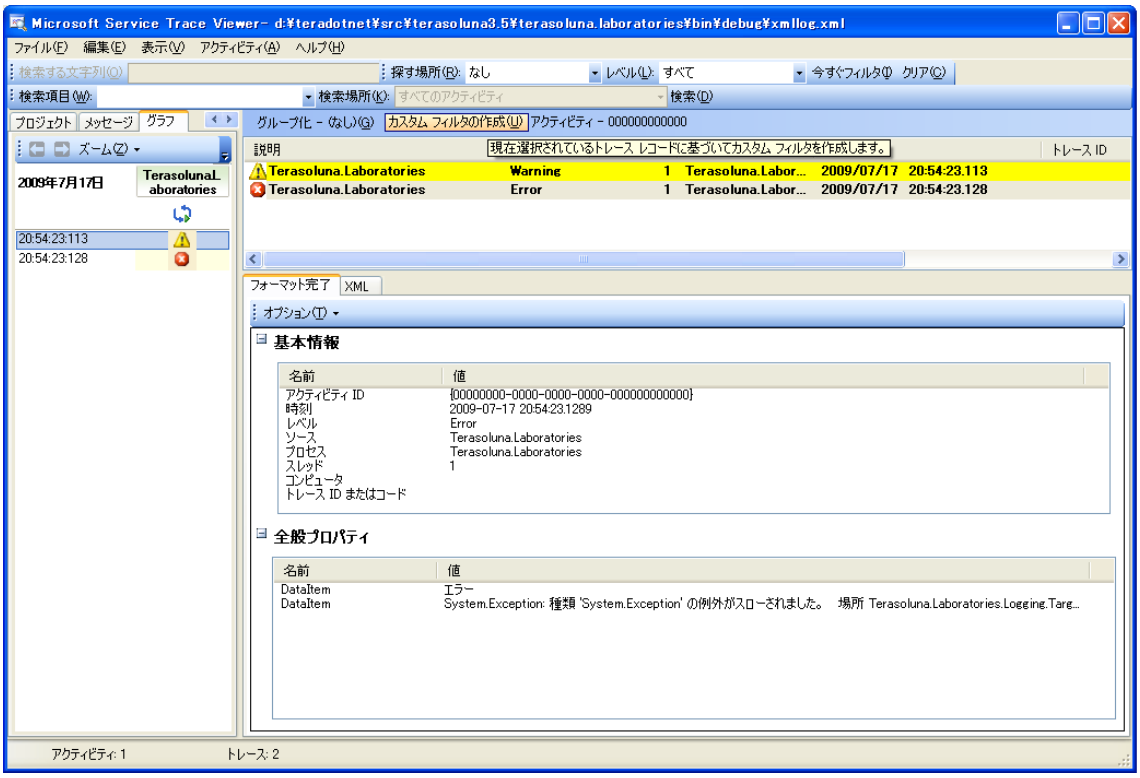


図 2　サービストレースビューア

```
<E2ETraceEvent xmlns="http://schemas.microsoft.com/2004/06/E2ETraceEvent">
        <System xmlns="http://schemas.microsoft.com/2004/06/windows/eventlog/system">
            <EventID>100</EventID>
            <Type>3</Type>
            <SubType Name="Warning">0</SubType>
            <Level>4</Level>
            <TimeCreated SystemTime="2009-07-17T11:54:23.1133288Z" />
            <Source Name="Terasoluna.Laboratories" />
            <Correlation ActivityID="{00000000-0000-0000-0000-000000000000}" />
            <Execution ProcessName="Terasoluna.Laboratories" ProcessID="2264" ThreadID="1" />
            <Channel />
            <Computer>XXXXXX</Computer>
        </System>
        <ApplicationData>
            <TraceData>
                <DataItem>警告</DataItem>
                <DataItem></DataItem>
            </TraceData>
        </ApplicationData>
</E2ETraceEvent>
<E2ETraceEvent xmlns="http://schemas.microsoft.com/2004/06/E2ETraceEvent">
        <System xmlns="http://schemas.microsoft.com/2004/06/windows/eventlog/system">
            <EventID>0</EventID>
            <Type>3</Type>
            <SubType Name="Error">0</SubType>
            <Level>2</Level>
            <TimeCreated SystemTime="2009-07-17T11:54:23.1289544Z" />
            <Source Name="Terasoluna.Laboratories" />
            <Correlation ActivityID="{00000000-0000-0000-0000-000000000000}" />
            <Execution ProcessName="Terasoluna.Laboratories" ProcessID="2264" ThreadID="1" />
            <Channel />
            <Computer>XXXXX</Computer>
        </System>
        <ApplicationData>
            <TraceData>
                <DataItem>エラー</DataItem>
                <DataItem>System.Exception: 種類 'System.Exception' の例外がスローされました。
                    場所 Terasoluna.Laboratories.Logging.TargetClass.Execute() 場所
                D:¥teraDotNet¥src¥Terasoluna3.5¥Terasoluna.Laboratories¥Logging¥TargetClass.cs:行
                54</DataItem>
            </TraceData>
        </ApplicationData>
</E2ETraceEvent>
```

リスト 7　XMLWriterTraceListener による出力例

✧ コンソール（ConsoleTraceListener による出力）
対象クラスのデフォルトログ（Terasoluna.Laboratories カテゴリ）が、DEBUG レベル(Verbose)以上で出力されている。また警告ログの EventID が 100 で設定されているのが分かる。

```
Terasoluna.Laboratories Verbose: 0 : デバッグ,
Terasoluna.Laboratories Information: 0 : 情報,
Terasoluna.Laboratories Warning: 100 : 警告,
Terasoluna.Laboratories Error: 0 : エラー, System.Exception: 種類 'System.Exception' の例外がスローされました。
   場所 Terasoluna.Laboratories.Logging.TargetClass.Execute()
   場所 D:¥teraDotNet¥src¥Terasoluna3.5¥Terasoluna.Laboratories¥Logging¥TargetClass.cs:行 54
```

リスト 8　ConsoleTraceListener による出力例

■ TERASOLUNA フレームワークが出力するログの設定

TERASOLUNA フレームワークの各機能は、本機能を利用し ILogger の GetLogger()メソッドでデフォルトログ（カテゴリ名がアセンブリ名のログ）を出力している。
フレームワーク自身が出力するログを出力させたい場合は、カテゴリ名にフレームワークのアセンブリ名（「Terasoluna」や「Terasoluna.Windows.Forms」など）を指定する。
以下に、アプリケーション構成ファイル(App.config)の設定例を示す。
この例では、ログ出力対象のフレームワークのアセンブリが、Terasoluna.Windows.Forms.dll であるため、カテゴリ名を「Terasoluna.Windows.Forms」で指定している。

```
        <!--フレームワーク（Terasoluna.Windows.Forms.dll）のデフォルトログ -->
        <source name="Terasoluna.Windows.Forms" switchName="FrameworkLogSwitch"
switchType="System.Diagnostics.SourceSwitch">
          <listeners>
            <add name="XMLLogFile" />
            <add name="ConsoleLog" />
            <remove name="Default" />
          </listeners>
        </source>
      <!--エントリポイントがあるアセンブリ(TourtSampleWinFormApp.exe)のデフォルトログ -->
        <source name="TourSampleWinFormsApp" switchName="DefaultLogSwitch"
switchType="System.Diagnostics.SourceSwitch">
          <listeners>
            <add name="XMLLogFile" />
            <add name="ConsoleLog" />
            <remove name="Default" />
          </listeners>
        </source>
        <!-- 業務個別のアセンブリ(TourSampleA01.dll)のデフォルトログ -->
        <source name="TourSampleA01" switchName="DefaultLogSwitch"
switchType="System.Diagnostics.SourceSwitch">
          <listeners>
            <add name="XMLLogFile" />
            <add name="ConsoleLog" />
            <remove name="Default" />
          </listeners>
        </source>
      ・・・
      </sources>
        <!-- ログスイッチの設定 -->
      <switches>
        <add name="FrameworkLogSwitch" value="Error" />
        <add name="DefaultLogSwitch" value="All" />
      </switches>
      ・・・
    </system.diagnostics>
</configuration>
```

リスト 9　アプリケーション構成ファイル（App.config）のログの設定例

■ 内部構成

◆ 構成クラス

本機能を構成するクラスを以下に示す。

表 6 構成クラス一覧

項番	クラス名	説明
Terasoluna.Logging 名前空間		
1	LogManager	ログ出力機能の管理クラス。インスタンス管理機能を利用して生成した ILoggerFactory 実装クラスを利用して ILogger 実装クラスを生成する。
2	ILogger	ログ出力クラスに関わらず、統一的な方法でログを出力するためのインタフェース。
3	LoggerBase	ILogger インタフェースを実装する基底クラス。ログ出力に関する共通的な機能を実装している。通常、ログ出力クラスを作成する場合はこのクラスを継承し、abstract メソッドを実装する。
4	ILoggerFactory	ILogger 実装クラスを生成するインタフェース。
5	LoggerFactoryBase	ILoggerFactory インタフェースを実装する基底クラス。 ILogger 実装クラス生成に関する共通的な機能を実装している。通常、ログ出力クラスを作成する場合はこのクラスを継承し、abstract メソッドを実装する。
6	TraceSourceLogger	ILogger のデフォルト実装クラス。TraceSource を使ってログを出力する。
7	TraceSourceLogFactory	TraceSourceLogger を生成するファクトリクラス。 TERASOLUNA フレームワークのデフォルトの ILoggerFactory 実装クラスである。
8	LogLevel	ログの出力レベルを定義する Enum。

■ 拡張ポイント

TERASOLUNA フレームワークでは、デフォルト実装として TraceSource を使ったログ出力機能を提供しているため、LoggingAB や Log4net などのロギングパッケージに変更する場合には、以下のクラスを作成する必要がある。

- ILogger インタフェース実装クラス

  ILogger に定義されているログ出力用メソッドと、出力ログレベル確認用プロパティを実装する。また、TraceSource のイベント ID(TraceSourceLogger.SetEventId メソッド)のように、個別のロギングパッケージの機能を実現するためのメソッドやプロパティを必要に応じて追加する。通常は、ILogger の基底クラスである LoggerBase クラスを継承し abstract メソッドを実装するとよい。

- ILoggerFactory インタフェース実装クラス

  ILoggger 実装クラスのインスタンスを取得するための抽象メソッドを実装する。

  通常は、ILoggerFactory の基底クラスである LoggerFactoryBase クラスを継承し abstract メソッドを実装するとよい。

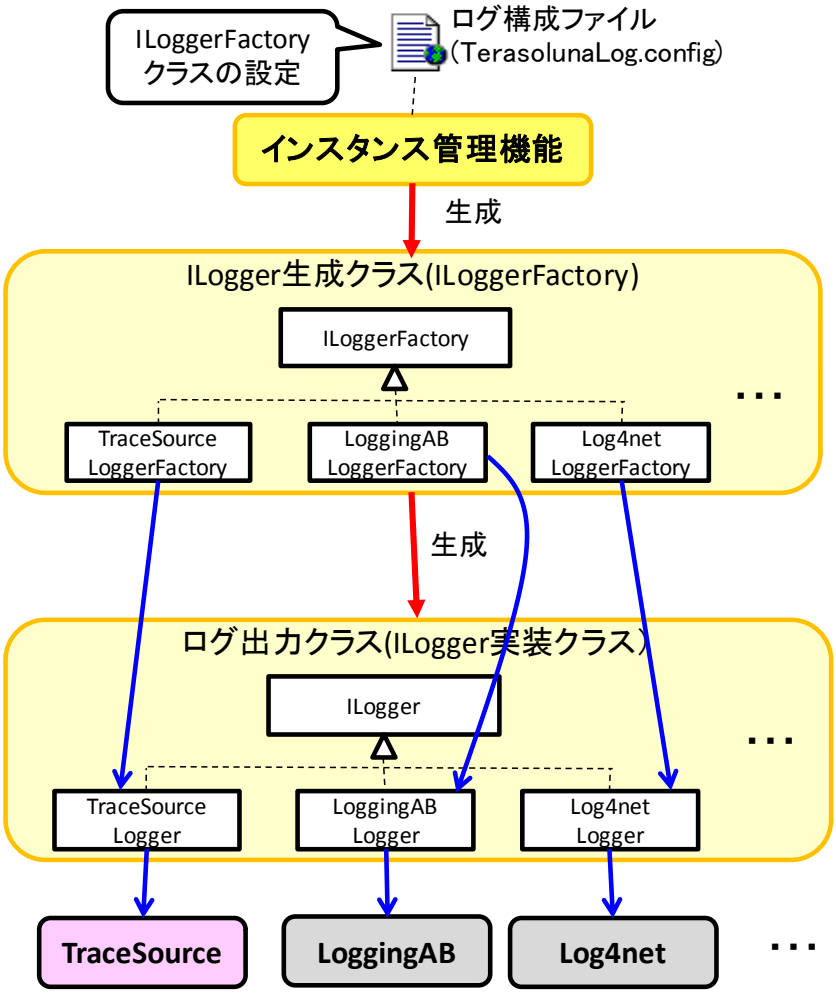


図 3 本機能が提供するインタフェースとロギングパッケージの関係

このとき、個別のロギングパッケージを、本機能のログレベルと対応させて実装する。表 7 にロギングパッケージと TERASOLUNA のログレベルの対応関係を示す。

表 7 TERASOLUNA と各ロギングパッケージのログレベルの対応

項番	TERASOLUNA のログレベル	ロギングパッケージのログレベル		
		TraceSource (TraceEventType 列挙体)	Logging AB	Log4net
1	FATAL	Critical	Critical	FATAL
2	ERROR	Error	Error	ERROR
3	WARN	Warning	Warning	WARN
4	INFO	Information	Information	INFO
5	DEBUG	Verbose	Verbose	DEBUG
6	TRACE	Verbose（※）	（※）	（※）

（※）ロギングパッケージには TRACE にあたるログレベルが定義されていないため、個別の Logger で定義する必要がある。標準実装の TraceSourceLogger における TRACE レベルでは、Verbose を出力している。

また、本機能を拡張して標準のログ出力機能以外の実装を利用する場合には、ログ構成ファイル(TerasolunaLog.config)を作成する必要がある。
ログ構成ファイルは、TERASOLUNA フレームワークが提供するカスタムアイテムテンプレートを使用して作成する。

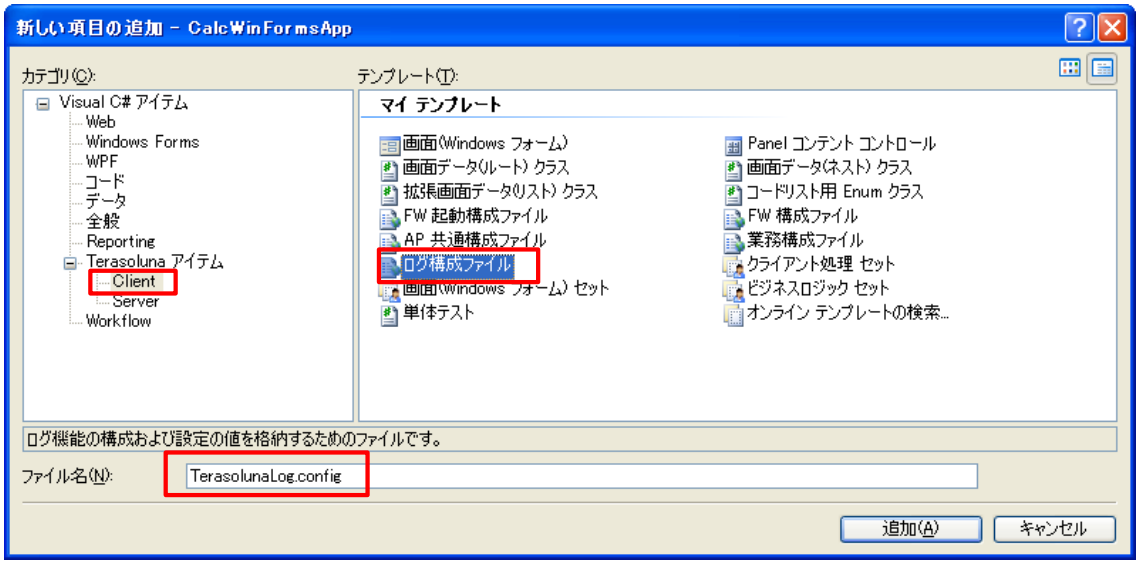


図 4 「ログ構成ファイル」カスタムアイテムテンプレート

ILoggerFactory クラスは「CM-02 インスタンス管理機能」を利用しインスタンスを生成するため、
Unity AB の設定により ILoggerFactory 実装クラスを指定する。
以下に、TerasolunaLog.config の記述例を示す。

```
<configuration>
・・・
  <unity>
    <typeAliases>
      <!-- typeAliasの設定 -->
      <typeAlias alias="ILoggerFactory"
                 type="Terasoluna.Logging.ILoggerFactory, Terasoluna" />
      <typeAlias alias="Log4jLoggerFactory"
                 type="Terasoluna.Sample.Logging.Log4jLoggerFactory, Terasoluna.Sample" />
    </typeAliases>
    <containers>
      <container>
        <types>
          <!-- ILoggerFactory実装クラスの設定 -->
          <type name="LoggerFactory" type="ILoggerFactory" mapTo="Log4jLoggerFactory"/>
        </types>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 10 TerasolunaLog.config の設定例

■ 関連機能

- CM-02 インスタンス管理機能