Nacionālais veselības dienests

|  |  |
| --- | --- |
|  |  |

Integrācijas platformas informācijas sistēmas izstrāde

Koplietojuma bibliotēku apraksts. I.sējums.

Standarts

NVD.IP.STD.KBA\_I

20.02.2013. versija 1.03

Rīgā 2013

|  |  |  |
| --- | --- | --- |
|  | **IEGULDĪJUMS TAVĀ NĀKOTNĒ** |  |

Dokumenta identifikācija

|  |  |
| --- | --- |
| Dokumenta ID: | NVD.IP.STD.KBA\_I.1.03 |
| Dokumenta nosaukums: | Integrācijas platformas informācijas sistēmas izstrāde.  Koplietojuma bibliotēku apraksts. I.sējums.  Standarts |
| Dokumenta kods: | NVD.IP.STD.KBA\_I |
| Versija: | Versija 1.03, Laidiens 20.02.2013. |

Saskaņojumi

|  |  |  |  |
| --- | --- | --- | --- |
| Organizācija | Vārds, uzvārds, amats | Datums | Paraksts |
| Nacionālais veselības dienests | L.Freimane, projekta vadītāja no Pasūtītāja puses |  |  |
| SIA "ABC software" | J.Korņijenko, projekta vadītājs no Izpildītāja puses | 20.02.2013. |  |
| SIA "ABC software" | E.Blumberga, projekta kvalitātes kontroles vadītāja | 20.02.2013. |  |

|  |
| --- |
| *Visas tekstā izmantotās preču zīmes pieder to īpašniekiem un ir izmantotas tikai kā atsauces.* |

Izmaiņu vēsture

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versija | Datums | Apraksts | Organizācija | Autors |
| 1.00 | 02.07.2012. | Izveidota dokumenta sākotnējā versija | SIA „ABC software” | E.Stāmurs |
| 1.01 | 04.10.2012. | Pievienots 3. Nodalījums „Logošanas pieslēgšana lietojumam (Logošana ar Enterprise Library)”. | SIA „ABC software” | P.Semenčuks |
| 1.02 | 18.12.2012. | Precizējumi saskaņā ar Enterprise Library 5.0 izmantošanu | SIA „ABC software” | E.Stāmurs |
| 1.03 | 20.02.2013. | Veikti redakcionāli labojumi | SIA „ABC software” | E.Stāmurs |

No 2011.gada 1.novembra, pamatojoties uz Ministru kabineta 2011.gada 7.septembra rīkojuma Nr.436 „Par Veselības norēķinu centra un Veselības ekonomikas centra reorganizāciju” 2.punktu, Nacionālais veselības dienests ir Veselības ekonomikas centra funkciju, pārvaldes uzdevumu, tiesību, saistību, finanšu līdzekļu, mantas un lietvedības pārņēmējs. Visā dokumentā Veselības ekonomikas centrs (VEC) pielīdzināms Nacionālais veselības dienests (NVD).

**Satura rādītājs**

[Attēlu saraksts 6](#_Toc380756791)

[1. Ievads 7](#_Toc380756792)

[1.1. Dokumenta nolūks 7](#_Toc380756793)

[1.2. Termini un pieņemtie apzīmējumi 7](#_Toc380756794)

[1.3. Saistība ar citiem dokumentiem 7](#_Toc380756795)

[2. Palīgklases 8](#_Toc380756796)

[2.1. Logošanas palīgklase 8](#_Toc380756797)

[2.1.1. Konstruktori 8](#_Toc380756798)

[2.1.1.1. Konstruktors „LogUtility” 8](#_Toc380756799)

[2.1.2. Īpašības 8](#_Toc380756800)

[2.1.2.1. „LoggingEnabled” īpašība 8](#_Toc380756801)

[2.1.3. Metodes 9](#_Toc380756802)

[2.1.3.1. Metode „Write” 9](#_Toc380756803)

[2.1.4. Klases izmantošanas piemērs 9](#_Toc380756804)

[2.2. Izņēmumu palīgklase 10](#_Toc380756805)

[2.2.1. Konstruktori 10](#_Toc380756806)

[2.2.1.1. Konstruktors „ExceptionUtility” 10](#_Toc380756807)

[2.2.2. Metodes 10](#_Toc380756808)

[2.2.2.1. Metode „ThrowHelper” 10](#_Toc380756809)

[2.2.2.2. Metode „ThrowHelperWarning” 10](#_Toc380756810)

[2.2.2.3. Metode „ThrowHelperError” 11](#_Toc380756811)

[2.2.2.4. Metode „ThrowHelperCritical” 11](#_Toc380756812)

[2.2.2.5. Metode „ThrowHelperFatal” 11](#_Toc380756813)

[2.2.2.6. Metode „IsFatal” 11](#_Toc380756814)

[2.2.2.7. Metode „ThrowHelperArgument” 11](#_Toc380756815)

[2.2.2.8. Metode „ThrowHelperArgument” 12](#_Toc380756816)

[2.2.2.9. Metode „ThrowHelperArgumentNull” 12](#_Toc380756817)

[2.2.2.10. Metode „ThrowHelperArgumentNull” 12](#_Toc380756818)

[2.2.2.11. Metode „UseActivityId” 12](#_Toc380756819)

[2.2.2.12. Metode „ClearActivityId” 13](#_Toc380756820)

[2.2.2.13. Metode „TraceHandeledException” 13](#_Toc380756821)

[2.2.3. Klases izmantošanas piemērs 13](#_Toc380756822)

[2.3. Trasēšanas palīgklase 13](#_Toc380756823)

[2.3.1. Konstruktori 14](#_Toc380756824)

[2.3.1.1. Konstruktors „TraceUtilty” 14](#_Toc380756825)

[2.3.2. Īpašības 14](#_Toc380756826)

[2.3.2.1. „IsTracingEnabled” īpašība 14](#_Toc380756827)

[2.3.3. Metodes 14](#_Toc380756828)

[2.3.3.1. Metode „StartTrace” 14](#_Toc380756829)

[2.3.3.2. Metode „StartTrace” 14](#_Toc380756830)

[2.3.3.3. Metode „Dispose” 15](#_Toc380756831)

[2.3.4. Klases izmantošanas piemērs 15](#_Toc380756832)

[2.4. Auditācijas palīgklase 15](#_Toc380756833)

[2.4.1. Konstruktori 15](#_Toc380756834)

[2.4.1.1. Konstruktors „EvesDiagnostic” 15](#_Toc380756835)

[2.4.2. Metodes 16](#_Toc380756836)

[2.4.2.1. Metode „WriteAudit” 16](#_Toc380756837)

[2.4.3. Auditācijas papildus īpašību „properties” aizpildīšana 16](#_Toc380756838)

[2.4.3.1. Aizpildīšanas scenārijs 17](#_Toc380756839)

[2.4.4. Klases izmantošanas piemērs 19](#_Toc380756840)

[2.5. Aktivitāšu palīgklase 20](#_Toc380756841)

[2.5.1. Metodes 21](#_Toc380756842)

[2.5.1.1. Metode „CreateActivity” 21](#_Toc380756843)

[2.5.1.2. Metode „CreateAsyncActivity” 21](#_Toc380756844)

[2.5.1.3. Metode „CreateBoundedActivity” 21](#_Toc380756845)

[2.5.1.4. Metode „Dispose” 22](#_Toc380756846)

[2.5.1.5. Metode „Suspend” 22](#_Toc380756847)

[2.5.1.6. Metode „Resume” 22](#_Toc380756848)

[2.5.1.7. Metode „Stop” 22](#_Toc380756849)

[2.5.1.8. Metode „Start” 22](#_Toc380756850)

[2.5.2. Klases izmantošanas piemērs 22](#_Toc380756851)

[2.6. Klašu izmantošanas piemērs 23](#_Toc380756852)

[3. Klašu izmantošanas piemēri 26](#_Toc380756853)

[3.1. Vispārējie norādījumi 26](#_Toc380756854)

[3.2. LogUtility 26](#_Toc380756855)

[3.3. Personalizētais audits 28](#_Toc380756856)

[3.4. TraceUtility 29](#_Toc380756857)

[3.5. ExceptionUtility 31](#_Toc380756858)

[3.6. LogActivity 33](#_Toc380756859)

[3.7. ExtraInformationProvider 36](#_Toc380756860)

[3.8. Enterprise Library 5.0.0.0 37](#_Toc380756861)

[3.9. WCF Servisa un klienta trasēšana 37](#_Toc380756862)

[3.9.1. Klienta konfigurācijas datne 37](#_Toc380756863)

[3.9.2. Servisa konfigurācijas datne 39](#_Toc380756864)

[3.9.3. Žurnalēšana ar Diagnostic.dll 40](#_Toc380756865)

[3.9.4. Sinhrons servisa izsaukums 41](#_Toc380756866)

[3.9.5. Asinhrons servisa izsaukums 42](#_Toc380756867)

# Attēlu saraksts

[1.attēls. Logošanas palīgklasē iekļautās metodes 8](#_Toc380756868)

[2.attēls. Izņēmumu palīgklasē iekļautās metodes 10](#_Toc380756869)

[3.attēls. Trasēšanas palīgklasē iekļautās metodes 13](#_Toc380756870)

[4.attēls. Auditācijas palīgklasē iekļautās metodes 15](#_Toc380756871)

[5.attēls Aktivitāšu palīgklasē iekļautās metodes un īpašības 20](#_Toc380756872)

[6.attēls. Žurnalēšanas rezultāti 28](#_Toc380756873)

[7.attēls. Trasēšanas rezultāti 31](#_Toc380756874)

[8.attēls. Izņēmumu žurnalēšanas rezultāti 33](#_Toc380756875)

[9.attēls. Aktivitāšu žurnalēšanas rezultāti 35](#_Toc380756876)

[10.attēls. ExtraInformationProvider pielietošanas rezultāti 37](#_Toc380756877)

[11.attēls. Sinhrona izsaukuma rezultāts 42](#_Toc380756878)

[12.attēls. Asinhrona izsaukuma rezultāts 43](#_Toc380756879)

# Ievads

## Dokumenta nolūks

Dokuments „Koplietojuma žurnalēšanas bibliotēku apraksts” satur žurnalēšanas bibliotēku Diagnostic.dll un Enterprise Library 4.0.0.0 aprakstu un palīgklašu izmantošanas instrukcijas. Šis dokuments ir paredzēts SIA „ABC software” izstrādātājiem, kā arī citiem iesaistītajiem izstrādātājiem, kas līdzdarbojas ar SIA „ABC software” programmnodrošinājumu izstrādē un pilnveidošanā.

## Termini un pieņemtie apzīmējumi

Visi šajā dokumentā izmantotie termini un apzīmējumi ir apkopoti Terminu un saīsinājumu indeksā **Error! Reference source not found.**.

## Saistība ar citiem dokumentiem

Dokuments ir izstrādāts, balstoties uz šādiem dokumentiem:

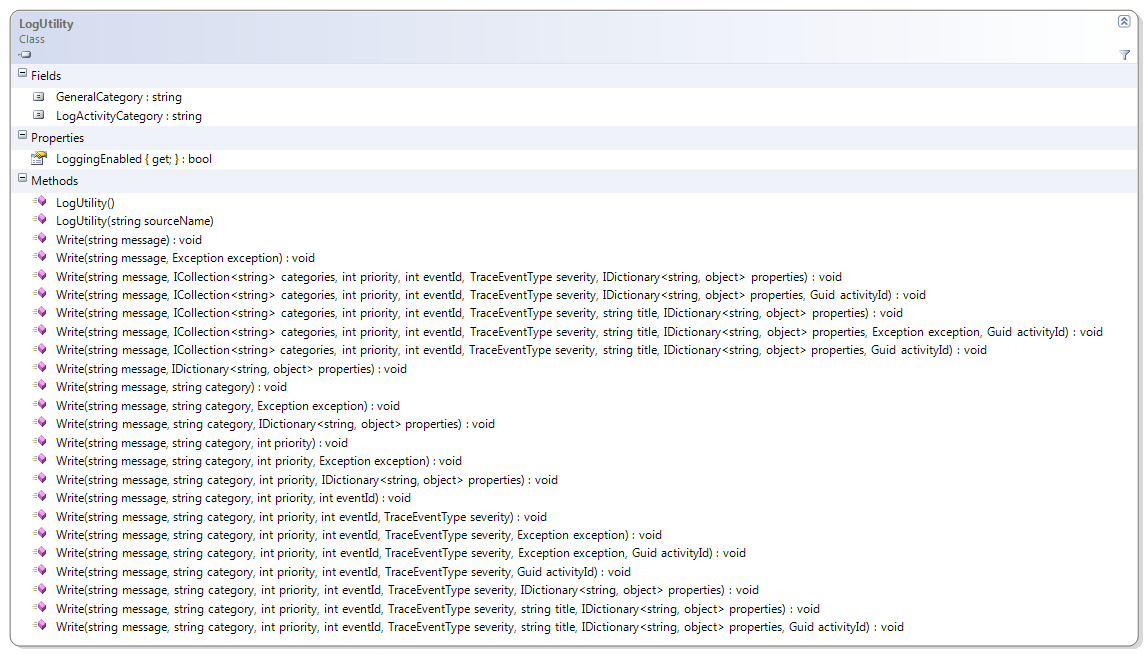
1. Integrācijas platformas informācijas sistēmas izstrāde. Programmatūras izstrādes vadlīnijas. Standarts. VEC.STD.VAD.
2. Integrācijas platformas informācijas sistēmas izstrāde. Terminu un saīsinājumu indekss (VEC.IP.TSI).
3. Enterprise Library konfigurācijas faili – <http://blogs.msdn.com/b/tomholl/archive/2006/04/02/entlib2externalconfig.aspx>
4. Instrukciju pamatā izmantots risinājums EnoughPI.sln, kas atrodas mapē „~\Diagnostic\CS\WindowsApp\begin”;
5. E-veselības integrācijas platformas infrastruktūra: auditācijas un izpildlaika pierakstu žurnāls (2.sējums). Administratoru rokasgrāmata (NVD.IP.VISS\_2010.STD.Apakšprojekta abreviatūra).

# Palīgklases

Bibliotēkā „Diagnostic.dll” izmantotās palīgklases nodrošina nepieciešamās žurnalēšanas un auditācijas funkcijas. Diagnostic bibliotēka ir balstīta uz .NET Tracing ietvara, skat. http://msdn.microsoft.com/en-us/library/ms730342.aspx.

## Logošanas palīgklase

Logošanas palīgklasē „LogUtility” iekļautas metodes, kas ļauj veikt notikumu žurnalēšanu.



1.attēls. Logošanas palīgklasē iekļautās metodes

### Konstruktori

#### Konstruktors „LogUtility”

public LogUtility(string sourceName)

Izveido klasi „LogUtility” ar uzdoto žurnalēšanas avotu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| sourceName | trasēšanas avots |

### Īpašības

#### „LoggingEnabled” īpašība

Īpašības apraksts:

public bool LoggingEnabled { get; }

Īpašība atgriež, vai žurnalēšana ir ieslēgta.

### Metodes

#### Metode „Write”

Metodes apraksts:

public void Write(string message, string category, int priority, int eventId, TraceEventType severity, string title, IDictionary<string, object> properties, Guid activityId)

Logo ziņojumu ar uzdotu kategoriju, prioritāti, identifikatoru, stingrību un papildus īpašībām.

Metodes parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | ziņojuma teksts |
| category | kategorija, ar kuru tiek rakstīts ziņojums |
| priority | prioritāte, ar kuru tiek rakstīts ziņojums |
| eventId | ziņojuma numurs vai identifikators |
| severity | ziņojuma stingrība |
| title | ziņojuma papildus apraksts |
| properties | papildus parametri |
| activityId | aktivitātes identifikators |

public void Write(string message, string category, int priority, int eventId, TraceEventType severity, Exception exception)

Logo ziņojumu ar uzdotu kategoriju, prioritāti, identifikatoru, stingrību un izņēmumu.

Metodes parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | ziņojuma teksts |
| category | kategorija, ar kuru tiek rakstīts ziņojums |
| priority | Prioritāte, ar kuru tiek rakstīts ziņojums |
| eventId | ziņojuma numurs vai identifikators |
| severity | ziņojuma stingrība |
| exception | izņēmums |
| activityId | aktivitātes identifikators |

### Klases izmantošanas piemērs

// Izveidojam informāciju žurnalēšanai

int eventId = 1;

int priority = 1;

string message = „Simple message”;

string category = „General”

TraceEventType severity = TraceEventType.Information;

// žurnalējam ziņojumu veidojot LogUtility eksemplāru.

LogUtility target = new LogUtility();

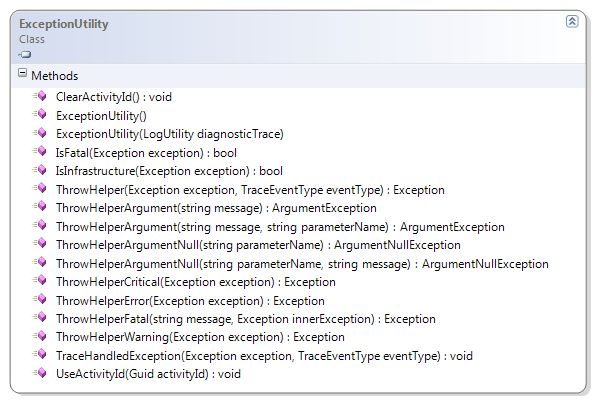
target.Write(message, category, priority, eventId, severity);

// vai izmantojot statisko DiagnosticTools, lai žurnalētu ziņojumu.

DiagnosticTools.LogUtil.Write(message, category, priority, eventId, severity);

## Izņēmumu palīgklase

Izņēmumu palīgklasē iekļautas metodes, kas ļauj apstrādāt izņēmumus.



2.attēls. Izņēmumu palīgklasē iekļautās metodes

### Konstruktori

#### Konstruktors „ExceptionUtility”

Konstruktora apraksts:

public ExceptionUtility()

Izveido klasi ExceptionUtility.

### Metodes

#### Metode „ThrowHelper”

Metodes apraksts:

public Exception ThrowHelper(Exception exception, TraceEventType eventType);

Apstrādā izņēmumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |
| eventType | izņēmuma stingrība |

#### Metode „ThrowHelperWarning”

Metodes apraksts:

public Exception ThrowHelperWarning(Exception exception);

Apstrādā izņēmumu ar stingrību „Brīdinājums”.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |

#### Metode „ThrowHelperError”

Metodes apraksts:

public Exception ThrowHelperError(Exception exception);

Apstrādā izņēmumu ar stingrību „Kļūda”.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |

#### Metode „ThrowHelperCritical”

Metodes apraksts:

public Exception ThrowHelperCritical(Exception exception);

Apstrādā izņēmumu ar stingrību „Kritisks”.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |

#### Metode „ThrowHelperFatal”

Metodes apraksts:

public Exception ThrowHelperFatal(string message, Exception innerException);

Izveido un apstrādā fatālo izņēmumu ar stingrību „Kļūda”.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | izņēmuma ziņojums |
| exception | Izņēmums, kurš radījis fatālo izņēmumu. |

#### Metode „IsFatal”

Metodes apraksts:

public static bool IsFatal(Exception exception);

Pārbauda vai uzdotais izņeēmums ir fatals.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |

#### Metode „ThrowHelperArgument”

Metodes apraksts:

public ArgumentException ThrowHelperArgument(string message);

Izveido un apstrādā parametra izņēmumu *ArgumentException* ar uzdoto aprakstu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | izņēmuma apraksts |

#### Metode „ThrowHelperArgument”

Metodes apraksts:

public ArgumentException ThrowHelperArgument(string message, string paramName);

Izveido un apstrādā parametra izņēmumu *ArgumentException* ar uzdoto aprakstu un parametra nosaukumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | izņēmuma apraksts |
| paramName | parametra nosaukums |

#### Metode „ThrowHelperArgumentNull”

Metodes apraksts:

public ArgumentNullException ThrowHelperArgumentNull(string message);

Izveido un apstrādā parametra izņēmumu *ArgumentNullException* ar uzdoto aprakstu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| message | izņēmuma apraksts |

#### Metode „ThrowHelperArgumentNull”

Metodes apraksts:

public ArgumentNullException ThrowHelperArgumentNull(string paramName, string message);

Izveido un apstrādā parametra izņēmumu *ArgumentNullException* ar uzdoto aprakstu un parametra nosaukumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| paramName | parametra nosaukums |
| message | izņēmuma apraksts |

#### Metode „UseActivityId”

Metodes apraksts:

public void UseActivityId(Guid activityId);

Veikt izņēmuma trasēšanu ar uzdoto aktivitātes identifikatoru.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| activityId | aktivitātes identifikators |

#### 

#### Metode „ClearActivityId”

Metodes apraksts:

public void ClearActivityId();

Noņemt uzdoto aktivitātes identifikatoru.

#### Metode „TraceHandeledException”

Metodes apraksts:

public void TraceHandledException(Exception exception, TraceEventType eventType);

Veic izņēmuma trasēšanu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| exception | izņēmums |
| eventType | izņēmuma stingrība |

### Klases izmantošanas piemērs

public void DoSomesing(string required\_parameter) {

// Izņēmumu apstrāde izmantojot ExceptionUtility eksemplāru.

ExceptionUtility excUtil = new ExceptionUtility()

if (required\_parameter == null){

throw excUtil.ThrowHelperArgumentNull(„required\_parameter”);

}

// vai izmantojot statiskās DiagnosticTools ExceptionUtil metodes.

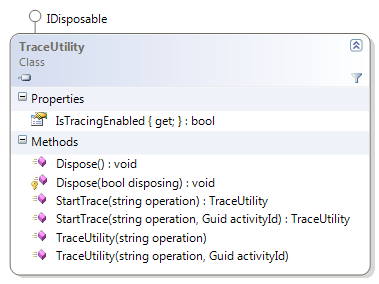
throw DiagnosticTools.ExceptionUtil.ThrowHelperArgumentNull(„required\_parameter”);

...

}

## Trasēšanas palīgklase

Klase palīdz veikt trasēšanas iespējas.



3.attēls. Trasēšanas palīgklasē iekļautās metodes

### Konstruktori

#### Konstruktors „TraceUtilty”

Konstruktora apraksts:

public TraceUtility(string operation)

Izveido klasi *TraceUtility* ar operācijas nosaukumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| operation | operācijas nosaukums |

public TraceUtility(string operation, Guid activityId)

Izveido klasi *TraceUtility* ar operācijas nosaukumu un aktivitātes identifikatoru.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| operation | operācijas nosaukums |
| activityId | aktivitātes identifikators |

### Īpašības

#### „IsTracingEnabled” īpašība

Īpašības apraksts:

public bool IsTracingEnabled { get; }

Īpašība atgriež vai trasēšana ir ieslēgta.

### Metodes

#### Metode „StartTrace”

Metodes apraksts:

public static TraceUtility StartTrace(sting operation, Guid activityId);

Izveido klasi *TraceUtility* ar uzdotu operācijas nosaukumu un aktivitātes identifikatoru.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| Operation | operācijas nosaukums |
| activityId | aktivitātes identifikators |

#### Metode „StartTrace”

Metodes apraksts:

public static TraceUtility StartTrace(sting operation);

Izveido klasi *TraceUtility* ar uzdotu operācijas nosaukumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| operation | operācijas nosaukums |

#### Metode „Dispose”

Metodes apraksts:

public void Dispose();

Atbrīvo *TraceUtility klases eksemplāra aizņemtos resursus.*

### Klases izmantošanas piemērs

public void DoSomesing() {

// Trasēšana izmantojot konstruktoru un „using” bloku.

using (new TraceUtility("DoSomesing")) {

...

}

// Vai trasēšana izmantojot metodi StartTrace un „using” bloku.

using (TraceUtility.StartTrace("DoSomesing")) {

...

}

// Vai trasēšana izmantojot metodes StartTrace un Dispose.

TraceUtility tracer = TraceUtility.StartTrace("DoSomesing");

...

tracer.Dispose();

// Vai trasēšana izmantojot konstruktoru un Dispose.

TraceUtility tracer = new TraceUtility("DoSomesing");

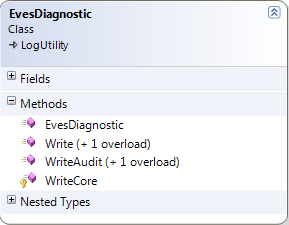
...

tracer.Dispose();

}

## Auditācijas palīgklase

Klase palīdz veikt auditāciju un žurnalēšanu (Log).



4.attēls. Auditācijas palīgklasē iekļautās metodes

### Konstruktori

#### Konstruktors „EvesDiagnostic”

Konstruktora apraksts:

public EvesDiagnostic(string sourceName)

Izveido klasi *EvesDiagnostic* ar bibliotēkas nosaukumu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| sourceName | bibliotēkas nosaukums |

### Metodes

#### Metode „WriteAudit”

Metodes apraksts:

public void WriteAudit(string actionType, string reason, int eventId, string message)

Metode kas raksta auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| actionType | Darbības veids (klasifikatora OID - 1.3.6.1.4.1.38760.2.15) |
| reason | Personas datu apstrādes pamatojums (klasifikatora OID - 1.3.6.1.4.1.38760.2.4) |
| eventId | Sistēmas stāvokļa identifikators (klasifikatora OID - 1.3.6.1.4.1.38760.2.14) |
| message | Auditācijas ziņojums |

Metodes apraksts:

public void WriteAudit(string actionType, string reason, int eventId, string message, IDictionary<string, object> properties)

Metode kas raksta auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| actionType | Darbības veids (klasifikatora OID - 1.3.6.1.4.1.38760.2.15) |
| reason | Personas datu apstrādes pamatojums (klasifikatora OID - 1.3.6.1.4.1.38760.2.4) |
| eventId | Sistēmas stāvokļa identifikators (klasifikatora OID - 1.3.6.1.4.1.38760.2.14) |
| message | Auditācijas ziņojums |
| properties | Papildus īpašības |

### Auditācijas papildus īpašību „properties” aizpildīšana

Papildus īpašības ir *KeyValuePair<string, object>* parametrs, kur *string* ir auditācijas ieraksta atslēga, kas ir reģistrēta datu bāze, bet *object* ir:

* *String* parametrs – parametrs, kas satur atslēgas vērtību;
* *ExtendedPropertiesValueStructure* struktūra – struktūra, kas satur atslēgas vērtību. Struktūra sastāv no *Value* parametra un *OID* parametra. *Value* ir atslēgas konkrēta vērtība, bet *OID* ir atslēgas HL7 klasifikatora identifikators.
* *LVAU\_MT000004UV01SubjectStructure* struktūra – struktūra HL7 shēmas aizpildīšanai. Izmantojot šo struktūru atslēga var būt jebkura teksta rinda. Atslēgu atbilstība HL7 shēmas elementiem tiek definēta 1.tabulā.

1.tabula

Atslēgu atbilstība HL7 shēmas elementiem

| HL7 elements | Atslēga | Apraksts |
| --- | --- | --- |
| LVAU\_MT000004UV01SubjectStructure.documentReference.  description | AuditSubject\_description | Auditācijas ieraksta kopējais apraksts. Ir jāaizpilda sekojošie lauki:   * description – auditācijas ieraksta kopējais apraksts. |
| LVAU\_MT000004UV01SubjectStructure.documentReference.  sourceName.id | AuditSubject\_source\_id | Informācijas sistēma kur atrodas pamata dokuments ar kuru notiek darbība. Ir jāaizpilda sekojošie lauki:   * id.Extension – informācijas sistēma kur atrodas pamata dokuments ar kuru notiek darbība; * id.Root – OID – 1.3.6.1.4.1.38760.2.3. |
| LVAU\_MT000004UV01SubjectStructure.documentReference.  documentId | AuditSubject\_document\_id | Dokumenta identifikators šī informācijas sistēmā. Ir jāaizpilda sekojošie lauki:   * documentId.Extension – dokumenta identifikators šajā informācijas sistēmā; * documentId.Root – OID – 1.3.6.1.4.1.38760.3.4.1. |
| LVAU\_MT000004UV01SubjectStructure.personDataAuditSubject.  documentReference.description | AuditSubject\_Person\_description | Apraksts par personu, par kuru tiek veikts audits. Ir jāaizpilda sekojošie lauki:  description – apraksts par personu, par kuru tiek veikts audits. |
| LVAU\_MT000004UV01SubjectStructure.personDataAuditSubject.  documentReference.sourceName.id | AuditSubject\_Person\_source\_id | Sistēma, kur tiek uzturēts referencejamais dokuments. Ir jāaizpilda sekojošie lauki:  id.Extension – sistēma, kur tiek uzturēts referencejamais dokuments;  id.Root – OID – 1.3.6.1.4.1.38760.2.3. |
| LVAU\_MT000004UV01SubjectStructure.personDataAuditSubject.  documentReference.documentId | AuditSubject\_Person\_document\_id | Referencējama dokumenta identifikators atbilstošā sistēmā. Ir jāaizpilda sekojošie lauki:  documentId.Extension – referencējama dokumenta identifikators atbilstošā sistēmā;  documentId.Root – OID – 1.3.6.1.4.1.38760.3.4.1. |
| LVAU\_MT000004UV01SubjectStructure.personDataAuditSubject.  assignedPerson.id | AuditSubject\_Person\_code | Personas personas kods, par kuru tiek veikts audits. Ir jāaizpilda sekojošie lauki:  id.personCode – personas kods;  id.OID – OID – 1.3.6.1.4.1.38760.3.1.1. |
| LVAU\_MT000004UV01SubjectStructure.personDataAuditSubject.  assignedPerson.assignedPerson.name | AuditSubject\_Person\_name\_given,  AuditSubject\_Person\_name\_family | Personas vārds un uzvārds, par kuru tiek veikts audits.. Ir jāaizpilda sekojošie lauki:  name.family – personas vārds;  name.given – personas uzvārds. |

#### Aizpildīšanas scenārijs

Šajā paragrāfā doti XML shēmas aizpildīšanas scenārija piemēri ar informāciju par to kādas vērtības jāaizpilda laukos:

* **msgID** (ExtendedPropertyKeys.Id – "Ziņojuma id" jeb LVAU\_MT000004UV01.DocumentReference. documentId.extension);
* **baseID** („PersonDataAuditSubject”.documentReference.documentId – "Dokumenta identifikators šajā informācijas sistēmā (OID – 1.3.6.1.4.1.38760.3.4.1) – tā saucamais pamata dokuments" jeb LVAU\_MT000004UV01.DocumentReference.documentId.root);
* **refID** (var būt vairāki) („PersonDataAuditSubject".personDataAuditSubject.documentReference.documentId – "Dokumenta identifikators atbilstošā sistēmā (OID – 1.3.6.1.4.1.38760.3.4.1) – tā saucamais references dokuments" jeb LVAU\_MT000004UV01.personDataAuditSubject. DocumentReference.documentId.extension);
* **refBaseID** (var būt vairāki) "PersonDataAuditSubject".personDataAuditSubject.documentReference.documentId – "Dokumenta identifikators atbilstošā sistēmā (OID – 1.3.6.1.4.1.38760.3.4.1) – tā saucamais references dokuments" jeb LVAU\_MT000004UV01.personDataAuditSubject. DocumentReference.documentId.root);
* **refPersID** (Parasti personas kods – var būt vairāki jeb LVAU\_MT000004UV01.personDataAuditSubject. assignedPerson.id.extension);
* **refPersBaseID** (Parasti personas koda OID – var būt vairāki jeb LVAU\_MT000004UV01.personDataAuditSubject. assignedPerson.id.root).

1. Ārstniecības persona (turpmāk tekstā ĀP) atlasa konkrētu recepti. Atlasītās receptes id=1, pieprasījuma id=9.
   1. „ĀP atlasīja konkrētu recepti” – OSB tie ir divi ziņojumi – „in” un „out”:
      1. **In: msgID** = 1**, baseID** = 1.3.6.1.4.1.38760.3.4.1 (ja tas ir GUID) – bet, tas nebūs īstas personas audits, jo te nav personas datu informācijas. Ja būtu gan receptes numurs, gan pacienta numurs, tad būtu: **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”.
      2. **Out:** Pieņemsim, ka „konkrēta recepte” satur personas datus (citādi tad vispār nav pamata rakstīt „personas auditu”). **msgID** = 1, **baseID** = 1.3.6.1.4.1.38760.3.4.1 (ja tas ir GUID), **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”.
2. ĀP atlasa recepšu sarakstu pēc pacienta personas koda (personas kods=15057511226), pieņemot, ka atlasīto recepšu id = [1, 2, 3], pieprasījuma id = 9.
   1. „ĀP atlasīja konkrētu recepti” – OSB tie ir divi ziņojumi - „in” un „out”:
      1. **In: msgID** = 9**, baseID** = 1.3.6.1.4.1.38760.3.4.1 (ja tas ir GUID). Ja būtu gan receptes numurs, gan pacienta numurs tad būtu **refId** = 1, **refBaseId**=1.3.6.1.4.1.38760.3.4.1, **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”.
      2. **Out: msgID** = 9**, baseID** =1.3.6.1.4.1.38760.3.4.1 (ja tas ir GUID).
      3. Katrai atlasītai receptei:
         1. **refID** = 1, **refBaseID**=1.3.6.1.4.1.38760.3.4.1, **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”;
         2. **refID** = 2, **refBaseID**=1.3.6.1.4.1.38760.3.4.1, **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”;
         3. **refID** = 3, **refBaseID**=1.3.6.1.4.1.38760.3.4.1, **refPersID** = „15057511226”, **refPersBaseID**=”1.3.6.1.4.1.38760.3.1.1”.

### Klases izmantošanas piemērs

var msg = string.IsNullOrEmpty(AuditMsg\_tb.Text) ? "Empty Audit Message" : AuditMsg\_tb.Text;

var actionCode = actionCode\_cb.Checked ? ActionTypeKeys.WRITE : "UnrealActionCode";

var reason = reason\_cb.Checked ? ReasonfOfActionOnPersonDataKeys.HEALTH\_CARE\_ADMINISTRATION : "UnrealReason";

if (property\_cb.Checked) {

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

#region ExtendedPropertiesValueStructure attributes

//Ziņojuma id

ExtendedPropertiesValueStructure messageId = new ExtendedPropertiesValueStructure();

messageId.Value = Guid.NewGuid().ToString();

messageId.OID = "1.3.6.1.4.1.38760.3.4.1";

properties.Add(ExtendedPropertyKeys.Id, messageId);

//Ziņojuma sūtītājs

ExtendedPropertiesValueStructure sender\_id = new ExtendedPropertiesValueStructure();

sender\_id.Value = "IP.AUDIT.APP";

sender\_id.OID = "1.3.6.1.4.1.38760.2.3";

properties.Add(ExtendedPropertyKeys.Sender\_id, sender\_id);

//Ziņojuma saņēmējs

ExtendedPropertiesValueStructure receiver\_id = new ExtendedPropertiesValueStructure();

receiver\_id.Value = "IP.AUDIT";

receiver\_id.OID = "1.3.6.1.4.1.38760.2.3";

properties.Add(ExtendedPropertyKeys.Receiver\_id, receiver\_id);

//Ziņojuma izveides datums

ExtendedPropertiesValueStructure creationTime = new ExtendedPropertiesValueStructure();

creationTime.Value = DateTime.Now.ToString();

properties.Add(ExtendedPropertyKeys.CreationTime, creationTime);

#endregion

LVAU\_MT000004UV01SubjectStructure subject = new LVAU\_MT000004UV01SubjectStructure();

subject.documentReference = new LVAU\_MT000004UV01DocumentReference();

#region IS identifier and reference to document in this IS

// informācijas sistēma kur atrodas pamata dokuments ar kuru notiek darbība

subject.documentReference.sourceName = new MCCI\_MT000100UV01\_LV01Device();

II sourceName = new II();

sourceName.Extension = "IP.AUDIT";

sourceName.Root = "1.3.6.1.4.1.38760.2.3";

subject.documentReference.sourceName.id.Add(sourceName);

// dokumenta identifikators šī informācijas sistēmā

subject.documentReference.documentId = new II();

subject.documentReference.documentId.Extension = Guid.NewGuid().ToString();

subject.documentReference.documentId.Root = "1.3.6.1.4.1.38760.3.4.1";

#endregion

subject.documentReference.description = "Kopējais apraksts par auditācijas ierakstu";

for (int i = 0; i < 3; i++) {

LVAU\_MT000004UV01PersonDataAuditSubject auditSubject = new LVAU\_MT000004UV01PersonDataAuditSubject();

#region documentReference

auditSubject.documentReference = new LVAU\_MT000004UV01DocumentReference();

auditSubject.documentReference.description = "Apraksts par personu, par kuru tiek veikts audits";

// sistēma, kur tiek uzturēts referencejamais dokuments

auditSubject.documentReference.sourceName = new MCCI\_MT000100UV01\_LV01Device();

II source = new II();

source.Extension = "IP.AUDIT";

source.Root = "1.3.6.1.4.1.38760.2.3";

auditSubject.documentReference.sourceName.id.Add(source);

// dokumenta identifikators atbilstošā sitēmā

auditSubject.documentReference.documentId = new II();

auditSubject.documentReference.documentId.Extension = Guid.NewGuid().ToString();

auditSubject.documentReference.documentId.Root = "1.3.6.1.4.1.38760.3.4.1";

#endregion

#region assignedPerson

// saistīta persona - personas datu audita objekts

auditSubject.assignedPerson = new COCT\_MT090100UV01AssignedPerson();

auditSubject.assignedPerson.assignedPerson = new COCT\_MT090100UV01Person();

EN en = new EN("Surname " + i.ToString(), "Name " + i.ToString());

II ii = new II("11111-1111" + i.ToString(), "1.3.6.1.4.1.38760.3.1.1");

auditSubject.assignedPerson.assignedPerson.name.Add(en);

auditSubject.assignedPerson.id.Add(ii);

#endregion

subject.personDataAuditSubject.Add(auditSubject);

}

properties.Add("PersonDataAuditSubject", subject);

mlu.WriteAudit(actionCode, reason, 0, msg, properties);

}

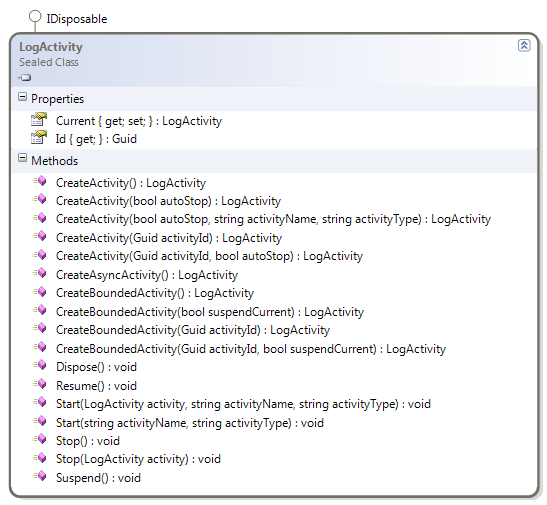
else {

mlu.WriteAudit(actionCode, reason, 0, msg);

}

## Aktivitāšu palīgklase

Aktivitāšu palīgklasē iekļautas metodes, kas ļauj apstrādāt



5.attēls Aktivitāšu palīgklasē iekļautās metodes un īpašības

### Metodes

#### Metode „CreateActivity”

metodes apraksts:

public LogActivity CreateActivity()

Izveido jaunu aktivitāti *LogActivity.*

public LogActivity CreateActivity(bool autoStop)

Izveido jaunu aktivitāti *LogActivity* ar automātiskas apturēšanas iespējām.

public LogActivity CreateActivity(bool autoStop, string activityName, string activityType)

Izveido jaunu aktivitāti *LogActivity* ar nosaukumu, tipu un automātiskas apturēšanas iespējām.

public LogActivity CreateActivity(Guid activityId)

Izveido jaunu aktivitāti *LogActivity* ar unikālu identifikatoru.

public LogActivity CreateActivity(Guid activityId, bool autoStop)

Izveido jaunu aktivitāti *LogActivity* ar unikālu identifikatoru un automātiskas apturēšanas iespējām.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| autoStop | norāda vai aktivitāte tiks automātiski apturēta |
| activityName | aktivitātes nosaukums |
| activityType | aktivitātes tips |
| activityId | aktivitātes identifikators |

#### Metode „CreateAsyncActivity”

metodes apraksts:

public LogActivity CreateAsyncActivity()

Izveido jaunu asinhrono aktivitāti *LogActivity.*

#### Metode „CreateBoundedActivity”

metodes apraksts:

public LogActivity CreateBoundedActivity()

Izveido jaunu saistīto aktivitāti *LogActivity.*

public LogActivity CreateBoundedActivity(bool suspendCurrent)

Izveido jaunu saistīto aktivitāti *LogActivity* ar tekošās aktivitātes apturēšanas iespējām.

public LogActivity CreateBoundedActivity(Guid activityId)

Izveido jaunu saistīto aktivitāti *LogActivity* ar unikālu identifikatoru.

public LogActivity CreateBoundedActivity(Guid activityId, bool suspendCurrent)

Izveido jaunu saistīto aktivitāti *LogActivity* ar unikālu identifikatoru un tekošās aktivitātes apturēšanas iespējām.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| suspendCurrent | norāda vai tekošā aktivitāte tiks apturēta |
| activityId | aktivitātes identifikators |

#### Metode „Dispose”

metodes apraksts:

public void Dispose()

Atbrīvo *LogActivity* klases eksemplāra aizņemtos resursus.

#### Metode „Suspend”

metodes apraksts:

public void Suspend()

Aiztur tekošās aktivitātes darbību.

#### Metode „Resume”

metodes apraksts:

public void Resume()

Atjauno tekošās aktivitātes darbību.

#### Metode „Stop”

metodes apraksts:

public void Stop()

Aptur tekošās aktivitātes darbību.

public void Stop(LogActivity activity)

Aptur norādītās aktivitātes darbību.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| activity | aktivitāte |

#### Metode „Start”

metodes apraksts:

public void Start(string activityName, string activityType)

Uzsāk aktivitātes darbību ar uzdoto nosaukumu un tipu.

public void Stop(LogActivity activity, string activityName, string activityType)

Uzsāk aktivitātes darbību ar uzdoto nosaukumu un tipu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| activity | aktivitāte |
| activityName | aktivitātes nosaukums |
| activityType | aktivitātes tips |

### Klases izmantošanas piemērs

public void DoSomesing() {

// Trasēšana izmantojot konstruktoru un „using” bloku.

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

la.Start(„MyActivity”, „SomeType”);

...

}

}

## Klašu izmantošanas piemērs

Katrā veidojamā lietojumā jābūt savai klasei, kas nodrošinātu nepieciešamo žurnalēšanu, auditāciju un trasēšanu. Programmas modulim izveidojam statisko klasi:

// ----------------------------------------------------------------------------

// <copyright file="DiagnosticTools.cs" company="ABC software">

// Copyright © ABC SOFTWARE. All rights reserved.

// The source code or its parts to use, reproduce, transfer, copy or

// keep in an electronic form only from written agreement ABC SOFTWARE.

// </copyright>

// ----------------------------------------------------------------------------

namespace Utils {

using System;

using System.Collections.Generic;

using System.Diagnostics;

using Diagnostic;

using Diagnostic.ExtraInformation;

/// <summary>

/// Klase implemetē metodes, kuras ir nepieciešamas lietojuma izpildes diagnostikai

/// </summary>

internal static class DiagnosticTools {

private const int DefaultPriority = -1;

private const int DefaultEventId = -1;

private static LogUtility logUtil;

private static ExceptionUtility exceptionUtil;

/// <summary>

/// Initializes static members of the <see cref="DiagnosticTools"/> class.

/// </summary>

static DiagnosticTools() {

logUtil = new LogUtility(System.Reflection.Assembly.GetCallingAssembly().FullName);

exceptionUtil = new ExceptionUtility(logUtil);

}

/// <summary>

/// Logošionas kategorijas

/// </summary>

public enum LogCategory {

/// <summary>

/// Kategorija ir paredzēta darbam ar <see cref="StsHelperObject"/>

/// </summary>

StsHelperObject = 1,

/// <summary>

/// Kategorija ir paredzēta darbam ar <see cref="DataRepositories.SamlTokenStorage"/>

/// </summary>

SamlTokenStorage = 2,

/// <summary>

/// Kategorija ir paredzēta darbam ar <see cref="Proxies.ReqProxy"/>

/// </summary>

RequestServiceProxy = 4

}

/// <summary>

/// Notifikācijas kategorijas

/// </summary>

[FlagsAttribute()]

public enum NotificationCategories {

/// <summary>

/// Notificējam administratorus

/// </summary>

Administrators,

/// <summary>

/// Notificējam moderatorus

/// </summary>

Moderators,

/// <summary>

/// Notificējam lietotājus

/// </summary>

Users,

}

/// <summary>

/// <see cref="ExceptionUtility"/> inicializēta instance

/// </summary>

public static ExceptionUtility ExceptionUtility {

get { return DiagnosticTools.exceptionUtil; }

}

/// <summary>

/// <see cref="LogUtility"/> inicializēta instance

/// </summary>

public static LogUtility LogUtility {

get { return DiagnosticTools.logUtil; }

}

/// <summary>

/// Metode nodrošina loga rakstīšanu

/// </summary>

/// <param name="msg">Ziņojums, kuru jāraksta logā</param>

/// <param name="category">Kategorijas veids</param>

/// <param name="eventId">Notikuma identifikators</param>

/// <param name="severity">Trasēšanas veids</param>

public static void WriteLog(string msg, LogCategory category, int eventId, TraceEventType severity) {

WriteLog(msg, category, eventId, severity, null);

}

/// <summary>

/// Metode nodrošina loga rakstīšanu

/// </summary>

/// <param name="msg">Ziņojums, kuru jāraksta logā</param>

/// <param name="category">Kategorijas veids</param>

/// <param name="eventId">Notikuma identifikators</param>

/// <param name="severity">Trasēšanas veids</param>

/// <param name="properties">Papildīpašības, kuras tiks pievienotas loga pierakstam</param>

public static void WriteLog(string msg, LogCategory category, int eventId, TraceEventType severity, IDictionary<string, object> properties) {

if (properties == null) {

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

}

// Ieveitojam informāciju par lietotāju

ManagedSecurityContextInformationProvider informationHelper =

new ManagedSecurityContextInformationProvider();

informationHelper.PopulateDictionary(properties);

logUtil.Write(msg, category.ToString(), DefaultPriority, eventId, severity, properties);

}

/// <summary>

/// Metode nodrošina notifikāciju

/// </summary>

/// <param name="message">Ziņojums notifikācjai</param>

/// <param name="categories">Kategorijas uz kuriem javeic notifikāciju</param>

public static void Notify(string message, NotificationCategories categories) {

ICollection<string> categoryCollection = new List<string>();

foreach (NotificationCategories notificationCategory in Enum.GetValues(typeof(NotificationCategories))) {

if ((categories & notificationCategory) == notificationCategory) {

categoryCollection.Add(notificationCategory.ToString());

}

}

IDictionary<string, object> properties = new Dictionary<string, object>();

// Ieveitojam informāciju par lietotāju

ManagedSecurityContextInformationProvider informationHelper =

new ManagedSecurityContextInformationProvider();

informationHelper.PopulateDictionary(properties);

logUtil.Write(message, categoryCollection, DefaultPriority, DefaultEventId, TraceEventType.Information, properties);

}

}

}

# Klašu izmantošanas piemēri

Bibliotēku „Diagnostic.dll” un „Enterprise Library 4.0.0.0” uzstādīšana un konfigurēšana atbilstoši izmantotajai žurnalēšanas palīgklasei.

## Vispārējie norādījumi

Tālāko instrukciju pamatā ir izmantots lietojums EnoughPI ([4]). Tas aprēķina konstanti pi līdz uzdotajai precizitātei. Lietotājs ievada vēlamo precizitāti lietojot NumericUpDown vadīklu un nospiež Calculate pogu. Aprēķinu var apturēt, nospiežot pogu Cancel.

Pirms tālāku darbību veikšanas nepieciešams izpildīt sekojošus priekšnosacījumus:

1. Izveidot jaunu risinājuma EnoughPI.sln ([4]) kopiju, kurā tiks veiktas visas turpmākās darbības;
2. Atvērt risinājumu EnoughPI.sln;
3. Atvērt projektu EnoughPI un iezīmēt katalogu References, izpildīt konteksta izvēlnes komandu Add Reference un pievienot projektam Diagnostic.dll bibliotēkas pēdējo versiju izmantojot nuget (https://epaktest.abcsoftware.lv:444/NuGet/nuget).

## LogUtility

Žurnalēšanas funkcionalitātes piesaistīšana esošam projektam izmantojot LogUtility klasi. Izpildītais risinājums uz kā balstīta šī instrukcija atrodas 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 *diagnosticConfiguration*, kas norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* 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

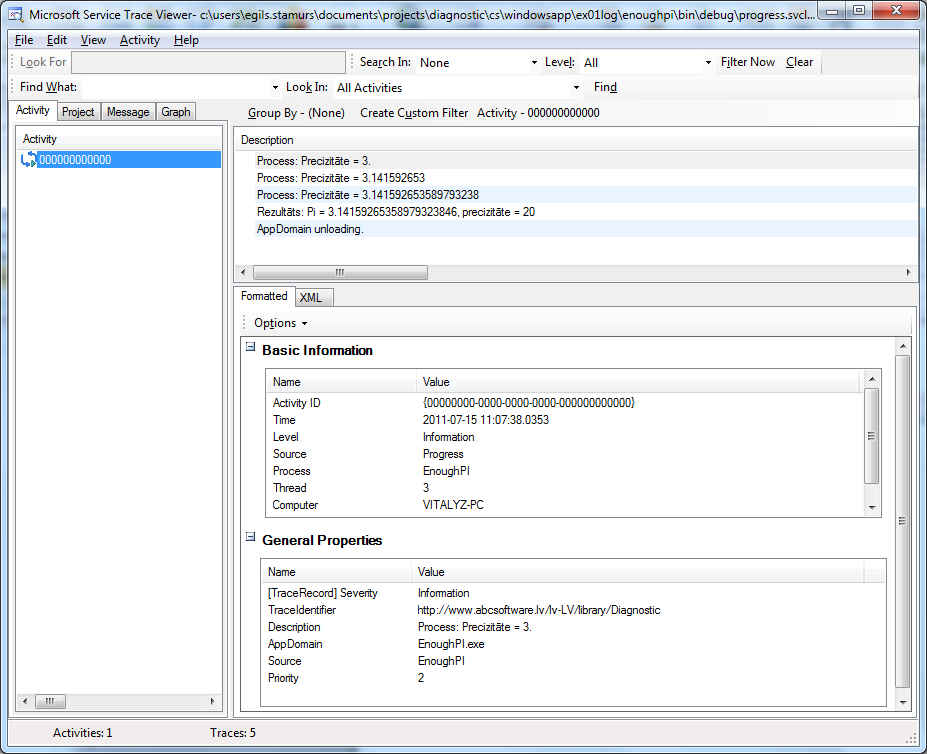
{

public const string General = "General";

public const string Progress = "Progress";

}

1. Iegūtais rezultāts apkopojot General un Progress žurnālus (skat. 6.attēlu).



6.attēls. Žurnalēšanas rezultāti

## Personalizētais audits

Personalizēta audita funkcionalitātes piesaistīšana esošam projektam izmantojot EvesDiagnostic klasi. Izpildītais risinājums uz kā balstīta šī instrukcija atrodas katalogā *„~\Diagnostic\CS\WindowsApp\* *ex06PersonalizedAudit”*.

1. Pievienot atsauci uz failu Abc.Diagnostic.Extension.dll;
2. Iezīmēt projekta failu Calc\Calculator.cs un izpildīt konteksta izvēlnes komandu View Code;
3. Pievienot vārdtelpu Abc.Diagnostic.Extension faila augšdaļā;

using Abc.Diagnostic.Extension;

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 *diagnosticConfiguration*, kas norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* 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="Audit.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="AuditLog"

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="Audit" switchValue="All">

<listeners>

<add name="AuditLog"/>

</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 EvesDiagnstic klases eksemplāru, klases Calculator ietvaros, norādot atribūtu sourceName, kas tiks lietots ziņojumu avota identificēšanai, šajā gadījumā projekta nosaukums *EnoughPI*;

EvesDiagnostic audit = new EvesDiagnostic("EnoughPI");

1. Nepieciešamajās vietās pievienot projekta kodam audita rakstīšanas metodes WriteAudit.

this.audit.WriteAudit(„auditType”, „reason”, 1, „message”);

## 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 *diagnosticConfiguration*, kas norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* 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;

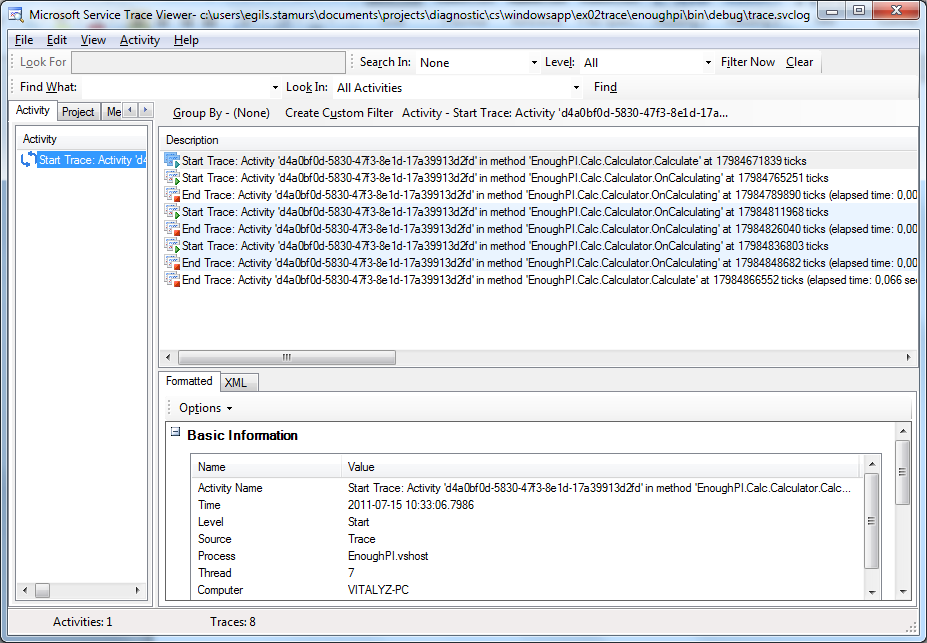
public struct Category

{

public const string Trace = "Trace";

}

1. Iegūtais rezultāts trasējot metodes Calculate un OnCalculating (skat. 7.attēlu).



7.attēls. 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 *diagnosticConfiguration*, kas norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* 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;

ExceptionUtility exceptionWriter = new ExceptionUtility();

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

protected void OnCalculatorException(CalculatorExceptionEventArgs args)

{

// TODO: Log exception

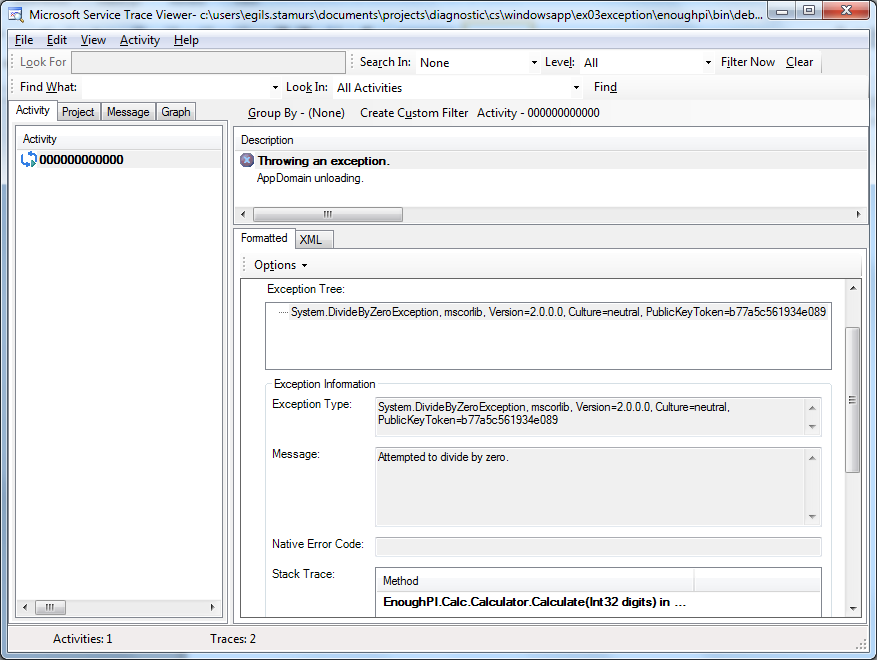
exceptionWriter.ThrowHelperError(args.Exception);

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 (skat. 8.attēlu).



8.attēls. 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 *diagnosticConfiguration*, kas norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* 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"/>

<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 identifikatoriem name = *„General”* un „Activity”. Klasēm piesaistīt vismaz vienu listeners failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora, vai definējot jaunu failu.

<sources>

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

<listeners>

<add name="GeneralLog"/>

</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. 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];

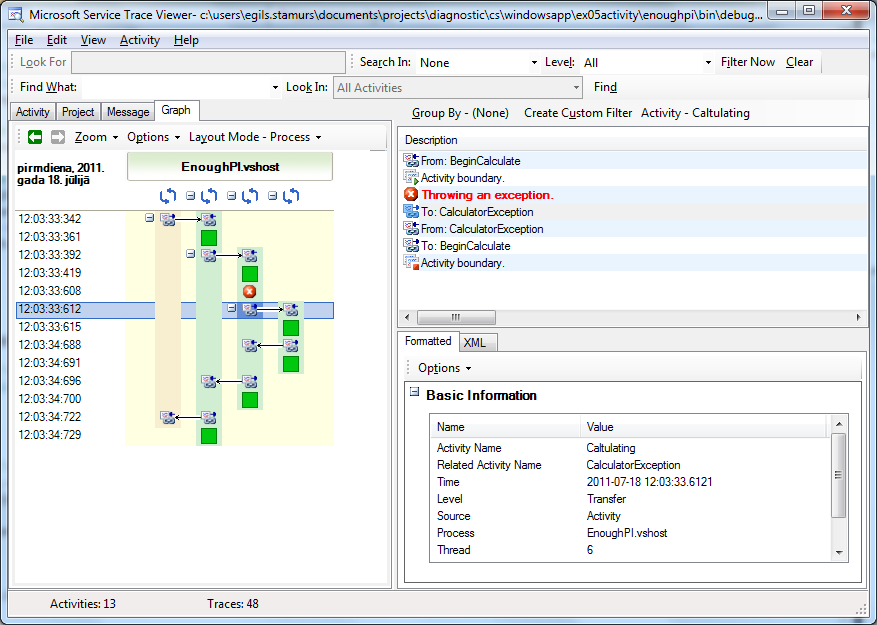
dlg.EndInvoke(ar);

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 (skat. 9.attēlu).



9.attēls. 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);

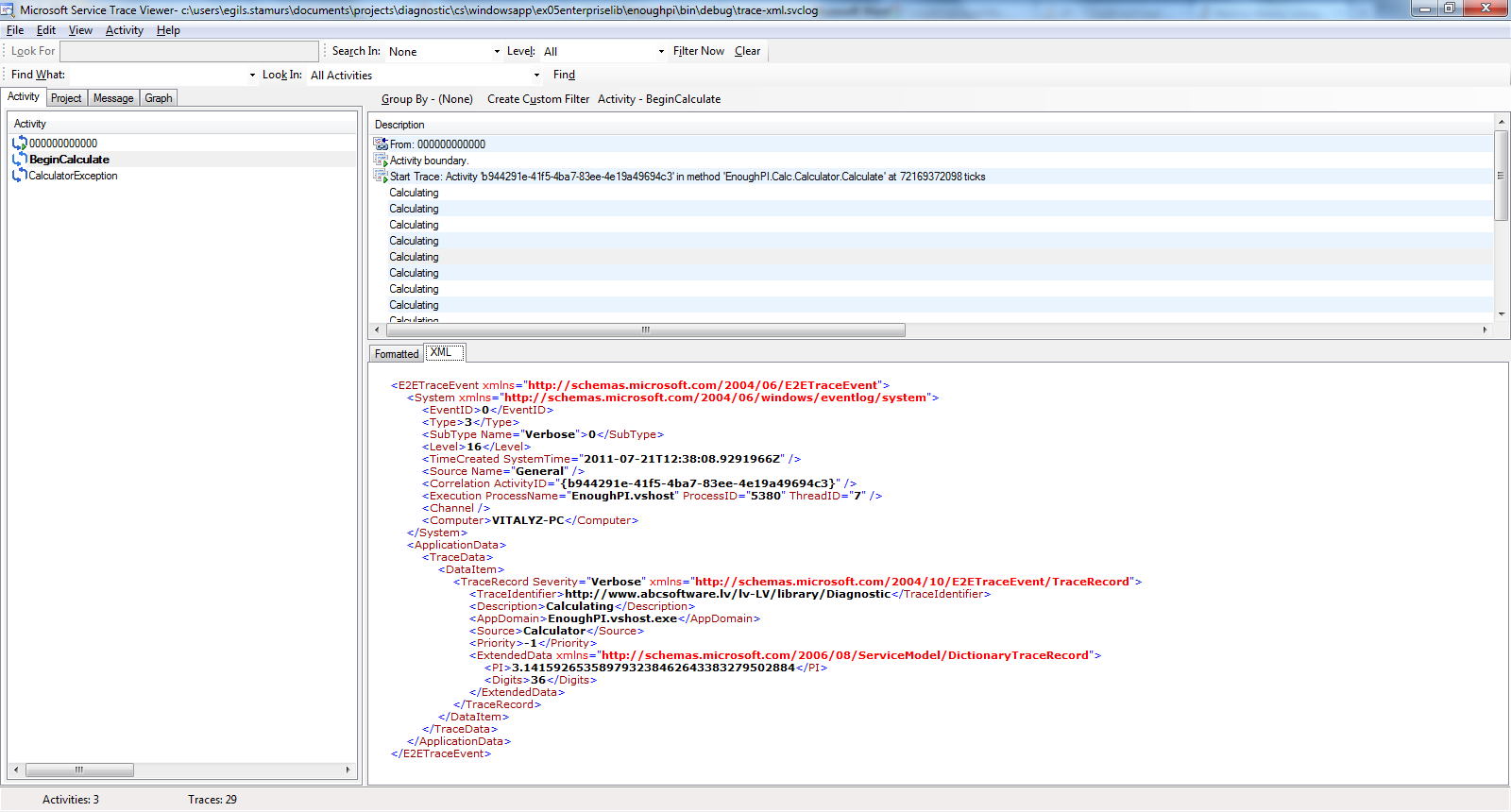
1. To pašu rezultātu iespējams sasniegt neveidojot atsevišķu klasi Dictionary tipa mainīgā aizpildīšanai, bet gan veicot sekojošas darbības:
   * Definēt Dictionary tipa mainīgo;
   * Pievienot mainīgajam nepieciešamās īpašības un to vērtības izsaucot metodi Add**;**
   * Izsaukt LogUtility metodi Write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

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(skat. 10.attēlu).



10.attēls. ExtraInformationProvider pielietošanas rezultāti

## Enterprise Library 5.0.0.0

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.

Plašākas Enterprise Library (5.0.0.0) konfigurēšanas iespējas var aplūkot sekojošā dokumentā - E-veselības integrācijas platformas infrastruktūra: auditācijas un izpildlaika pierakstu žurnāls (2.sējums). Administratoru rokasgrāmata (NVD.IP.VISS\_2010.STD.Apakšprojekta abreviatūra).

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”*.

## WCF Servisa un klienta trasēšana

Šajā piemērā tiks aprakstītas darbības, kas jāveic, lai piesaistītu esošam WCF servisam un klientam aktivitāšu trasēšanas funkcionalitāti izmantojot iebūvētās žurnalēšanas un trasēšanas funkcijas, kā arī papildus ziņojumu žurnalēšanu ar Diagnostic.dll bibliotēku.

Izpildītais risinājuma piemērs uz kā balstīta šī instrukcija atrodama katalogā *„~\Diagnostic\CS WebService\ex01WCFActivity”.*

### Klienta konfigurācijas datne

Modificēt klienta konfigurācijas datni app.config, izmantojot WCF Configuration Editor vai manuāli pievienojot sekojošus atribūtus:

* 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="app\_tracelog.svclog"

type="System.Diagnostics.XmlWriterTraceListener, System,

Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="ServiceModelTraceListener"

traceOutputOptions="Timestamp">

<filter type="" />

</add>

</sharedListeners>

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

<sources>

<source name="System.ServiceModel"

switchValue="Information,ActivityTracing"

propagateActivity="true">

<listeners>

<add name="ServiceModelTraceListener">

<filter type="" />

</add>

</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>

* Sadaļu system.serviceModel, kas satur sadaļas diagnostic, bindings un client. Sadaļā diagnostic iespējot vai atspējot ziņojumu žurnalēšanas iespējas;

<system.serviceModel>

<diagnostics>

<messageLogging logEntireMessage="true"

logMessagesAtTransportLevel="true" />

</diagnostics>

* Sadaļā bindings uzstādīt servisa piesaistes konfigurāciju;

<bindings>

<basicHttpBinding>

<binding name="BasicHttpBinding\_IService1"

closeTimeout="00:01:00" openTimeout="00:01:00"

receiveTimeout="00:10:00" sendTimeout="00:01:00"

allowCookies="false" bypassProxyOnLocal="false"

hostNameComparisonMode="StrongWildcard"

maxBufferSize="65536" maxBufferPoolSize="524288"

maxReceivedMessageSize="65536"

messageEncoding="Text" textEncoding="utf-8"

transferMode="Buffered" useDefaultWebProxy="true">

<readerQuotas maxDepth="32" maxStringContentLength="8192"

maxArrayLength="16384" maxBytesPerRead="4096"

maxNameTableCharCount="16384" />

<security mode="None">

<transport clientCredentialType="None"

proxyCredentialType="None"

realm="" />

<message clientCredentialType="UserName"

algorithmSuite="Default" />

</security>

</binding>

</basicHttpBinding>

</bindings>

* Sadaļā client piesaistīt servisu – adress un norādīt piesaistes tipu – binding un tās  
  konfigurāciju – bindingConfiguration.

<client>

<endpoint address="http://localhost:64775/Service1.svc"

binding="basicHttpBinding"

bindingConfiguration="BasicHttpBinding\_IService1"

contract="ServiceRef.IService1"

name="BasicHttpBinding\_IService1" />

</client>

</system.serviceModel>

### Servisa konfigurācijas datne

Modificēt servisa konfigurācijas datni web.config, izmantojot WCF Configuration Editor vai manuāli pievienojot sekojošus atribūtus:

* 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="Web\_tracelog.svclog"

type="System.Diagnostics.XmlWriterTraceListener, System,

Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="ServiceModelTraceListener"

traceOutputOptions="Timestamp">

<filter type="" />

</add>

</sharedListeners>

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

<sources>

<source name="System.ServiceModel"

switchValue="Information,ActivityTracing"

propagateActivity="true">

<listeners>

<add name="ServiceModelTraceListener">

<filter type="" />

</add>

</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>

* Sadaļu system.serviceModel, kas satur sadaļas diagnostic, services un behaviors. Sadaļā diagnostic iespējot vai atspējot ziņojumu žurnalēšanas iespējas;

<system.serviceModel>

<diagnostics>

<messageLogging logEntireMessage="true"

logMessagesAtTransportLevel="true" />

</diagnostics>

* Sadaļā services uzstādīt servisa konfigurāciju;

<services>

<service behaviorConfiguration="WcfService.Service1Behavior"

name="WcfService.Service1">

<endpoint address="" binding="basicHttpBinding"

bindingConfiguration=""

contract="WcfService.IService1">

<identity>

<dns value="localhost" />

</identity>

</endpoint>

<endpoint address="mex" binding="mexHttpBinding"

contract="IMetadataExchange" />

</service>

</services>

* Sadaļā behaviors uzstādīt servisa konfigurāciju.

<behaviors>

<serviceBehaviors>

<behavior name="WcfService.Service1Behavior">

<serviceMetadata httpGetEnabled="true"/>

<serviceDebug includeExceptionDetailInFaults="false"/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

### Žurnalēšana ar Diagnostic.dll

Trasējamības uzlabošanai tiek piesaistīta papildus žurnalēšana gan servisa, gan klienta pusē izmatojot LogUtility;

* Papildināt konfigurācijas failus web.config un app.config;

<!--for Diagnostic.dll logUtility-->

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

<listeners>

<add name="ServiceModelTraceListener"></add>

</listeners>

</source>

* Web.app sadaļā *diagnosticConfiguration* norāda izmantojamo žurnāla rakstītāju.

<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic, Version=1.0.0.0" />

* Pievienot klienta un servisa risinājumiem Diagnostic.dll bibliotēku un atbilstošas vārdtelpas;
* Izveidot žurnalēšanas metodes nepieciešamajās vietās gan servisa gan klienta pusē;

LogUtility logWriter = new LogUtility("WCFService");

public string GetData(int value)

{

logWriter.Write("Received value: " + value.ToString(), "Activity");

System.Threading.Thread.Sleep(value);

return string.Format("You entered: {0}", value);

}

logWriter.Write("Entered: " + i.ToString(), "Activity");

rez = c.GetData(i);

logWriter.Write("Response: " + rez, "Activity");

OutputTextBox.Text = rez;

### Sinhrons servisa izsaukums

Sinhroni servisa izsaukumi var tikt automātiski apstrādāti bez papildus kodēšanas. Bet, lai nodrošinātu korektu aktivitāšu atspoguļojumu tiek izmantota LogActivity klases funkcionalitāte (skat. 11.attēlu).

private void SyncGetDataButton\_Click(object sender, EventArgs e)

{

int i;

string rez;

LogActivity la = LogActivity.CreateBoundedActivity();

la.Start("SyncCall", "FromClient");

using (ServiceRef.Service1Client c = new ServiceRef.Service1Client())

{

if (!int.TryParse(InputTextBox.Text, out i))

{

OutputTextBox.Text = "Ievadīt skaitli!";

}

else

{

logWriter.Write("Entered: " + i.ToString(), "Activity");

rez = c.GetData(i);

logWriter.Write("Response: " + rez, "Activity");

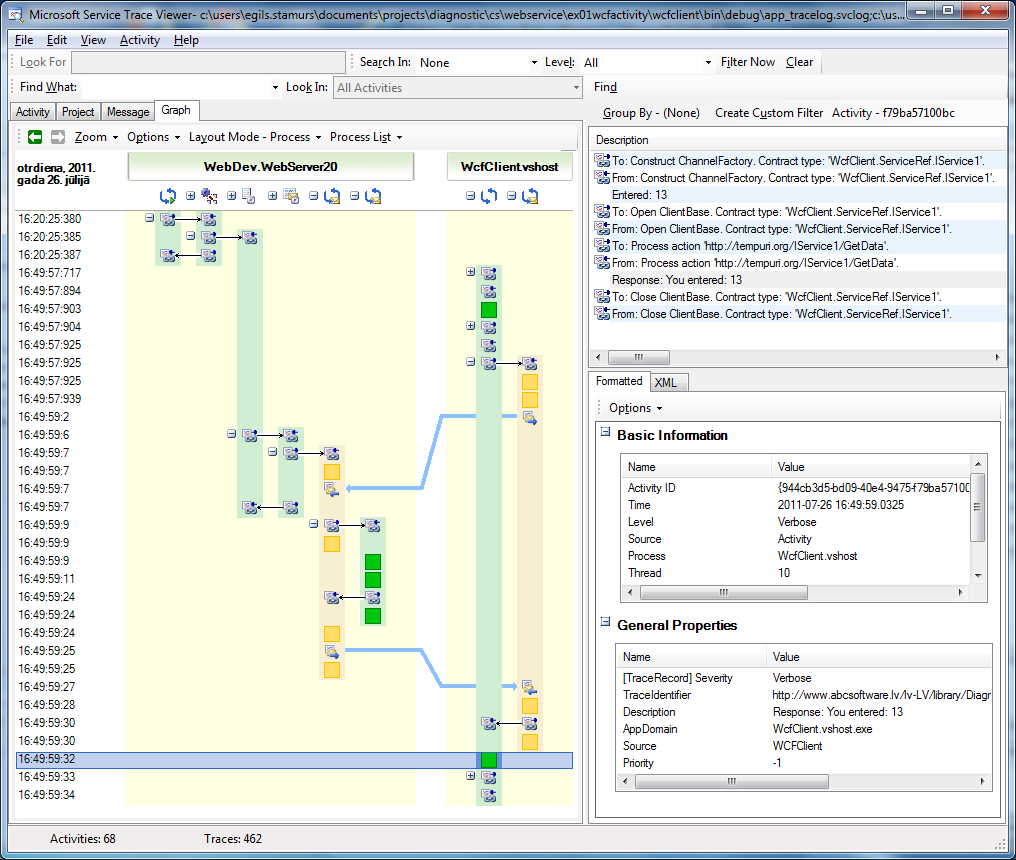
OutputTextBox.Text = rez;

}

}

la.Stop();

}



11.attēls. Sinhrona izsaukuma rezultāts

### Asinhrons servisa izsaukums

Asinhrona servisa izsaukuma gadījumā nepieciešams izmantot LogActivity klases funkcionalitāti, lai nodrošinātu korektas ziņojumu identifikatoru vērtības izsaucot servisu un atiežoties no tā (skat. 12.attēlu). Metode AsyncGetDataButton\_Click nodrošina asinhronu servisa izsaukumu:

private void AsyncGetDataButton\_Click(object sender, EventArgs e)

{

int i;

LogActivity l = LogActivity.CreateAsyncActivity();

using (LogActivity la = LogActivity.CreateBoundedActivity(l.Id))

{

la.Start("AsyncCall", "FromClient");

if (!int.TryParse(InputTextBox.Text, out i))

{

OutputTextBox.Text = "Ievadīt skaitli!";

}

else

{

logWriter.Write("Async Entered: " + i.ToString(), "Activity");

ServiceRef.Service1Client c = new ServiceRef.Service1Client();

l.Suspend();

c.GetDataAsync(Convert.ToInt32(this.InputTextBox.Text), l);

c.GetDataCompleted += new EventHandler<ServiceRef.

GetDataCompletedEventArgs>(c\_GetDataCompleted);

}

}

}

Metode c\_GetDataCompleted nodrošina datu izvadi pēc izsaukuma izpildīšanas;

private void c\_GetDataCompleted(object sender, ServiceRef.GetDataCompletedEventArgs e)

{

LogActivity l = e.UserState as LogActivity;

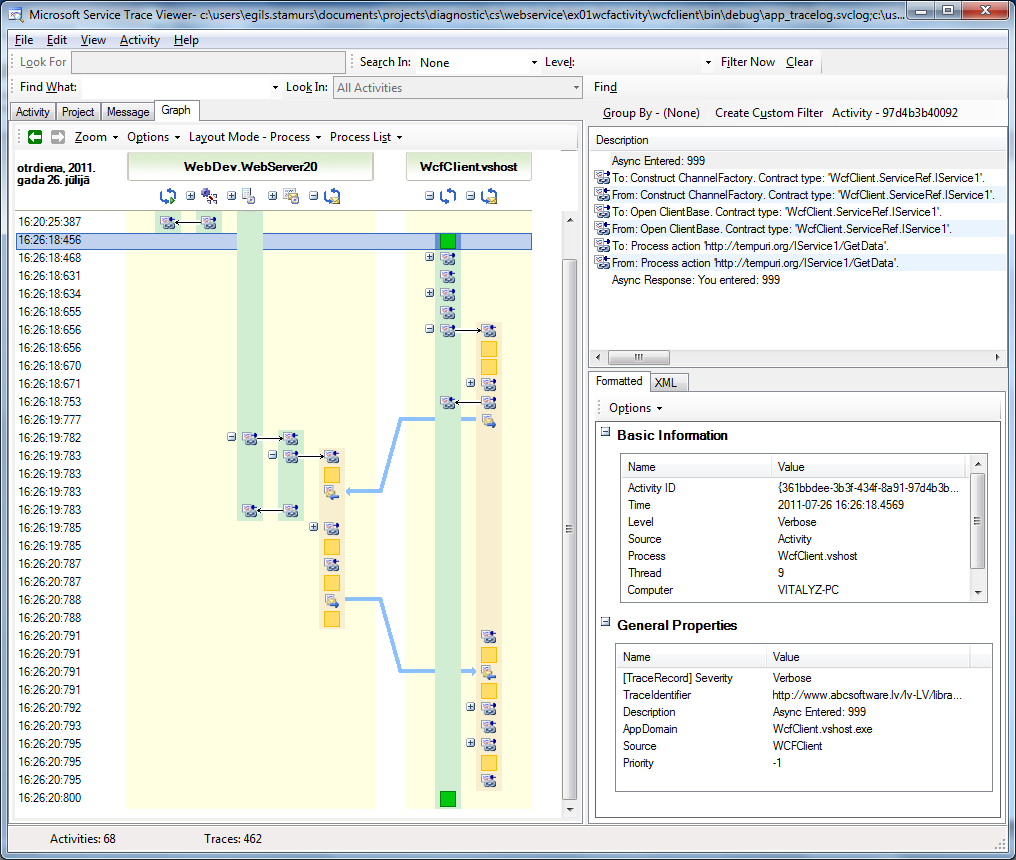
l.Resume();

this.OutputTextBox.Text = e.Result;

logWriter.Write("Async Response: " + e.Result, "Activity", -1, -1, System.Diagnostics.TraceEventType.Verbose, l.Id);

l.Stop();

}



12.attēls. Asinhrona izsaukuma rezultāts