Dignostic.dll pielietošana Windows lietojumos

Saturs

[Ievads 3](#_Toc299456254)

[Dokumenta nolūks 3](#_Toc299456255)

[Saistītie dokumenti 3](#_Toc299456256)

[Vispārējie norādījumi 3](#_Toc299456257)

[1. LogUtility 4](#_Toc299456258)

[2. TraceUtility 7](#_Toc299456259)

[3. ExceptionUtility 9](#_Toc299456260)

[4. LogActivity 11](#_Toc299456261)

[5. ExtraInformationProvider 14](#_Toc299456262)

[6. Enterprise Library 4.0.0.0 16](#_Toc299456263)

# Ievads

## Dokumenta nolūks

Šī ir instrukcija kā izmantot Diagnostic.dll un Enterprise Library bibliotēkas, lai nodrošinātu sekojošas funkcijas:

* Žurnalēšana (Logging) izmantojot LogUtility;
* Trasēšana (Tracing) izmantojot TraceUtility;
* Izņēmumu žurnalēšana izmantojot ExceptionUtility;
* Aktivitāšu trasēšana izmatojot LogActivity;

## Saistītie dokumenti

[1] –  Koplietojuma bibliotēku apraksts, VSMTVA-132-E\_VES\_SP\_KBA;

[2] – Instrukciju pamatā tiek izmantots risinājums EnoughPI.sln, kas atrodas mapē „*~\Diagnostic\CS\WindowsApp\begin*”;

[3] – Enterprise Library konfigurācijas faili - <http://blogs.msdn.com/b/tomholl/archive/2006/04/02/entlib2externalconfig.aspx>

Lietojums EnoughPI aprēķina konstanti pi līdz uzdotajai precizitātei. Lietotājs ievada vēlamo precizitāti lietojot NumericUpDown vadīklu un nospiež Calculate pogu. Aprēķinu var apturēt nospiežot pogu Cancel.

## Vispārējie norādījumi

Pirms tālāku darbību veikšanas nepieciešams izpildīt sekojošus priekšnosacījumus:

1. Izveidot jaunu risinājuma EnoughPI.sln kopiju, kurā tiks veiktas visas turpmākās darbības; [[2](#solution)]
2. Atvērt risinājumu EnoughPI.sln;
3. Atvērt projektu EnoughPI un iezīmēt katalogu References, izpildīt konteksta izvēlnes komandu Add Reference un pievienot projektam Diagnostic.dll bibliotēku. (tipiski atrodama katalogā *„~\Diagnostic\Lib”*).

# LogUtility

Šajā piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu esošam projektam žurnalēšanas funkcionalitāti izmantojot LogUtility klasi. Izpildītais risinājuma piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex01Log”*.

1. Iezīmēt projekta failu Calc\Calculator.cs un izpildīt konteksta izvēlnes komandu View Code;
2. Pievienot vārdtelpu Diagnostic faila augšdaļā;

using Diagnostic;

1. Modificēt konfigurācijas datni app.config, pievienojot sekojošus atribūtus:

* Konfigurācijas sekciju diagnosticConfiguration;

<configSections>

<section name="diagnosticConfiguration"

type="Diagnostic.Configuration.DiagnosticSettings,

Diagnostic, Version=1.0.0.0"/>

</configSections>

* Sadaļu system.diagnostics kas satur sadaļas sharedListeners (pēc izvēles) un sources. Sadaļā sharedListeners definēt koplietojamos žurnalēšanas failus, norādot sekojošus atribūtus: atrašanās vieta – initializeData, šajā gadījumā lietojuma katalogā, tips – type, identifikators – name, papildus žurnalēšanas iespējas – traceOutputOptions un citi atribūti pēc nepieciešamības;

<system.diagnostics>

<sharedListeners>

<add initializeData="Progress.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="ProgressLog"

traceOutputOptions="Timestamp"/>

<add initializeData="General.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="GeneralLog"

traceOutputOptions="Timestamp"/>

</sharedListeners>

* Sadaļā sources definēt ziņojumu klases source ar atbilstošiem identifikatoriem name. Katrai klasei piesaistīt vismaz vienu žurnalēšanas failu listeners, norādot kādu no koplietojamajiem failiem pēc tā identifikatora name, vai definējot jaunu žurnalēšanas failu;

<sources>

<source name="General" switchValue="All">

<listeners>

<add name="GeneralLog"/>

</listeners>

</source>

<source name="Progress" switchValue="All">

<listeners>

<add name="ProgressLog" />

</listeners>

</source>

</sources>

* Sadaļā system.diagnostics definēt atribūtu trace ar īpašību autoflush un vērtību true. Kas norāda ka rakstīšana žurnalēšanas failā tiks veikta uzreiz.

<trace autoflush="true" />

</system.diagnostics>

1. Failā Calc\Calculator.cs izveidot LogUtility klases eksemplāru, klases Calculator ietvaros, norādot atribūtu sourceName, kas tiks lietots ziņojumu avota identificēšanai, šajā gadījumā projekta nosaukums *EnoughPI*;

LogUtility logwriter = new LogUtility("EnoughPI");

1. Nepieciešamajās vietās pievienot projekta kodam rakstīšanas metodes Write. Kategorijā norādot žurnalēšanas klases identifikatoru; [[1]](#KBA)

// TODO: Log final result

string message = string.Format("Rezultāts: Pi = {0}, precizitāte = {1}", args.Pi, args.Digits);

logwriter.Write(message, Category.General, Priority.Normal, 1, System.Diagnostics.TraceEventType.Information);

// TODO: Log exception

logwriter.Write("Izņēmums", Category.General, Priority.High, 4, System.Diagnostics.TraceEventType.Error, args.Exception);

1. Nepieciešamības gadījumā modificēt risinājumā iekļautā projekta EnoughPI.Logging ziņojumu kategoriju Category un prioritāšu Priority struktūras, kas definētas Constants.cs failā;

public struct Priority

{

public const int Lowest = 0;

public const int Low = 1;

public const int Normal = 2;

public const int High = 3;

public const int Highest = 4;

}

public struct Category

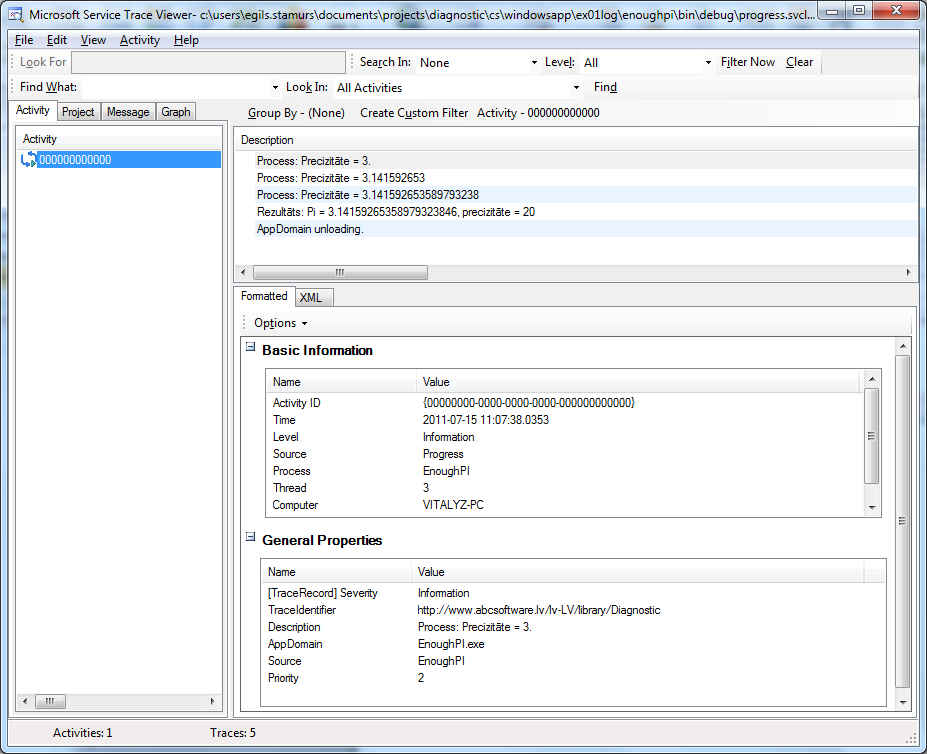
{

public const string General = "General";

public const string Progress = "Progress";

}

1. Iegūtais rezultāts apkopojot General un Progress žurnālus (Ilustrācija 1‑1).



Ilustrācija 1‑1 Žurnalēšanas rezultāti

# TraceUtility

Šajā piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu esošam projektam trasēšanas funkcionalitāti izmantojot TraceUtility klasi. Izpildītais risinājuma piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex02Trace”*.

1. Iezīmēt projekta failu Calc\Calculator.cs un izpildīt konteksta izvēlnes komandu View Code;
2. Pievienot vārdtelpu Diagnostic faila augšdaļā;

using Diagnostic;

1. Modificēt konfigurācijas datni app.config, pievienojot sekojošus atribūtus:

* Konfigurācijas sekciju diagnosticConfiguration;

<configSections>

<section name="diagnosticConfiguration"

type="Diagnostic.Configuration.DiagnosticSettings,

Diagnostic, Version=1.0.0.0"/>

</configSections>

* Sadaļu system.diagnostics kas satur sadaļas sharedListeners (pēc izvēles) un sources. Sadaļā sources definēt ziņojumu klases source ar atbilstošiem identifikatoriem name. Katrai klasei piesaistīt vismaz vienu listeners failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora name, vai definējot jaunu failu norādot sekojošus atribūtus: atrašanās vieta – initializeData, tips – type, identifikators – name, papildus iespējas – traceOutputOptions un citi atribūti pēc nepieciešamības;

<system.diagnostics>

<sources>

<source name="Trace" switchValue="All">

<listeners>

<add initializeData="Trace.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="Tracer"

traceOutputOptions="Timestamp"/>

</listeners>

</source>

</sources>

* Sadaļā system.diagnostics definēt atribūtu trace ar īpašību autoflush un vērtību true. Kas norāda ka rakstīšana failā tiks veikta uzreiz.

<trace autoflush="true" />

</system.diagnostics>

1. Projekta kodā izvietot trasēšanas metodes StartTrace, iekļaujot trasējamo koda fragmentu using operatora figūriekavās, un kā parametru norādot ziņojumu klases source identifikatoru name; [[1]](#KBA)

protected void OnCalculating(CalculatingEventArgs args)

{

using (TraceUtility.StartTrace(Category.Trace))

{

if (Calculating != null)

Calculating(this, args);

}

}

1. Nepieciešamības gadījumā modificēt risinājumā iekļautā projekta EnoughPI.Logging ziņojumu kategoriju Category un prioritāšu Priority struktūras, kas definētas Constants.cs failā;

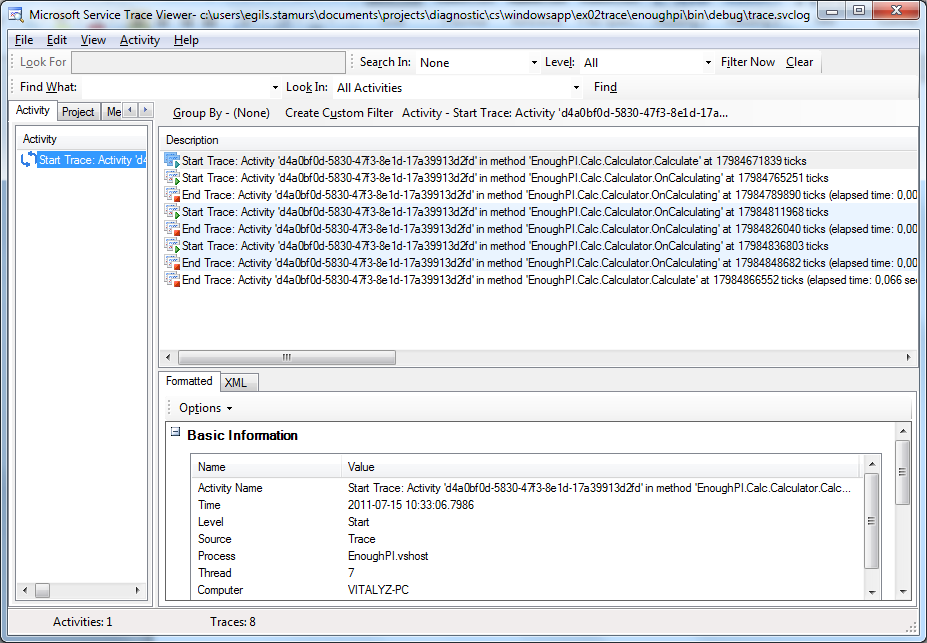
public struct Category

{

public const string Trace = "Trace";

}

1. Iegūtais rezultāts trasējot metodes Calculate un OnCalculating (Ilustrācija 2‑1).



Ilustrācija 2‑1 Trasēšanas rezultāti

# ExceptionUtility

Piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu esošam projektam izņēmumu žurnalēšanas funkcionalitāti izmantojot ExceptionUtility klasi. Realizētais piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex03Exception”*.

1. Iezīmēt projekta failu Calc\Calculator.cs un izpildīt konteksta izvēlnes komandu View Code;
2. Pievienot vārdtelpu Diagnostic faila augšdaļā;

using Diagnostic;

1. Modificēt konfigurācijas datni app.config, pievienojot sekojošus atribūtus:

* Konfigurācijas sekciju diagnosticConfiguration;

<configSections>

<section name="diagnosticConfiguration"

type="Diagnostic.Configuration.DiagnosticSettings,

Diagnostic, Version=1.0.0.0"/>

</configSections>

* Sadaļu system.diagnostics kas satur sadaļas sharedListeners(pēc izvēles) un sources. Sadaļā sources definēt ziņojumu klasi source ar identifikatoru name = *„General”*. Klasei piesaistīt vismaz vienu listeners failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora name, vai definējot jaunu failu norādot sekojošus atribūtus: atrašanās vieta – initializeData, tips – type, identifikators – name, papildus iespējas – traceOutputOptions un citi atribūti pēc nepieciešamības;

<system.diagnostics>

<sources>

<source name="General" switchValue="All">

<listeners>

<add initializeData="Exception.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="Exception"

traceOutputOptions="Timestamp"/>

</listeners>

</source>

</sources>

* Sadaļā system.diagnostics definēt atribūtu trace ar īpašību autoflush un vērtību true. Kas norāda ka rakstīšana failā tiks veikta uzreiz.

<trace autoflush="true" />

</system.diagnostics>

1. Failā Calc\Calculator.cs izveidot ExceptionUtility klases eksemplāru klases Calculator ietvaros, kā parametru norādot LogUtility klases eksemplāru ar parametru sourceName, kas tiks lietots ziņojumu avota identificēšanai;

ExceptionUtility exceptionWriter = new ExceptionUtility(new LogUtility("EnoughPI"));

1. Projekta kodā pēc nepieciešamības izvietot izņēmumu rakstīšanas metodes, piemēram ThrowHelperError vai citas; [[1]](#KBA)

protected void OnCalculatorException(CalculatorExceptionEventArgs args)

{

// TODO: Log exception

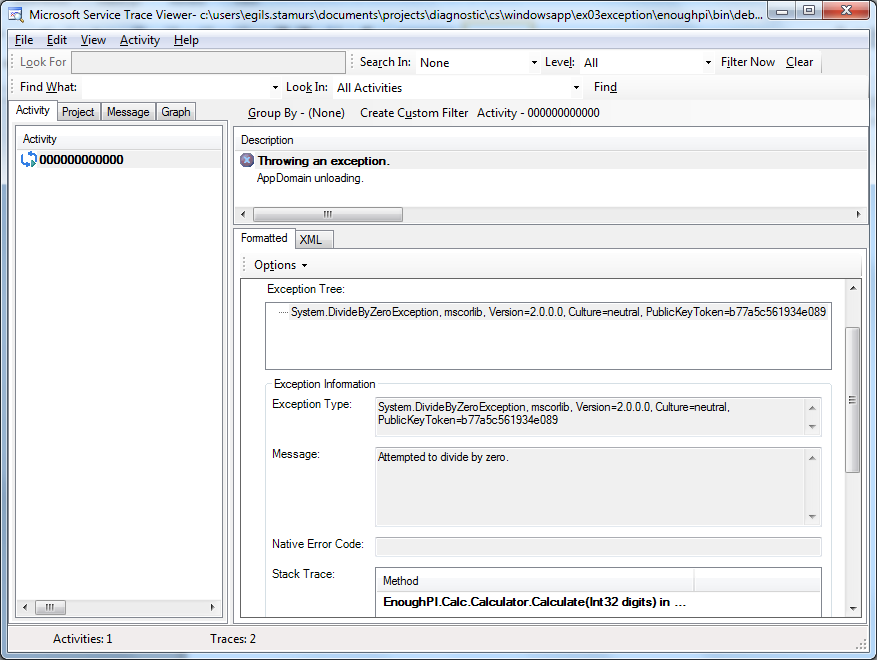
exceptionWriter.ThrowHelperError(args.Exception);

if (CalculatorException != null)

CalculatorException(this, args);

}

1. Šajā gadījumā netiek specificēta ziņojumu kategorija, bet lietota noklusētā kategorija „General”, tāpēc jāizveido atbilstošu ziņojumu klasi app.config failā (3.solis);
2. Rezultāta iegūšanai tika radīta izņēmuma situācija un iegūts sekojošs rezultāts (Ilustrācija 3‑1).



Ilustrācija 3‑1 Izņēmumu žurnalēšanas rezultāti

# LogActivity

Piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu esošam projektam aktivitāšu trasēšanas funkcionalitāti izmantojot LogActivity klasi. Realizētais piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex04Activity”*.

1. Iezīmēt projekta failu Calc\Calculator.cs un izpildīt konteksta izvēlnes komandu View Code;
2. Pievienot vārdtelpu Diagnostic faila augšdaļā;

using Diagnostic;

1. Modificēt konfigurācijas datni app.config, pievienojot sekojošus atribūtus:

* Konfigurācijas sekciju diagnosticConfiguration;

<configSections>

<section name="diagnosticConfiguration"

type="Diagnostic.Configuration.DiagnosticSettings,

Diagnostic, Version=1.0.0.0"/>

</configSections>

* Sadaļu system.diagnostics kas satur sadaļas sharedListeners (pēc izvēles) un sources. Sadaļā sharedListeners definēt koplietojamos trasēšanas failus norādot sekojošus atribūtus: atrašanās vieta – initializeData, šajā gadījumā lietojuma katalogā, tips – type, identifikators – name, papildus žurnalēšanas iespējas – traceOutputOptions un citi atribūti pēc nepieciešamības;

<system.diagnostics>

<sharedListeners>

<add initializeData="Activity.svclog"

type="System.Diagnostics.XmlWriterTraceListener, System,

Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="ActivityLog"

traceOutputOptions="Timestamp"/>

</sharedListeners>

* Sadaļā sources definēt ziņojumu klases source ar identifikatoriem name = *„General”* un „Activity”. Klasēm piesaistīt vismaz vienu listeners failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora name, vai definējot jaunu failu.

<sources>

<source name="General" switchValue="All">

<listeners>

<add name="ActivityLog"/>

</listeners>

</source>

<source name="Activity" switchValue ="All">

<listeners>

<add name="ActivityLog"/>

</listeners>

</source>

</sources>

* Sadaļā system.diagnostics definēt atribūtu trace ar īpašību autoflush un vērtību true. Kas norāda ka rakstīšana failā tiks veikta uzreiz.

<trace autoflush="true" />

</system.diagnostics>

1. Failā Calc\Calculator.cs izsaukt LogActivity metodi UseDiagnosticTrace, lai iespējotu aktivitāšu trasēšanu.

LogActivity.UseDiagnosticTrace(new LogUtility("EnoughPI"));

1. Projekta kodā pēc nepieciešamības izvietot aktivitāšu trasēšanas metodes CreateBoundedActivity. Norādot aktivitātes darbības apgabalu izmantojot using operatoru.

protected void OnCalculated(CalculatedEventArgs args)

{

using (var la = LogActivity.CreateBoundedActivity(false))

{

la.Start("Calculated", "type");

if (Calculated != null)

Calculated(this, args);

}

}

Vai lietojot metodes Start() un Dispose();

protected void OnCalculatorException(CalculatorExceptionEventArgs args)

{

LogActivity.CreateBoundedActivity(false).Start("CalculatorException",

"type");

if (CalculatorException != null)

CalculatorException(this, args);

LogActivity.Current.Dispose();

}

1. Asinhronu metožu izsaukumu gadījumā:

* Definēt LogActivity tipa mainīgo klases Calculator ietvaros;

private LogActivity la;

* Izveidot aktivitāti CreateBoundedActivity asinhronajā izsaukumā un uzsākt tās trasēšanu izsaucot metodi Start();

public IAsyncResult BeginCalculate(int digits)

{

la = LogActivity.CreateBoundedActivity();

la.Start("BeginCalculate", "type");

dlg = new CalculateDelegate(this.Calculate);

AsyncCallback callback = new AsyncCallback(this.CalculateCallback);

return dlg.BeginInvoke(digits, callback, new object[] {dlg, la});

}

* Apturēt izveidoto aktivitāti izsaucot metodi Dispose() atgriežoties no asinhronā izsaukuma.

private void CalculateCallback(IAsyncResult ar)

{

CalculateDelegate dlg = (CalculateDelegate) (ar.AsyncState as

object[])[0];

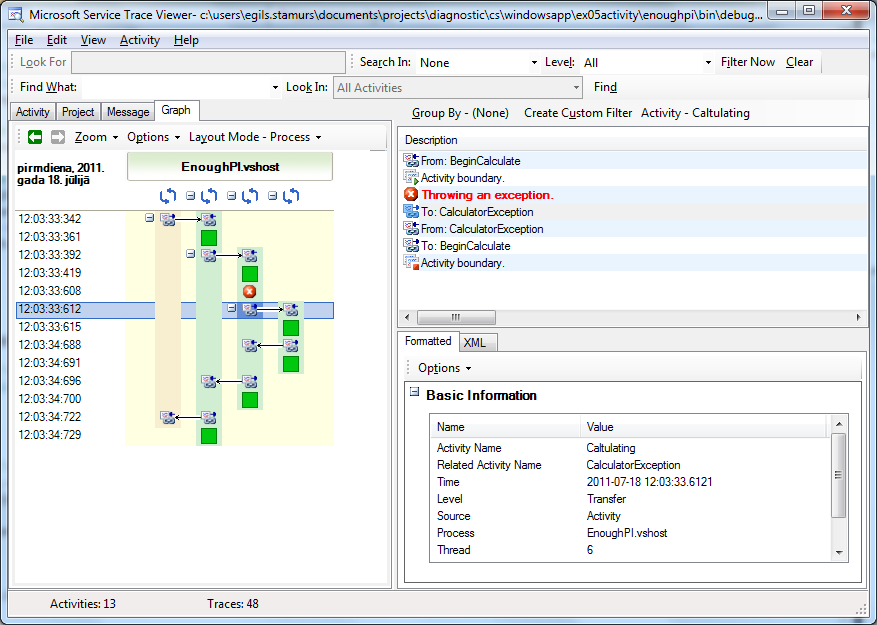
dlg.EndInvoke(ar);

LogActivity la = (LogActivity)(ar.AsyncState as object[])[1];

la.Dispose();

}

1. Trasējot aktivitātes netiek specificēta ziņojumu kategorija, bet lietotas noklusētās kategorijas „Activity” un „General”, tāpēc jāizveido atbilstošas ziņojumu klases app.config failā (3.solis);
2. Rezultāta iegūšanai tika radīta arī izņēmuma situācija un veikta tās žurnalēšana izmantojot ExceptionUtility un iegūts sekojošs rezultāts (Ilustrācija 4‑1).



Ilustrācija 4‑1 Aktivitāšu žurnalēšanas rezultāti

# ExtraInformationProvider

Klase ExtraInformationProvider nodrošina lietotāja veidotu īpašību iekļaušanu žurnālu failos. Piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu ExtraInformationProvider funkcionalitāti iepriekš aplūkotajai žurnalēšanai izmantojot LogUtility.

Realizētais piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex05EnterpriseLib”*.

1. Pievienot klasēm sekojošas vārdteplas:

using System.Collections.Generic;

using Diagnostic.ExtraInformation;

1. Izveidot klasi ar IExtraInformationProvider interfeisu un implementēt tā metodi PopulateDictionary. Metode aizpilda Dictionary tipa mainīgo ar īpašībām un to vērtībām. Izveidot klases konstruktoru, kuram kā atribūti tiks padotas žurnalēšanas failos ierakstāmās vērtības;

public class ExtraInfoProvider : IExtraInformationProvider

{

private string value;

private int digits;

public ExtraInfoProvider(string value, int digits)

{

this.value = value;

this.digits = digits;

}

public void PopulateDictionary(IDictionary<string, object> dictionary)

{

if (this.value != null)

{

dictionary.Add("PI", this.value);

dictionary.Add("Digits", this.digits);

}

}

}

1. Lai ierakstītu žurnalēšanas failā vēlamās īpašības nepieciešams veikt sekojošas darbības:
   * Definēt Dictionary tipa mainīgo;
   * Izveidot klases eksemplāru un konstruktorā padot failā rakstāmās vērtības;
   * Izsaukt klases eksemplāra metodi PopulateDictionary un tās parametrā uzdot iepriekš definēto Dictionary tipa mainīgo;
   * Izsaukt LogUtility metodi Write un parametros norādīt ziņojuma tekstu un definēto Dictionary tipa mainīgo. [[1](#KBA)]

Dictionary<string, object> p = new Dictionary<string, object>();

var x = new ExtraInfoProvider(args.Pi, args.Digits);

x.PopulateDictionary(p);

logWriter.Write("Result", p);

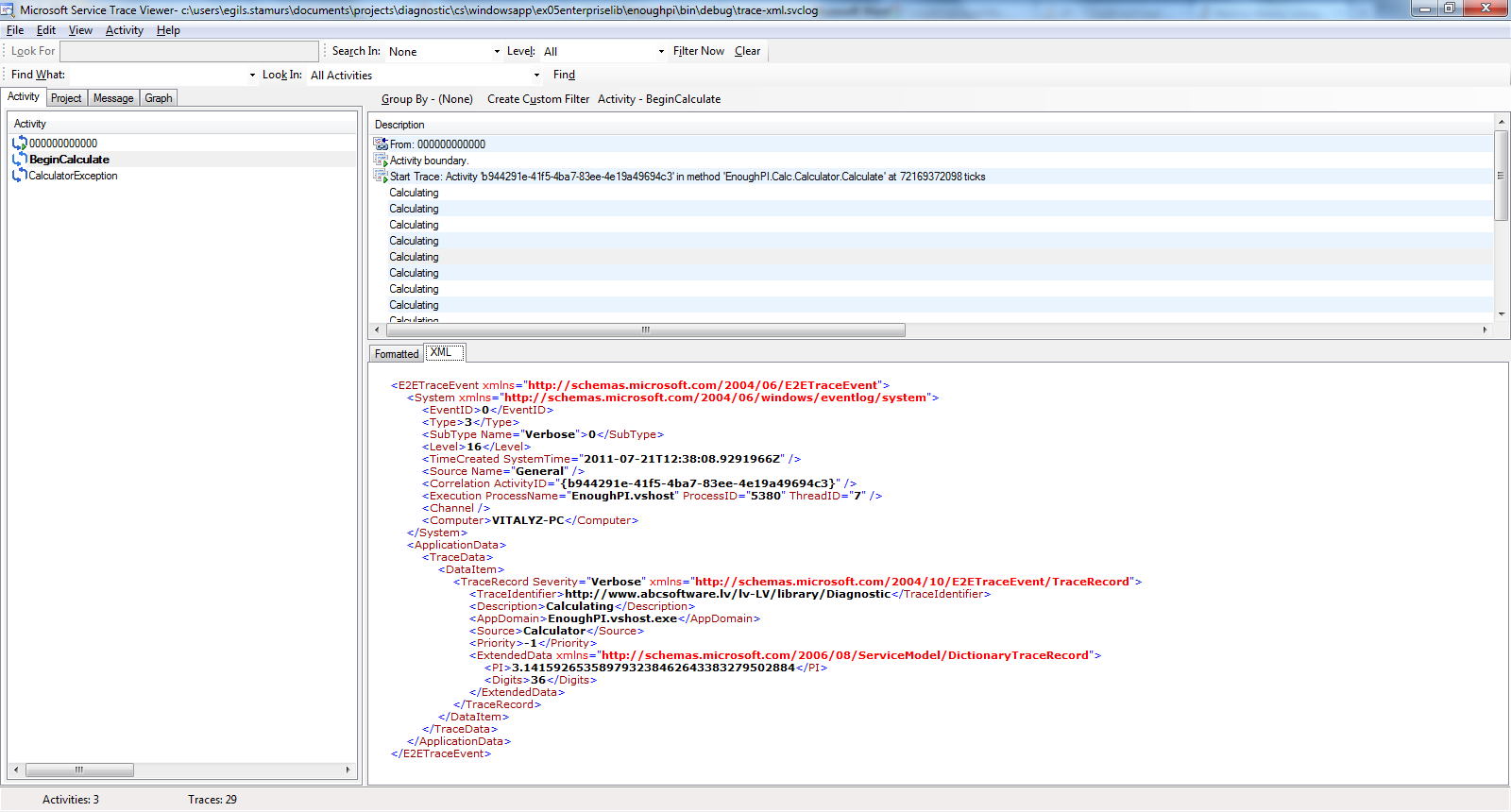
1. To pašu rezultātu iespējams sasniegt neveidojot atsevišķu klasi Dictionary tipa mainīgā aizpildīšanai, bet gan veicot sekojošas darbības:
   * Definēt Dictionary tipa mainīgo;
   * Pievienot mainīgajam nepieciešamās īpašības un to vērtības izsaucot metodi Add;
   * Izsaukt LogUtility metodi Write parametros norādot ziņojuma tekstu un definēto Dictionary tipa mainīgo. [[1](#KBA)]

Dictionary<string,object> p = new Dictionary<string,object>();

p.Add("Pi",args.Pi);

logWriter.Write("Result", p);

1. Iegūtais rezultāts pievienojot īpašības Pi un Digits (Ilustrācija 5‑1);



Ilustrācija 5‑1 ExtraInformationProvider pielietošanas rezultāti

# Enterprise Library 4.0.0.0

Visas iepriekš aplūkotās žurnalēšanas un trasēšanas funkcijas var tikt realizētas ne tikai izmantojot Diagnostic bibliotēku, bet arī izmantojot Enterprise Library bibliotēkas, bez jebkādām izmaiņām kodā. Lai to realizētu nepieciešams veikt Enterprise Library konfigurēšanu. Pēc noklusējuma tiek lietota Diagnostic.dll, lai veiktu rakstīšanu žurnalēšanas failos, bet iekļaujot Enterprise Library bibliotēkas tiks izmantotas tās.

Piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu un konfigurētu Enterprise Library bibliotēkas esošam projektam. Realizētais piemērs uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ ex05EnterpriseLib”*.

1. Atvērt projektu EnoughPI un iezīmēt katalogu References, izpildīt konteksta izvēlnes komandu Add Reference un pievienot projektam sekojošas bibliotēkas (tipiski atrodamas katalogā *„~\Diagnostic\Lib”*):

* Microsoft.Practices.EnterpriseLibrary.Logging.dll;
* Microsoft.Practices.EnterpriseLibrary.Common.dll;
* Microsoft.Practices.ObjectBuilder2.dll.

1. Modificēt konfigurācijas datni app.config, papildus pievienojot sekojošus atribūtus:

* Konfigurācijas sekciju enterpriseLibrary.ConfigurationSource; [[3](#ELconfig)]

<configSections>

<section name="enterpriseLibrary.ConfigurationSource"

type="Microsoft.Practices.EnterpriseLibrary.Common.

Configuration.ConfigurationSourceSection,

Microsoft.Practices.EnterpriseLibrary.Common,

Version=4.0.0.0, Culture=neutral,

PublicKeyToken=31bf3856ad364e35"/>

</configSections>

* Sadaļu enterpriseLibrary.ConfigurationSource kas satur sadaļu sources. Tajā norāda Enterprise Library ārējos konfigurācijas failus, to nosaukumu un atrašanās vietu - filePath, tipu - type un identifikatoru - name. Kā arī norāda pielietotā konfigurācijas faila identifikatoru - name parametrā selectedSource.

<enterpriseLibrary.ConfigurationSource selectedSource="File Configuration Source">

<sources>

<add name="File Configuration Source"

type="Microsoft.Practices.EnterpriseLibrary.Common.

Configuration.FileConfigurationSource,

Microsoft.Practices.EnterpriseLibrary.Common,

Version=4.0.0.0, Culture=neutral,

PublicKeyToken=31bf3856ad364e35"

filePath="Enterprise.config"/>

</sources>

</enterpriseLibrary.ConfigurationSource>

1. Izveidot Enterprise Library konfigurācijas datni, piemēram – *„Enterprise.config”*, un pievienojot sekojošus atribūtus:

* Konfigurācijas sekcijas loggingConfiguration un dataConfiguration;

<configSections>

<section name="loggingConfiguration"

type="Microsoft.Practices.EnterpriseLibrary.Logging.

Configuration.LoggingSettings,

Microsoft.Practices.EnterpriseLibrary.Logging,

Version=4.0.0.0, Culture=neutral,

PublicKeyToken=31bf3856ad364e35"

requirePermission="false" />

<section name="dataConfiguration"

type="Microsoft.Practices.EnterpriseLibrary.Data.

Configuration.DatabaseSettings,

Microsoft.Practices.EnterpriseLibrary.Data, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35" />

</configSections>

* sekcijā loggingConfiguration definēt sekojošas sadaļas:
  + listeners;
  + formatters;
  + categorySources;
  + specialSources.

<loggingConfiguration name="Logging Application Block"

tracingEnabled="true"

defaultCategory="General" logWarningsWhenNoCategoriesMatch="true">

* sadaļa listeners satur žurnalēšanas failus un to atribūtus;

<listeners>

<add fileName="trace.log" header="-----------------------------------"

footer="----------------------------------------"

formatter="Text Formatter"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.

Configuration.FlatFileTraceListenerData,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

traceOutputOptions="DateTime" filter="All"

type="Microsoft.Practices.EnterpriseLibrary.Logging.

TraceListeners.FlatFileTraceListener,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

name="FlatFile TraceListener" />

<add fileName="unp.svclog"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.

Logging.Configuration.XmlTraceListenerData,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

traceOutputOptions="DateTime" filter="All"

type="Microsoft.Practices.EnterpriseLibrary.Logging.

TraceListeners.XmlTraceListener,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

name="unprocessed" />

<add fileName="trace-xml.svclog"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.

Configuration.XmlTraceListenerData,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

traceOutputOptions="Timestamp" filter="All"

type="Microsoft.Practices.EnterpriseLibrary.Logging.

TraceListeners.XmlTraceListener,

Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=31bf3856ad364e35"

name="XML Trace Listener" />

</listeners>

* Sadaļa formatters satur ziņojumu šablonus;

<formatters>

<add template="Timestamp: {timestamp}&#xD;&#xA;Message:

{message}&#xD;&#xA;Category: {category}&#xD;&#xA;Priority:

{priority}&#xD;&#xA;EventId: {eventid}&#xD;&#xA;Severity:

{severity}&#xD;&#xA;Title:{title}&#xD;&#xA;Machine:

{machine}&#xD;&#xA;Thread Name: {threadName}&#xD;&#xA;Extended

Properties: {dictionary({key} - {value}&#xD;&#xA;)}"

type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.

TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging,

Version=4.0.0.0, Culture=neutral,PublicKeyToken=31bf3856ad364e35"

name="Text Formatter" />

</formatters>

* Sadaļa categorySources satur ziņojumu kategorijas un tām piesaistītos žurnalēšanas failus;

<categorySources>

<add switchValue="All" name="Activity">

<listeners>

<add name="FlatFile TraceListener" />

<add name="XML Trace Listener" />

</listeners>

</add>

<add switchValue="All" name="General">

<listeners>

<add name="FlatFile TraceListener" />

<add name="XML Trace Listener" />

</listeners>

</add>

<add switchValue="All" name="Log">

<listeners>

<add name="FlatFile TraceListener" />

<add name="XML Trace Listener" />

</listeners>

</add>

<add switchValue="All" name="Trace">

<listeners>

<add name="FlatFile TraceListener" />

<add name="XML Trace Listener" />

</listeners>

</add>

</categorySources>

* Sadaļa specialSources satur specifiskas ziņojumu kategorijas un tām piesaistītos žurnalēšanas failus.

<specialSources>

<allEvents switchValue="All" name="All Events" />

<notProcessed switchValue="All" name="Unprocessed Category" />

<errors switchValue="All" name="Logging Errors &amp; Warnings">

<listeners>

<add name="unprocessed" />

</listeners>

</errors>

</specialSources>

</loggingConfiguration>

# Versijas

Diagnostic.dll V1.0 for .NET 2.0-4.0 (File Version 1.0.0.8)

Abc.Diagnostics.dll V1.1 for .NET 4.5

* Izmanījās vārdu telpa no Dignostic uz Abc.Diagnostics
* Uztur EntrpriceLibrary 6.0

IVIS.Diagnostics V1.0 for .NET3.5-4.0 (File Version 1.0.0.0)

IVIS.Diagnostics V1.1 for .NET4.5

* Atsauksme uz Abc.Diagnostics V1.1
* Metodēs WriteAudit, SendNotificatoin izmantojas optional parametri.