Valsts reģionālās attīstības aģentūra



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

Par Valsts informācijas sistēmu savietotāja, Latvijas valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana

3.daļa "VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju"

VISS sistēmas žurnāls

Koplietojuma bibliotēku apraksts

VRAA-6\_15\_11\_58-VISS\_2010-VISS\_ZUR-KBA

24.09.2013. versija 1.03

Rīgā 2013

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

Dokumenta identifikācija

|  |  |
| --- | --- |
| Dokumenta ID: | VRAA-6\_15\_11\_58-VISS\_2010-VISS\_ZUR-KBA-V1.03-24.09.2013. |
| Dokumenta nosaukums: | Par Valsts informācijas sistēmu savietotāja, Latvijas valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana.  3.daļa "VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju".  VISS sistēmas žurnāls.  Koplietojuma bibliotēku apraksts. |
| Dokumenta kods: | VRAA-6\_15\_11\_58-VISS\_2010-VISS\_ZUR-KBA |
| Versija: | Versija 1.03, Laidiens 24.09.2013. (saīsināti V1.03 24.09.2013.) |

Saskaņojumi

|  |  |  |  |
| --- | --- | --- | --- |
| Organizācija | Vārds, uzvārds, amats | Datums | Paraksts |
| Valsts reģionālās attīstības aģentūra | Atbildīgā persona no Pasūtītāja puses |  |  |
| SIA "ABC software" | J.Korņijenko, projekta vadītājs par tehniskiem jautājumiem no Izpildītāja puses | 24.09.2013. |  |
| SIA "ABC software" | M.Pētersons projekta vadītājs par administratīviem jautājumiem no Izpildītāja puses | 24.09.2013. |  |
| SIA "ABC software" | E.Blumberga, projekta kvalitātes kontroles vadītāja | 24.09.2013. |  |

Izmaiņu vēsture

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versija | Datums | Apraksts | Organizācija | Autors |
| 1.00 | 14.09.2011. | Izveidota dokumenta sākotnējā versija | SIA „ABC software” | E.Stāmurs |
| 1.01 | 27.11.2012. | Pievienots paplašinājumu klases apraksts. Izlabotas metodes SendUserNotification un SendGroupNotification (userPersonCode uz userCode) | SIA „ABC software” | E.Stāmurs, J.Korņijenko |
| 1.02 | 07.03.2013. | Precizēts auditācijas un paziņojumu izsūtīšanas funkcionalitātes apraksts. | SIA „ABC software” | J.Korņijenko |
| 1.03 | 24.09.2013. | Precizēts auditācijas un paziņojumu izsūtīšanas funkcionalitātes apraksts. | SIA „ABC software” | J.Korņijenko |

**Satura rādītājs**

[Attēlu saraksts 6](#_Toc383009081)

[1. Ievads 7](#_Toc383009082)

[1.1. Dokumenta nolūks 7](#_Toc383009083)

[1.2. Termini un pieņemtie apzīmējumi 7](#_Toc383009084)

[1.3. Saistība ar citiem dokumentiem 7](#_Toc383009085)

[2. Palīgklases 8](#_Toc383009086)

[2.1. Logošanas palīgklase 8](#_Toc383009087)

[2.1.1. Konstruktori 8](#_Toc383009088)

[2.1.1.1. Konstruktors „LogUtility” 8](#_Toc383009089)

[2.1.2. Īpašības 8](#_Toc383009090)

[2.1.2.1. „LoggingEnabled” īpašība 8](#_Toc383009091)

[2.1.3. Metodes 8](#_Toc383009092)

[2.1.3.1. Metode „Write” 8](#_Toc383009093)

[2.1.4. Piemērs 9](#_Toc383009094)

[2.2. Izņēmumu palīgklase 9](#_Toc383009095)

[2.2.1. Konstruktori 10](#_Toc383009096)

[2.2.1.1. Konstruktors „ExceptionUtility” 10](#_Toc383009097)

[2.2.2. Metodes 10](#_Toc383009098)

[2.2.2.1. Metode „ThrowHelper” 10](#_Toc383009099)

[2.2.2.2. Metode „ThrowHelperWarning” 10](#_Toc383009100)

[2.2.2.3. Metode „ThrowHelperError” 11](#_Toc383009101)

[2.2.2.4. Metode „ThrowHelperCritical” 11](#_Toc383009102)

[2.2.2.5. Metode „ThrowHelperFatal” 11](#_Toc383009103)

[2.2.2.6. Metode „IsFatal” 11](#_Toc383009104)

[2.2.2.7. Metode „ThrowHelperArgument” 11](#_Toc383009105)

[2.2.2.8. Metode „ThrowHelperArgument” 12](#_Toc383009106)

[2.2.2.9. Metode „ThrowHelperArgumentNull” 12](#_Toc383009107)

[2.2.2.10. Metode „ThrowHelperArgumentNull” 12](#_Toc383009108)

[2.2.2.11. Metode „UseActivityId” 12](#_Toc383009109)

[2.2.2.12. Metode „ClearActivityId” 12](#_Toc383009110)

[2.2.2.13. Metode „TraceHandeledException” 13](#_Toc383009111)

[2.3. Trasēšanas palīgklase 13](#_Toc383009112)

[2.3.1. Konstruktori 13](#_Toc383009113)

[2.3.1.1. Konstruktors „TraceUtilty” 13](#_Toc383009114)

[2.3.2. Metodes 14](#_Toc383009115)

[2.3.2.1. Metode „StartTrace” 14](#_Toc383009116)

[2.3.3. Klases izmantošanas piemērs 14](#_Toc383009117)

[2.4. Paplašinājumu palīgklase 14](#_Toc383009118)

[2.4.1. Paziņojumi (Notifikācijas) 14](#_Toc383009119)

[2.4.1.1. Metodes 17](#_Toc383009120)

[2.4.2. Audits 19](#_Toc383009121)

[2.4.2.1. Metodes 19](#_Toc383009122)

[2.4.2.2. Parametru vērtību specificēšana 20](#_Toc383009123)

[3. Klašu izmantošanas piemēri 22](#_Toc383009124)

[3.1. Vispārējie norādījumi 22](#_Toc383009125)

[3.2. LogUtility 22](#_Toc383009126)

[3.3. TraceUtility 24](#_Toc383009127)

[3.4. ExceptionUtility 26](#_Toc383009128)

[3.5. LogActivity 27](#_Toc383009129)

[3.6. ExtraInformationProvider 29](#_Toc383009130)

[3.7. Enterprise Library 4.0.0.0 31](#_Toc383009131)

[3.8. WCF Servisa un klienta trasēšana 34](#_Toc383009132)

[3.8.1. Klienta konfigurācijas datne 34](#_Toc383009133)

[3.8.2. Servisa konfigurācijas datne 35](#_Toc383009134)

[3.8.3. Žurnalēšana ar Diagnostic.dll 36](#_Toc383009135)

[3.8.4. Sinhrons servisa izsaukums 37](#_Toc383009136)

[3.8.5. Asinhrons servisa izsaukums 38](#_Toc383009137)

[3.9. Paplašinājumu palīgklase 39](#_Toc383009138)

[3.9.1. Notifikācijas servisa konfigurācija. 41](#_Toc383009139)

# Attēlu saraksts

[1.attēls. Logošanas palīgklasē iekļautās metodes 8](#_Toc383009068)

[2.attēls. Izņēmumu palīgklasē iekļautās metodes 10](#_Toc383009069)

[3.attēls. Trasēšanas palīgklasē iekļautās metodes 13](#_Toc383009070)

[4.attēls. Paplašinājuma palīgklasē iekļautās metodes 14](#_Toc383009071)

[5.attēls. Notifikāciju nosūtīšanas struktūra 15](#_Toc383009072)

[6.attēls. Žurnalēšanas rezultāti 24](#_Toc383009073)

[7.attēls. Trasēšanas rezultāti 25](#_Toc383009074)

[8.attēls. Izņēmumu žurnalēšanas rezultāti 27](#_Toc383009075)

[9.attēls. Aktivitāšu žurnalēšanas rezultāti 29](#_Toc383009076)

[10.attēls. ExtraInformationProvider pielietošanas rezultāti 31](#_Toc383009077)

[11.attēls. Sinhrona izsaukuma rezultāts 38](#_Toc383009078)

[12.attēls. Asinhrona izsaukuma rezultāts 39](#_Toc383009079)

[13.attēls. Ziņojumu un audita rezultāti 41](#_Toc383009080)

# 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 [2].

## Saistība ar citiem dokumentiem

Dokuments ir izstrādāts, balstoties uz šādiem dokumentiem:

1. IVIS izstrādes vadlīnijas (URN:100000:DOC-FR-VNP).
2. IVIS „Terminu un saīsinājumu indekss.” (EPS-31/2005-IVIS-OLIMPS-TSI-V0.03 07.03.2006.).
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”.

# Palīgklases

Bibliotēkā „Diagnostic.dll” izmantotās palīgklases nodrošina nepieciešamās žurnalēšanas un auditācijas funkcijas.

## Logošanas palīgklase

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

LogUtility

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, IDictionary<string, object> properties)

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 |
| properties | papildus parametri |

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 |

### Piemērs

// Izveidojam informāciju trasēšanai

int eventId = 1;

string message = „Simple message”;

string category = „General”

TraceEventType severity = TraceEventType.Information;

// Trasējam ziņoijumu

LogUtility target = new LogUtility();

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

## Izņēmumu palīgklase

Izņēmumu palīgklasē iekļautas metodes, kas ļauj apstrādāt izņēmumus.

ExceptionUtility

2.attēls. Izņēmumu palīgklasē iekļautās metodes

### Konstruktori

#### Konstruktors „ExceptionUtility”

Konstruktora apraksts:

public ExceptionUtility()

Izveido klasi ExceptionUtility.

public ExceptionUtility(LogUtility diagnosticTrace)

Izveido klasi ExceptionUtility ar trasēšanas iespējām.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| diagnosticTrace | trasēšanas klase |

### 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 izņēmums ir fatāls.

Parametru apraksts:

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

#### Metode „ThrowHelperArgument”

Metodes apraksts:

public Exception ThrowHelperArgument(string message);

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

Parametru apraksts:

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

#### Metode „ThrowHelperArgument”

Metodes apraksts:

public Exception 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 Exception ThrowHelperArgumentNull(string message);

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

Parametru apraksts:

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

#### Metode „ThrowHelperArgumentNull”

Metodes apraksts:

public Exception 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 |

## Trasēšanas palīgklase

Klase palīdz veikt trasēšanas iespējas.

TraceUtility

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 |

### 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 |

### Klases izmantošanas piemērs

public void DoSomesing() {

using (new TraceUtility("DoSomesing")) {

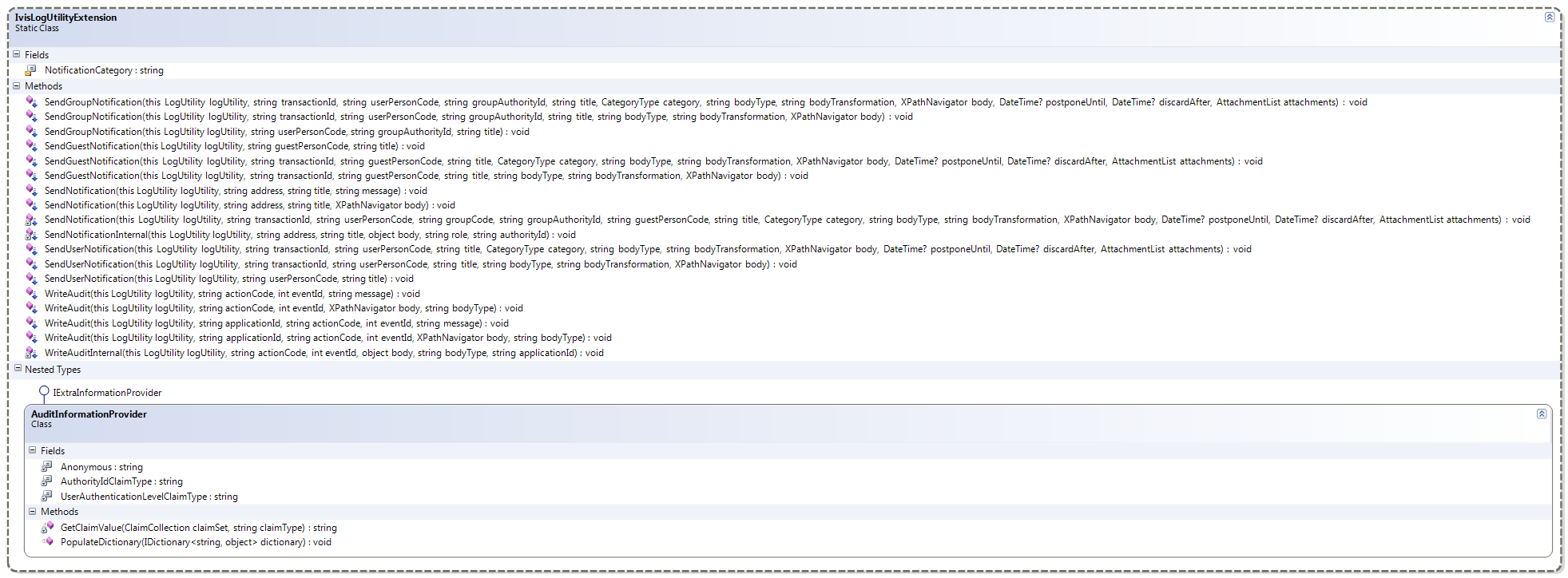
...

}

}

## Paplašinājumu palīgklase

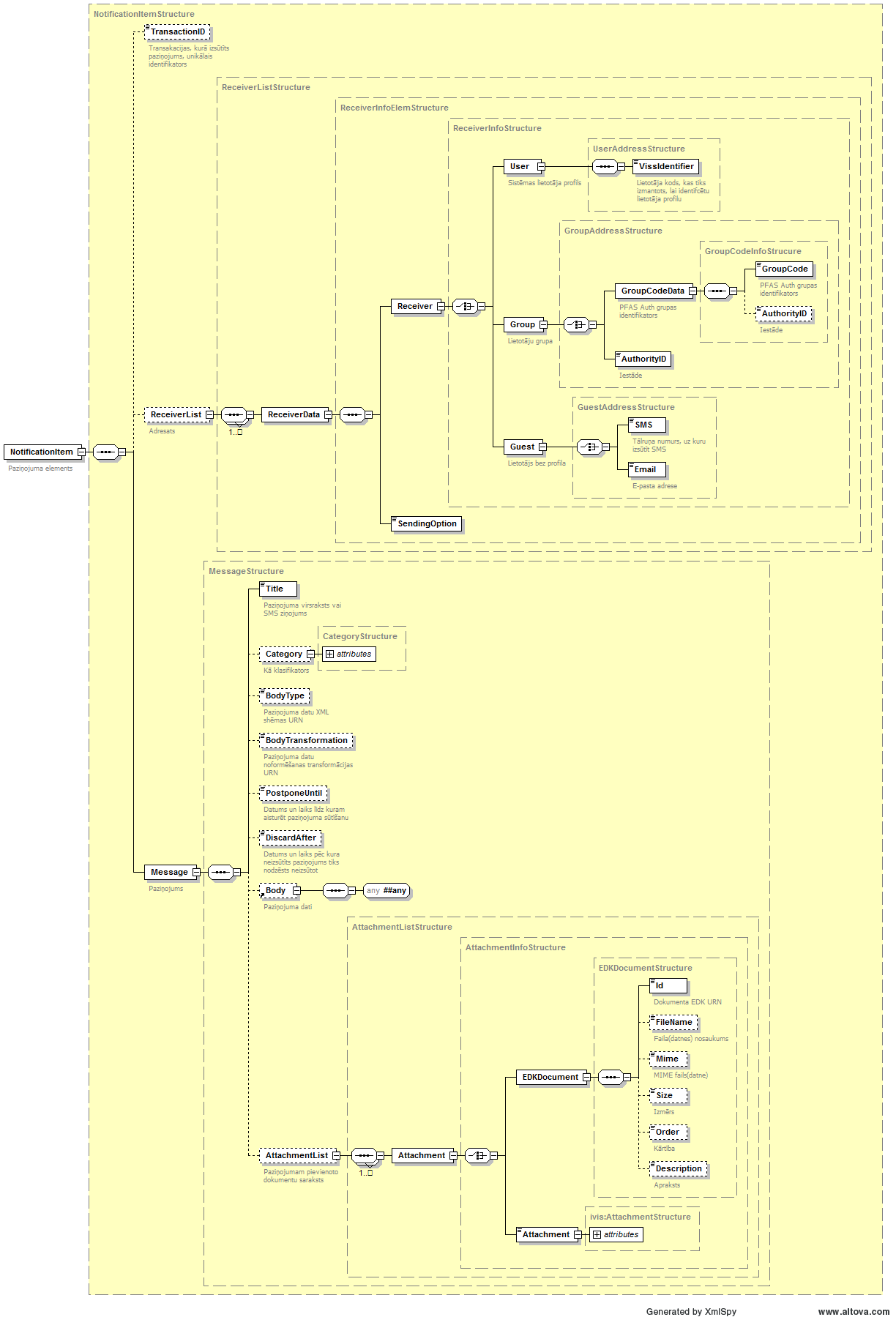
Satur papildus metodes notifikāciju sūtīšanai un audita rakstīšanai.



4.attēls. Paplašinājuma palīgklasē iekļautās metodes

### Paziņojumi (Notifikācijas)

Notifikācijas tiek nosūtītās izmantojot notifikāciju servisu, vai arī testēšanas nolūkos rakstītas teksta datnēs. Notifikāciju sūtīšanu nodrošina vairākas metodes, kas atrodas *IVIS.Diagnostics.d*ll bibliotēkas *IVIS.Diagnostics.IvisLogUtilityExtension* vārdtelpā. Notifikācijas tiek nosūtītas izmantojot <http://ivis.eps.gov.lv/XMLSchemas/100000/IVISInfrastructure/v2-0/Notification.xsd> struktūru.



5.attēls. Notifikāciju nosūtīšanas struktūra

Struktūras entītiju un atribūtu apraksts ir pieejams 1.tabulā.

1.tabula

Struktūras entītiju apraksts

| Entītija un tās apraksts | Entītijas atribūti | Atribūta apraksts |
| --- | --- | --- |
| Message (Paziņojums) – visi vienam paziņojumam specifiskie dati | Title (Tēma) | Īss paziņojuma satura vai izsūtīšanas mērķa apraksts, ko noteicis paziņojuma izveidotājs. SMS gadījumā – arī paziņojuma saturs, e-pasta gadījumā *subject*. |
| Category (Kategorija) | Paziņojuma kategorija, kas ir jebkāda ārēja vai iekšēja klasifikatora vērtība ar/bez to identificējoša koda |
| BodyType (Paziņojuma satura tips) | Paziņojuma datu XML shēmas URN VISS resursu katalogā |
| BodyTransformation (Paziņojuma satura transformācijas URN) | Paziņojuma datu noformēšanas transformācijas URN VISS resursu katalogā |
| PostponeUntil (Izsūtīšanas laiks) | Ja norādīts, datums un laiks, līdz kuram ir atlikta paziņojuma izsūtīšana |
| DiscardAfter (Atcelt izsūtīšanu pēc) | Datums un laiks, kuru sasniedzot, neizsūtīti paziņojumi tiek anulēti, t.i. netiks izsūtīti. |
| Body (Saturs) | Paziņojuma teksts vai dati, kas veido paziņojuma tekstu. |
| AttachmentList (pielikumi) | Skatīt Attachment entītiju. |
| Attachment (Pielikums) – atsauce uz elektronisko dokumentu krātuvē saglabātu dokumentu, vai ziņojumam pievienots dokuments. | EDKDocument (EDK dokuments) | Skatīt EDKDocument entītiju |
| Attachment | Pievienotais dokuments |
| EDKDocument (EDK dokuments) – VISS elektronisko dokumentu krātuvē saglabāts dokuments | Id (identifikators) | VISS elektronisko dokumentu krātuvē saglabāta dokumenta unikālais identifikators URN formātā. |
| FileName (Datnes nosaukums) | Dokumenta datnes nosaukums |
| Mime (Datnes Mime tips) | Datnes veids pēc Mime standarta |
| Size (Izmērs) | Datnes izmērs baitos |
| Order (Secība) | Dokumenta secība paziņojumā |
| Description (Apraksts) | Dokumenta apraksts no VISS elektronisko dokumentu krātuves |
| ReceiverList (Adresāti) –informācija par paziņojuma adresātiem – tie var būt konkrēti lietotāji, anonīmu lietotāju kontaktinformācija vai norāde uz lietotāju grupu un/vai iestādi | ReceiverData (Saņēmāja dati) | Skatīt ReceiverData entītiju |
| ReceiverData (Adresāta dati) detalizēta informācija par katru adresātu | Receiver (Saņēmējs) | Skatīt Receiver entītiju |
| SendingOption (Sūtīšanas kanāls) | Paziņojuma sūtīšanas kanāls – viens no:   * telephone (tālruņa numurs SMS sūtīšanai); * kdv (klienta darba vieta); * email (e-pasta adrese) |
| Receiver (Adresāts) | User (Lietotājs) | Skatīt entītijas User aprakstu. |
| Group (Lietotāju grupa) | Skatīt entītijas Group aprakstu |
| Guest (Lietotājs – viesis) | Skatīt entītijas Guest aprakstu |
| User (Lietotājs) | VissIdentifier (VISS identifikators) | Lietotāja VISS identifikators |
| Group (lietotāju grupa) | GroupCodeData | Skatīt entītijas GroupCodeData aprakstu |
| AuthorityID (Iestādes identifikators) | VISS iestādes 6 zīmju identifikators |
| GroupCodeData (Lietotāju grupas dati) | GroupCode | PFAS Auth grupas identifikators |
| AuthorityID (Iestādes identifikators) | VISS iestādes 6 zīmju identifikators |
| Guest (Viesis) – nodrošina specificētā tālruņa numura vai e-pasta adreses saglabāšanu līdz paziņojuma izsūtīšanas brīdim | SMS | Tālruņa numurs, uz kuru izsūtīt SMS |
| Email | E-pasta adrese, uz kuru izsūtīt elektronisko vēstuli |

#### Metodes

Izsaucot notifikācijas metodes, jāievēro sekojoši ierobežojumi:

1. Obligāti jānorāda vienu no saņēmēju identificējošajiem parametriem: *groupCode (*ar *authorityId)*, *userCode, authorityId* (visiem iestādes lietotājiem) vai *address*.
2. Obligāti jānorāda parametru *title*.
3. Ja tiek norādīts parametrs *body,* jānorāda arī parametrs *bodyType*. Papildus, ir iespēja noradīt *body* transformāciju *bodyTransformation*.
4. Ja tiek norādīts parametrs *bodyTransformation*, jānorāda arī parametri *bodyType* un *body*.
5. Datuma formāts netiek ierobežots un var tik, uzdots līdz nepieciešamajai precizitātei.
6. Parametra *body* izmērs pēc transformācijas nedrīkst pārsniegt 5MB.
7. IVIS.Diagnostic.dll bibliotēka neierobežo nosūtāmo pielikumu izmēru, to nosaka EDK un Notifikāciju servisa konfigurācija.
8. IVIS.Diagnostic.dll bibliotēka veic sūtīšanu asinhroni, paziņojuma izsūtīšana ir atkarīga no klausītāja konfigurācijas.

Metožu parametru izmantošana atkarība no sūtīšanas kanāla ir redzama 2.tabulā.

2.tabula

Metožu parametru izmantošana atkarība no sūtīšanas kanāla (M – obligāts, O – neobligāts)

|  |  |  |  |
| --- | --- | --- | --- |
| **Parametra nosaukums** | **E-pasts** | **IDDV/KDV** | **Sms** |
| transactionId | O | O |  |
| address, groupCode (ar authorityId), authorityId vai userCode | M | M | M |
| title | M | M | M |
| category | O | O |  |
| bodyTransformation (ja ir noradīts body) | O | O |  |
| Body un bodyType | O | O |  |
| postponeUnitil | O | O |  |
| discardAfter | O | O |  |
| attachments | O | O |  |

##### Metode „SendGroupNotification”

Metodes apraksts:

public static void SendGroupNotification(string transactionId, string groupCode, string authorityId, string title, CategoryType category, string bodyType, string bodyTransformation, XPathNavigator body, DateTime? postponeUntil, DateTime? discardAfter, AttachmentList attachments, SendingType sendingType = SendingType.email)

Nosūta lietotāju grupai paziņojumu pēc lietotāju grupas koda un/vai iestādes identifikatora.

Parametru apraksts:

| Nosaukums | Apraksts |
| --- | --- |
| transactionId | transakcijas, kurā izsūtīts paziņojums unikālais identifikators, tikai e-pakalpojumiem. |
| groupCode | saņēmēja lietotāju grupas identifikators |
| authorityId | saņēmēja iestādes identifikators, kas ir reģistrēts VISS Klasifikatoru kataloga Iestāžu klasifikatorā |
| title | paziņojuma virsraksts vai SMS ziņojums |
| category | paziņojuma kategorija, kas ir jebkāda ārēja vai iekšēja klasifikatora vērtība ar/bez to identificējošā koda |
| bodyType | paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification”. |
| bodyTransformation | paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga) |
| body | paziņojuma dati |
| postponeUnitil | datums un laiks, līdz kuram aizturēt paziņojuma sūtīšanu |
| discardAfter | datums un laiks, pēc kura nenosūtīts paziņojums, tiks nodzēsts neaizsūtot. |
| attachments | paziņojumam pievienotie dokumenti:   * fiziskie pielikumi * reference uz EDK pievienoto dokumentu |
| sendingType | Paziņojuma sūtīšanas iespēja: e-pasts, IDDV vai sms. Pēc noklusējuma šis parametrs vienmēr ir e-pasts. |

##### Metode „SendNotification”

Metodes apraksts:

public static void SendNotification(string transactionId, string address, string title, CategoryType category, string bodyType, string bodyTransformation, XPathNavigator body, DateTime? postponeUntil, DateTime? discardAfter, AttachmentList attachments)

Nosūta lietotājam bez profila paziņojumu uz e-pastu.

Parametru apraksts:

| Nosaukums | Apraksts |
| --- | --- |
| transactionId | transakcijas, kurā izsūtīts paziņojums unikālais identifikators, tikai e-pakalpojumiem. |
| address | saņēmēja e-pasts |
| title | paziņojuma virsraksts vai SMS ziņojums |
| category | paziņojuma kategorija |
| bodyType | paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification”. |
| bodyTransformation | paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga) |
| body | paziņojuma dati |
| postponeUnitil | datums un laiks, līdz kuram aizturēt paziņojuma sūtīšanu |
| discardAfter | datums un laiks, pēc kura nenosūtīts paziņojums, tiks nodzēsts neaizsūtot. |
| attachments | paziņojumam pievienotie dokumenti:   * fiziskie pielikumi * reference uz EDK pievienoto dokumentu |

##### Metode „SendUserNotification”

Metodes apraksts:

public static void SendUserNotification(string transactionId, string userCode, string title, CategoryType category, string bodyType, string bodyTransformation, XPathNavigator body, DateTime? postponeUntil, DateTime? discardAfter, AttachmentList attachments, SendingType sendingType = SendingType.email)

Nosūta sistēmas lietotājam paziņojumu pēc tā personas koda.

Parametru apraksts:

| Nosaukums | Apraksts |
| --- | --- |
| transactionId | transakcijas, kurā izsūtīts paziņojums unikālais identifikators, tikai e-pakalpojumiem. |
| userCode | personas kods vai cits saņēmēja identifikators |
| title | paziņojuma virsraksts vai SMS ziņojums |
| category | paziņojuma kategorija |
| bodyType | paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification”. |
| bodyTransformation | paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga) |
| body | paziņojuma dati |
| postponeUnitil | datums un laiks, līdz kuram aizturēt paziņojuma sūtīšanu |
| discardAfter | datums un laiks, pēc kura nenosūtīts paziņojums, tiks nodzēsts neaizsūtot. |
| attachments | paziņojumam pievienotie dokumenti:   * fiziskie pielikumi * reference uz EDK pievienoto dokumentu |
| sendingType | Paziņojuma sūtīšanas iespēja: e-pasts, IDDV vai sms. Pēc noklusējuma šis parametrs vienmēr ir e-pasts. |

### Audits

Metodes atrodas *IVIS*.*Diagnostics*.*dll* bibliotēkas IVIS.*Diagnostics*.*IvisLogUtilityExtension* vārdtelpā. Auditācijas datu sūtītāja informācija tiek iegūtā no drošības talona.

#### Metodes

##### Metode „WriteAudit”

Metodes apraksts:

public static void WriteAudit(string actionCode, int eventId, XPathNavigator body, string bodyType)

Raksta veiktās darbības auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| actionCode | darbības veids, skat. 2.4.2.2.sadaļā |
| eventId | sistēmas stāvokļa identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmas notikumu klasifikatorā |
| body | paziņojuma dati |
| bodyType | paziņojuma datu XML shēmas URN |

##### Metode „WriteAudit”

Metodes apraksts:

public static void WriteAudit(string applicationId, string actionCode, int eventId, XPathNavigator body, string bodyType, string objectId, string objectTypeId)

Raksta veiktās darbības auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| applicationId | lietojuma identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmu klasifikatorā |
| actionCode | darbības veids, skat. 2.4.2.2.sadaļā |
| eventId | sistēmas stāvokļa identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmas notikumu klasifikatorā |
| body | paziņojuma dati |
| bodyType | paziņojuma datu XML shēmas URN |
| objectId | Objekta, ar kuru notiek darbība identifikators |
| objectTypeId | Objekta, ar kuru notiek darbība tips |

##### Metode „WriteAudit”

Metodes apraksts:

public static void WriteAudit(string actionCode, int eventId, string message)

Raksta veiktās darbības auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| actionCode | darbības veids, skat. 2.4.2.2.sadaļā |
| eventId | sistēmas stāvokļa identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmas notikumu klasifikatorā |
| Message | paziņojuma teksts |

##### Metode „WriteAudit”

Metodes apraksts:

static void WriteAudit(string applicationId, string actionCode, int eventId, string message, string objectId, string objectTypeId)

Raksta veiktās darbības auditu.

Parametru apraksts:

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| applicationId | lietojuma identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmu klasifikatorā |
| actionCode | darbības veids, skat. 2.4.2.2.sadaļā |
| eventId | sistēmas stāvokļa identifikators, kas ir reģistrēts VISS Klasifikatoru katalogā Informācijas sistēmas notikumu klasifikatorā |
| message | paziņojuma teksts |
| objectId | Objekta, ar kuru notiek darbība identifikators |
| objectTypeId | Objekta, ar kuru notiek darbība tips |

#### Parametru vērtību specificēšana

Izsaucot *WriteAudit* metodes jāievēro sekojoši priekšnosacījumi uzdodot parametru vērtības:

|  |  |  |
| --- | --- | --- |
| Notikums | Identifikātors | Apraksts |
| Created | 0 | Tiek pievienots objekts |
| Read | 1 | Tiek nolasīts objekts (no datu bāzes vai faila) |
| Edited | 2 | Tiek modificēts objekts |
| Deleted | 3 | Tiek dzēsts objekts |

Parametru vērtību piešķiršana:

|  |  |  |
| --- | --- | --- |
| Parametrs | Vērtīiba | Piezīmes |
| actionCode | {objekta tips}[Notikums] |  |
| eventId | [Identifikators] |  |
| message | {objekta tips} {objekta nosaukums} [Notikums] | Lokalizēts atbilstoši konfigurācijai |

Parametru vērtību piemēri:

|  |  |  |  |
| --- | --- | --- | --- |
| applicationId | actionCode | eventid | message |
| VISS.RK | XSDType | 0 | Tips ‘{0}’ izveidots |
| VISS.RK | XSDGroup | 0 | Grupa ‘{0}’ izveidota |
| VISS.RK | TypeDeleted | 3 | Tips ‘{0}’ izdzēsts |
| VISS.RK | TypeEdited | 2 | Tips ‘{0}’ modificēts |
| VISS.RK | GroupEdited | 2 | Grupa ‘{0}’ modificēta |

# 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 bibliotēku. (tipiski atrodama katalogā *„~\Diagnostic\Lib”*).

## LogUtility

Žurnalēšanas funkcionalitātes piesaistīšana esošam projektam izmantojot *LogUtility* klasi. Izpildītais risinājums 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 s*o*urce 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

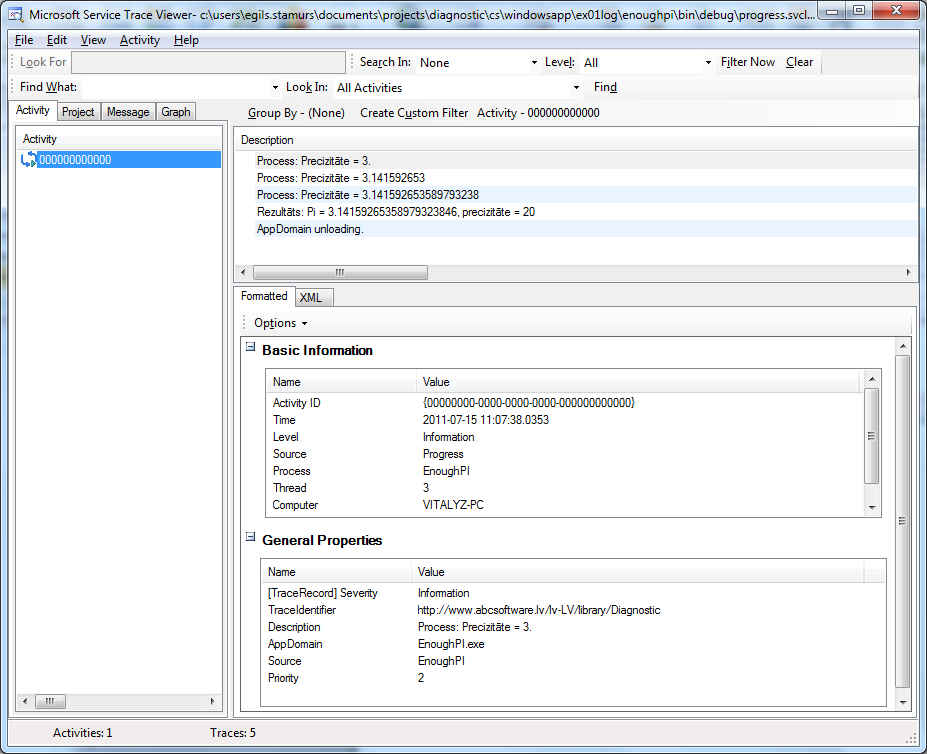
{

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

## 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 atrodama 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;

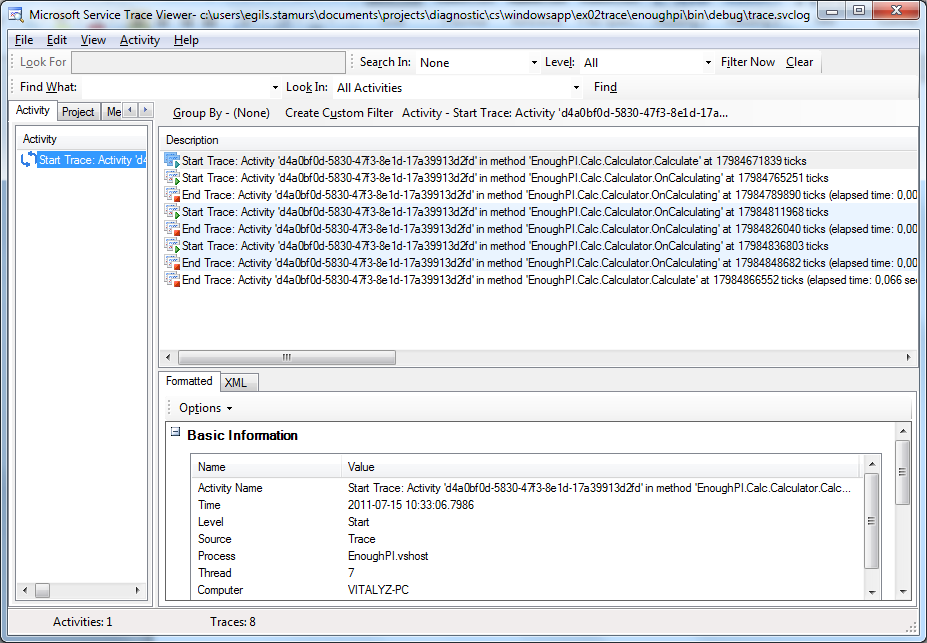
public struct Category

{

public const string Trace = "Trace";

}

1. Iegūtais rezultāts trasējot metodes *Calculate* un *OnCalculating* (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 *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

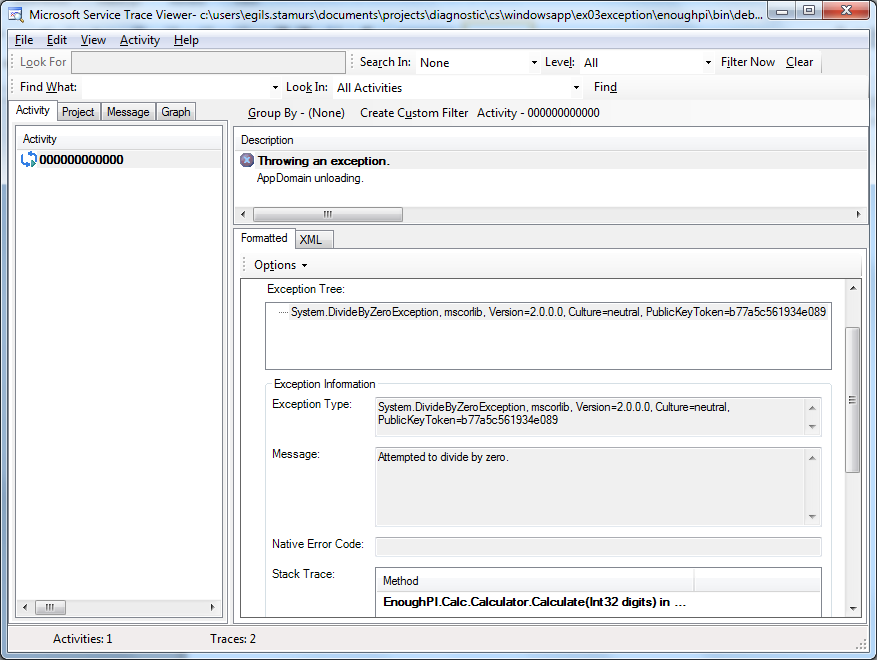
exceptionWriter.ThrowHelperError(args.Exception);

if (CalculatorException != null)

CalculatorException(this, args);

}

1. Šajā gadījumā netiek specificēta ziņojumu kategorija, bet lietota noklusētā kategorija „General”, tāpēc jāizveido atbilstoša ziņojumu klase *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 *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];

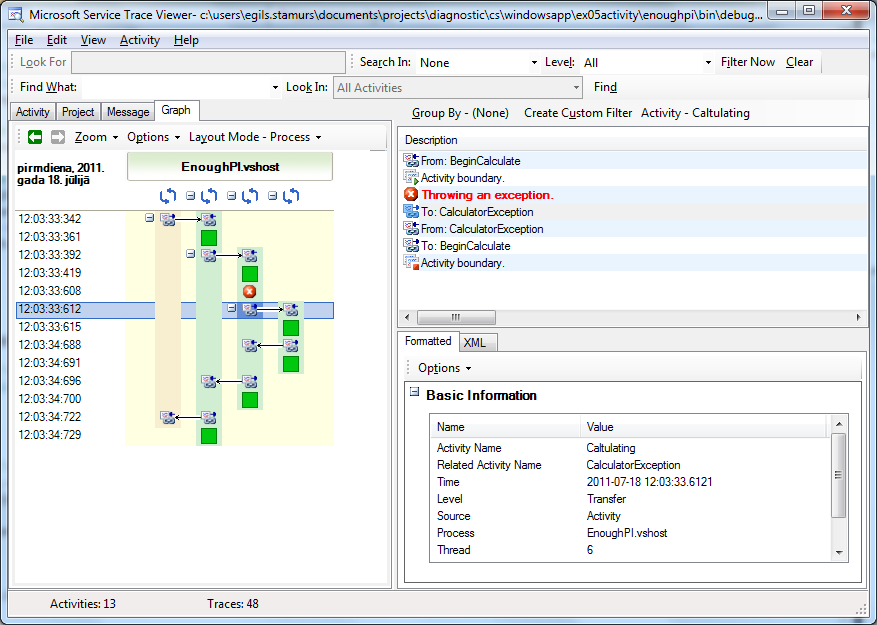
dlg.EndInvoke(ar);

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 (skat. 9.attēls.).



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);

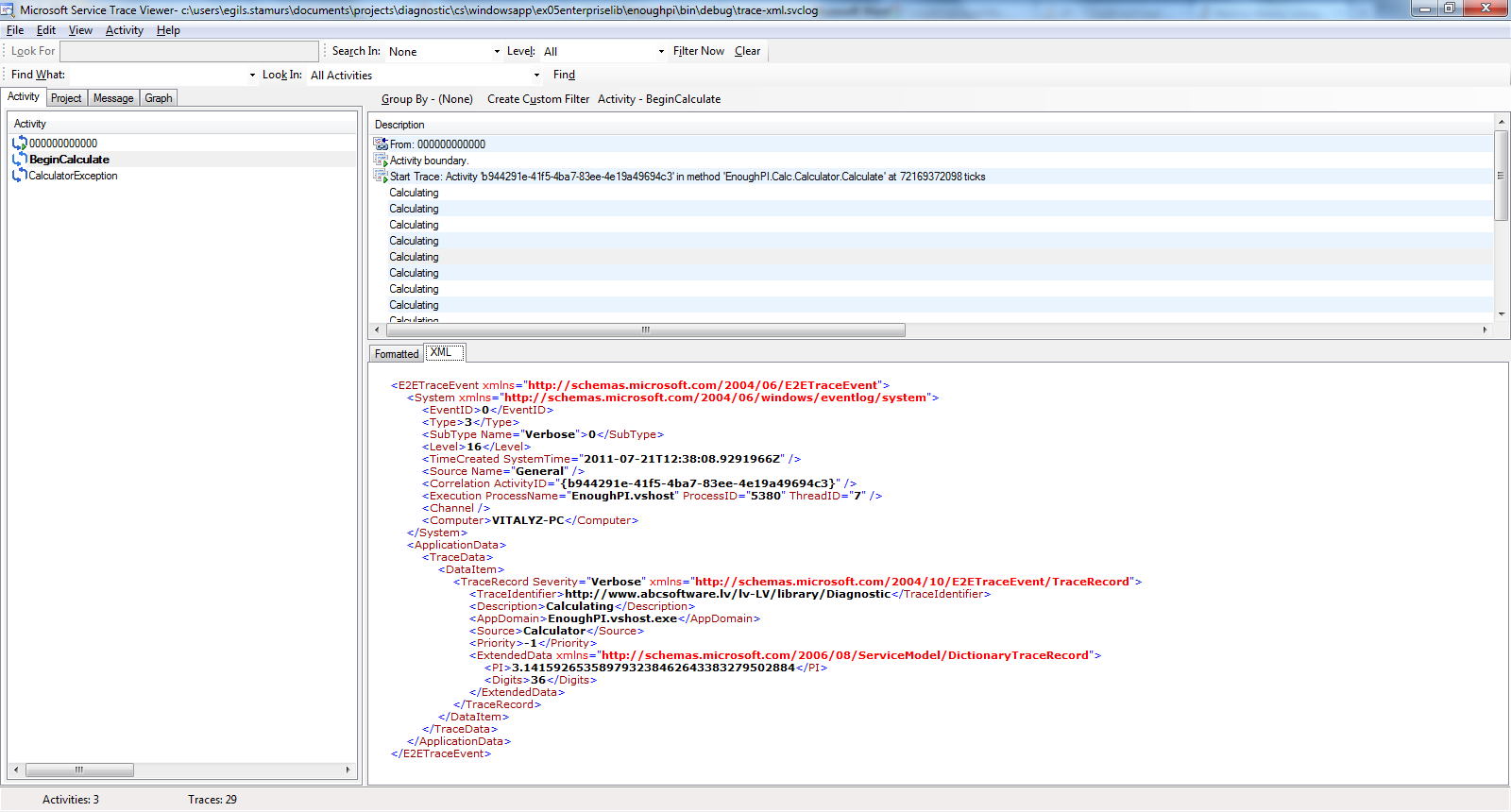
1. To pašu rezultātu iespējams sasniegt neveidojot atsevišķu klasi *Dictionary* tipa mainīgā aizpildīšanai, bet gan veicot sekojošas darbības:
   * Definēt *Dictionary* tipa mainīgo;
   * Pievienot mainīgajam nepieciešamās īpašības un to vērtības izsaucot metodi *Add***;**
   * Izsaukt *LogUtility* metodi *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.attēls.).



10.attēls. 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]

<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>

## 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 atrodams 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>

* 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.attēls.).

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");

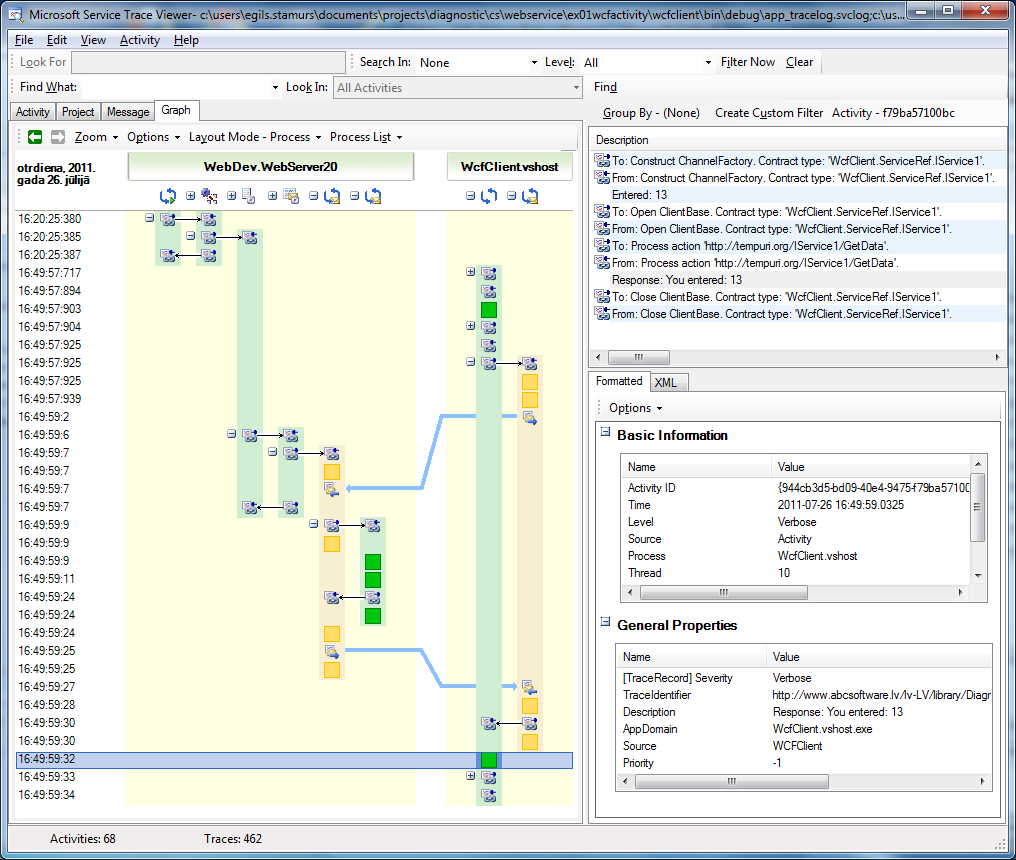
OutputTextBox.Text = rez;

}

}

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;

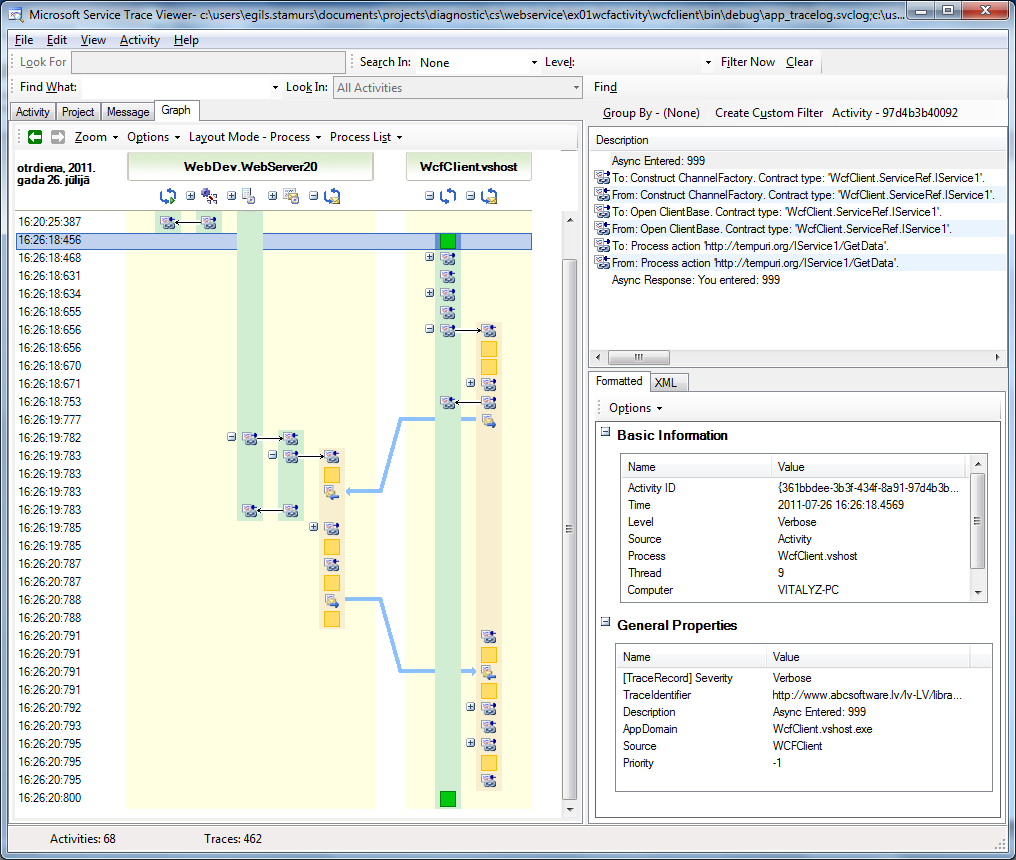
l.Resume();

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

## Paplašinājumu palīgklase

Paplašinājuma funkcionalitātes piesaistīšana esošam projektam. Izpildītais risinājums uz kā balstīta šī instrukcija atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex06Extensions”*.

1. Atvērt projektu *EnoughPI* un iezīmēt katalogu References, izpildīt konteksta izvēlnes komandu *Add* *Reference* un pievienot projektam IVIS.*Diagnostic*.*dll* bibliotēku. (tipiski atrodama katalogā *„~\Diagnostic\Lib”*), kas satur izmantojamās paplašinājuma metodes.
2. Iezīmēt projekta failu *Calc*\*Calculator*.*cs* un izpildīt konteksta izvēlnes komandu *View* *Code*;
3. Pievienot vārdtelpas *Diagnostic* un *Ivis*.*Diagnostic* faila augšdaļā;

using Diagnostic;

using IVIS.Diagnostics;

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*

<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="Notification.svclog"

type="System.Diagnostics.XmlWriterTraceListener,

System, Version=2.0.0.0, Culture=neutral,

PublicKeyToken=b77a5c561934e089"

name="NotificationLog"

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="Notification" switchValue="All">

<listeners>

<add name="NotificationLog"/>

</listeners>

</source>

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

<listeners>

<add name="AuditLog" />

</listeners>

</source>

</sources>

* Sadaļā *system*.dia*g*nostics 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 nepieciešamās rakstīšanas metodes.

Audita rakstīšanai:

protected void OnCalculated(CalculatedEventArgs args) {

logwriter.WriteAudit("Calculated", 1, string.Format("Calculation result is {0}", args.Pi));

}

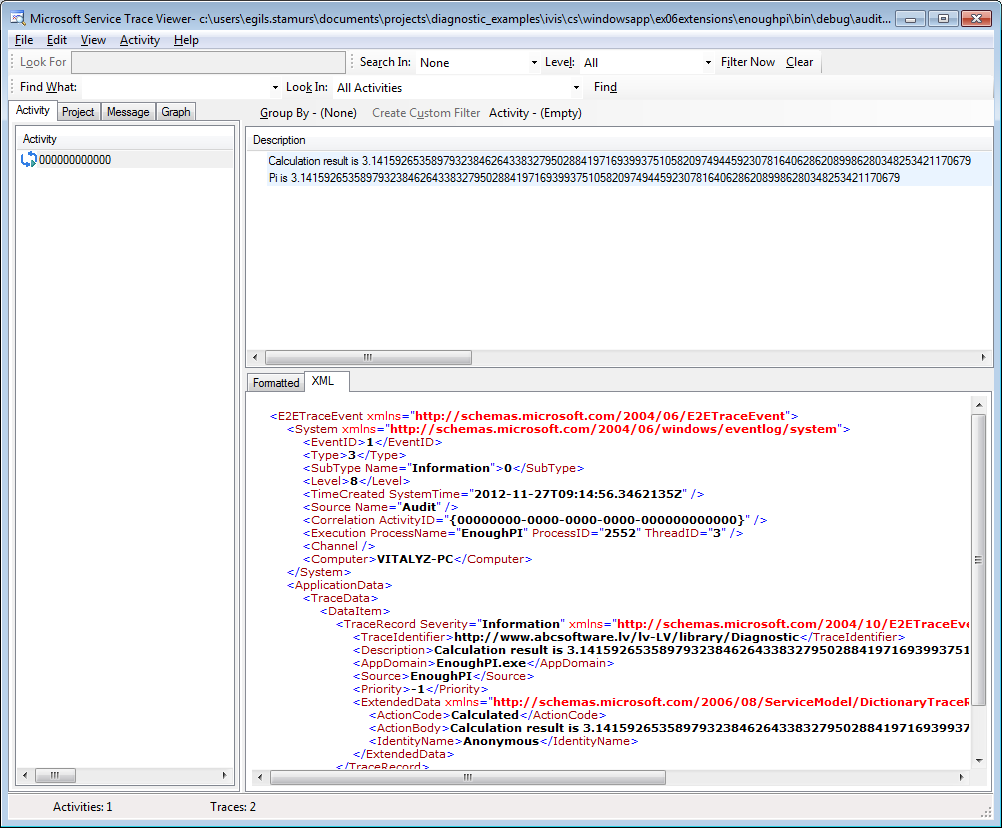
Paziņojumu rakstīšanai:

protected void OnCalculated(CalculatedEventArgs args) {

logwriter.SendUserNotification("01018133322", string.Format("Pi is {0}", args.Pi));

}

1. Iegūtais rezultāts apkopojot *Audit* un *Notification* žurnālus (skat. 13.attēls.).



13.attēls. Ziņojumu un audita rezultāti

### Notifikācijas servisa konfigurācija.

Lai sūtītu notifikācijas uz e-pastu nevis, kā iepriekš demonstrēts, rakstītu datnēs, nepieciešams veikt notifikācijas servisa konfigurēšanu un diagnostikas konfigurācijas sekciju papildināšanu:

1. Modificēt konfigurācijas sadaļas *diagnosticConfiguration* atribūtu *type;*

<diagnosticConfiguration type="IVIS.Diagnostics.IvisLogWriter, IVIS.Diagnostics"/>

1. Papildināt sekcijas *sources* apakšsekciju *listeners*;

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

<listeners>

<!-- Konfigurācija paziņojumu transformēšanai un sūtīšanai pa pastu -->

<add transformationSchemaURN="URN:IVIS:100001:XSD-eServiceRegistry-eServiceRegistry-v1-0-XSLT-EservRegis"

defaultMessageTitle="MyDefaultTitle" initializeData="NotificationServiceV2"

type="IVIS.Diagnostics.NotificationService2TraceListener, IVIS.Diagnostics"

name="Notification2TraceListener">

<filter type="" />

</add>

</listeners>

</source>

1. Ja tiek izmantots .Net 3.5 pievienot sekciju *system.serviceModel*;

<system.serviceModel>

<diagnostics>

<messageLogging logEntireMessage="true" logMalformedMessages="false"

logMessagesAtServiceLevel="false" logMessagesAtTransportLevel="false" />

</diagnostics>

<bindings>

<customBinding>

<!-- .NET3.5 ivis tv -->

<binding name="ws2007FederationNoSct">

<security authenticationMode="IssuedTokenOverTransport" messageSecurityVersion="WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12BasicSecurityProfile10" requireSecurityContextCancellation="false">

<issuedTokenParameters>

<claimTypeRequirements>

<add claimType="http://www.oasis-open.org/RSA2004/attributes/AUTHORITY" isOptional="false"/>

<add claimType="https://ivis.eps.gov.lv/schema/identity/claims/legalentity" isOptional="false"/>

<add claimType="http://docs.oasis-open.org/wsfed/authorization/200706/claims/action" isOptional="false"/>

<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier" isOptional="false" />

</claimTypeRequirements>

<issuer address="https://ivistv.abcsoftware.lv/PFAS/PFAS.STS/v1-1/STS/Issue.svc/trust/13/certificatemixed" binding="ws2007HttpBinding" bindingConfiguration="certificateMixed"/>

<issuerMetadata address="https://ivistv.abcsoftware.lv/PFAS/PFAS.STS/v1-1/STS/Issue.svc/mex"/>

</issuedTokenParameters>

<secureConversationBootstrap/>

</security>

<textMessageEncoding/>

<httpsTransport/>

</binding>

</customBinding>

<ws2007HttpBinding>

<binding name="certificateMixed">

<security mode="TransportWithMessageCredential">

<message clientCredentialType="Certificate" establishSecurityContext="false"/>

</security>

</binding>

</ws2007HttpBinding>

</bindings>

<client>

<!-- .NET3.5 ivistv-->

<endpoint name="NotificationServiceV2" address="https://ivistv.abcsoftware.lv/Notification/v2-0/ws2007FederationNoSct" contract="NotificationService.INotificationServiceContract"

binding="customBinding" bindingConfiguration="ws2007FederationNoSct" behaviorConfiguration="certificate"/>

</client>

<behaviors>

<endpointBehaviors>

<behavior name="certificate">

<clientCredentials>

<clientCertificate findValue="7d a6 83 cf 8b 85 9a 23 8f 26 c3 19 1a 98 fb 0c e3 9f 60 12" storeLocation="LocalMachine" x509FindType="FindByThumbprint" />

</clientCredentials>

</behavior>

</endpointBehaviors>

</behaviors>

</system.serviceModel>

1. Ja tiek izmantots .Net 4.0 vai jaunāks pievienot sekciju *system.serviceModel*;

<system.serviceModel>

<diagnostics>

<!-- Nomainīt vērtības uz true, lai veiktu servisa logu rakstīšanu-->

<messageLogging logEntireMessage="false" logMessagesAtTransportLevel="false" logMalformedMessages="false"/>

</diagnostics>

<bindings>

<!-- .NET4.0 ivistv -->

<ws2007FederationHttpBinding>

<binding name="ws2007FederationNoSct">

<security mode="TransportWithMessageCredential">

<message establishSecurityContext="false">

<claimTypeRequirements>

<add claimType="http://www.oasis-open.org/RSA2004/attributes/AUTHORITY" isOptional="false"/>

<add claimType="https://ivis.eps.gov.lv/schema/identity/claims/legalentity" isOptional="false"/>

<add claimType="http://docs.oasis-open.org/wsfed/authorization/200706/claims/action" isOptional="false"/>

<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier" isOptional="false" />

</claimTypeRequirements>

<issuer address="https://ivistv.abcsoftware.lv/PFAS/PFAS.STS/v1-1/STS/Issue.svc/trust/13/certificatemixed" binding="ws2007HttpBinding" bindingConfiguration="certificateMixed"/>

<issuerMetadata address="https://ivistv.abcsoftware.lv/PFAS/PFAS.STS/v1-1/STS/Issue.svc/mex"/>

</message>

</security>

</binding>

</ws2007FederationHttpBinding>

<ws2007HttpBinding>

<binding name="certificateMixed">

<security mode="TransportWithMessageCredential">

<message clientCredentialType="Certificate" establishSecurityContext="false"/>

</security>

</binding>

</ws2007HttpBinding>

</bindings>

<client>

<!-- .NET4.0 ivistv -->

<endpoint name="NotificationServiceV2"

address="https://ivistv.abcsoftware.lv/Notification/v2-0/ws2007FederationNoSct"

contract="NotificationService.INotificationServiceContract"

binding="ws2007FederationHttpBinding"

bindingConfiguration="ws2007FederationNoSct"

behaviorConfiguration="certificate"

/>

</client>

<behaviors>

<endpointBehaviors>

<behavior name="certificate">

<clientCredentials>

<clientCertificate findValue="7d a6 83 cf 8b 85 9a 23 8f 26 c3 19 1a 98 fb 0c e3 9f 60 12" storeLocation="LocalMachine" x509FindType="FindByThumbprint" />

</clientCredentials>

</behavior>

</endpointBehaviors>

</behaviors>

</system.serviceModel>

1. Importēt sertifikātu *„~\Diagnostic\Certificates\ex06Extensions.pfx”* norādot paroli „123”. Ja nepieciešams izmantot citu sertifikātu, modificēt sekciju *system.serviceModel / behaviors / endpointBehaviors / behavior / clientCredentials / clientCertificate*.