VALSTS REĢIONĀLĀS ATTĪSTĪBAS AĢENTŪRA

VALSTS INFORMĀCIJAS SISTĒMU SAVIETOTĀJA (VISS) UN VIENOTĀ VALSTS UN PAŠVALDĪBU PAKALPOJUMU PORTĀLA WWW.LATVIJA.LV PILNVEIDOŠANA UN UZTURĒŠANA

VISS SISTĒMAS ŽURNĀLS

KOPLIETOJUMA BIBLIOTĒKU APRAKSTS

VRAA-13_7_17_41-VISS_2016-VISS_ZUR-KBA

07.11.2022. versija 1.13

Rīgā 2018

Satura rādītājs

Attēlu saraksts

1. Ievads

1.1. Dokumenta nolūks

Dokuments „Koplietojuma žurnalēšanas bibliotēku apraksts" satur žurnalēšanas bibliotēku *Diagnostic.dll* un *Enterprise Library* 4.0.0.0 aprakstu un palīgklašu izmantošanas instrukcijas. Vēl šajā dokumentā sniegta arī Diagnostic sekcijas konfigurācijas instrukcija administratoriem VISS vidēm. Šis dokuments ir paredzēts SIA „ABC software" izstrādātājiem, VRAA administratoriem, kā arī citiem iesaistītajiem izstrādātājiem, kas līdzdarbojas programmnodrošinājumu izstrādē un pilnveidošanā.

1.2. Termini un pieņemtie apzīmējumi

Visi šajā dokumentā izmantotie termini un apzīmējumi ir apkopoti Terminu un saīsinājumu indeksā [2].

1.3. Saistība ar citiem dokumentiem

Dokuments ir izstrādāts, balstoties uz šādiem dokumentiem:

[1] 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 izstrāde. Vadlīnijas (VRAA-6_15_11_58-VISS_2010-VISS-VDL).

[2] "Valsts informācijas sistēmu savietotāju (VISS) un Vienotā valsts un pašvaldību pakalpojumu portāla www.latvija.lv pilnveidošana un uzturēšana". Terminu un saīsinājumu indekss. (VRAA-13_7_17_41-VISS_2016-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".

2. Palīgklases

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

2.1. Logošanas palīgklase

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

1.attēls. Logošanas palīgklasē iekļautās metodes

2.1.1. Konstruktori

2.1.1.1. Konstruktors „LogUtility”

```
public LogUtility(string sourceName)
```

Izveido klasi „LogUtility” ar uzdoto žurnalēšanas avotu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
sourceName	trasēšanas avots

2.1.2. Īpašības

2.1.2.1. „LoggingEnabled” īpašība

Īpašības apraksts:

```
public bool LoggingEnabled { get; }
```

Īpašība atgriež informāciju, vai žurnalēšana ir ieslēgta.

2.1.3. Metodes

2.1.3.1. Metode „Write”

Metodes apraksts:

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

Logo ziņojumu ar uzdoto kategoriju, prioritāti, identifikatoru, nozīmīgumu 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 nozīmīgums
properties	papildus parametri

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

Logo ziņojumu ar uzdoto kategoriju, prioritāti, identifikatoru, nozīmīgumu 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 nozīmīgums
exception	izņēmums

2.1.4. Piemērs

```
// Izveidojam informāciju trasēšanai
int eventId = 1;
string message = „Simple message”;
string category = „General”
TraceEventType severity = TraceEventType.Information;

// Trasējam ziņojumu
LogUtility target = new LogUtility();
target.Write(message, category, priority, eventId, severity);
```

2.2. Izņēmumu palīgklase

Izņēmumu palīgklasē iekļautas metodes, kas ļauj apstrādāt izņēmumus.

2.attēls. Izņēmumu palīgklasē iekļautās metodes

2.2.1. Konstruktori

2.2.1.1. 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

2.2.2. Metodes

2.2.2.1. Metode „ThrowHelper”

Metodes apraksts:

```
public Exception ThrowHelper(Exception exception, TraceEventType eventType);
```

Apstrādā izņēmumu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums
eventType	izņēmuma veids

2.2.2.2. Metode „ThrowHelperWarning”

Metodes apraksts:

```
public Exception ThrowHelperWarning(Exception exception);
```

Apstrādā izņēmumu ar veidu „Brīdinājums”.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums

2.2.2.3. Metode „ThrowHelperError”

Metodes apraksts:

```
public Exception ThrowHelperError(Exception exception);
```

Apstrādā izņēmumu ar veidu „Kļūda”.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums

2.2.2.4. Metode „ThrowHelperCritical”

Metodes apraksts:

```
public Exception ThrowHelperCritical(Exception exception);
```

Apstrādā izņēmumu ar veidu „Kritisks”.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums

2.2.2.5. Metode „ThrowHelperFatal”

Metodes apraksts:

```
public Exception ThrowHelperFatal(string message, Exception innerException);
```

Izveido un apstrādā fatālo izņēmumu ar veidu „Kļūda”.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
message	izņēmuma ziņojums
exception	Izņēmums, kurš radījis fatālo izņēmumu

2.2.2.6. Metode „IsFatal”

Metodes apraksts:

```
public static bool IsFatal(Exception exception);
```

Pārbauda, vai uzdotais izņēmums ir fatāls.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums

2.2.2.7. 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

2.2.2.8. 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

2.2.2.9. 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

2.2.2.10. 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

2.2.2.11. Metode „UseActivityId”

Metodes apraksts:

```
public void UseActivityId(Guid activityId);
```

Veikt izņēmuma trasēšanu ar uzdoto aktivitātes identifikatoru.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
activityId	aktivitātes identifikators

2.2.2.12. Metode „ClearActivityId”

Metodes apraksts:

```
public void ClearActivityId();
```

Noņemt uzdoto aktivitātes identifikatoru.

2.2.2.13. Metode „TraceHandeledException”

Metodes apraksts:

```
public void TraceHandledException(Exception exception, TraceEventType eventType);
```

Veic izņēmuma trasēšanu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
exception	izņēmums
eventType	izņēmuma stingrība

2.3. Trasēšanas palīgklase

Klase palīdz veikt trasēšanas iespējas.

3.attēls. Trasēšanas palīgklasē iekļautās metodes

2.3.1. Konstruktori

2.3.1.1. 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

2.3.2. Metodes

2.3.2.1. Metode „StartTrace"

Metodes apraksts:

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

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

Parametru apraksts:

NOSAUKUMS	APRAKSTS
Operation	operācijas nosaukums
activityId	aktivitātes identifikators

2.3.3. Klases izmantošanas piemērs

```
public void DoSomesing() {
    using (new TraceUtility("DoSomesing")) {
        ...
    }
}
```

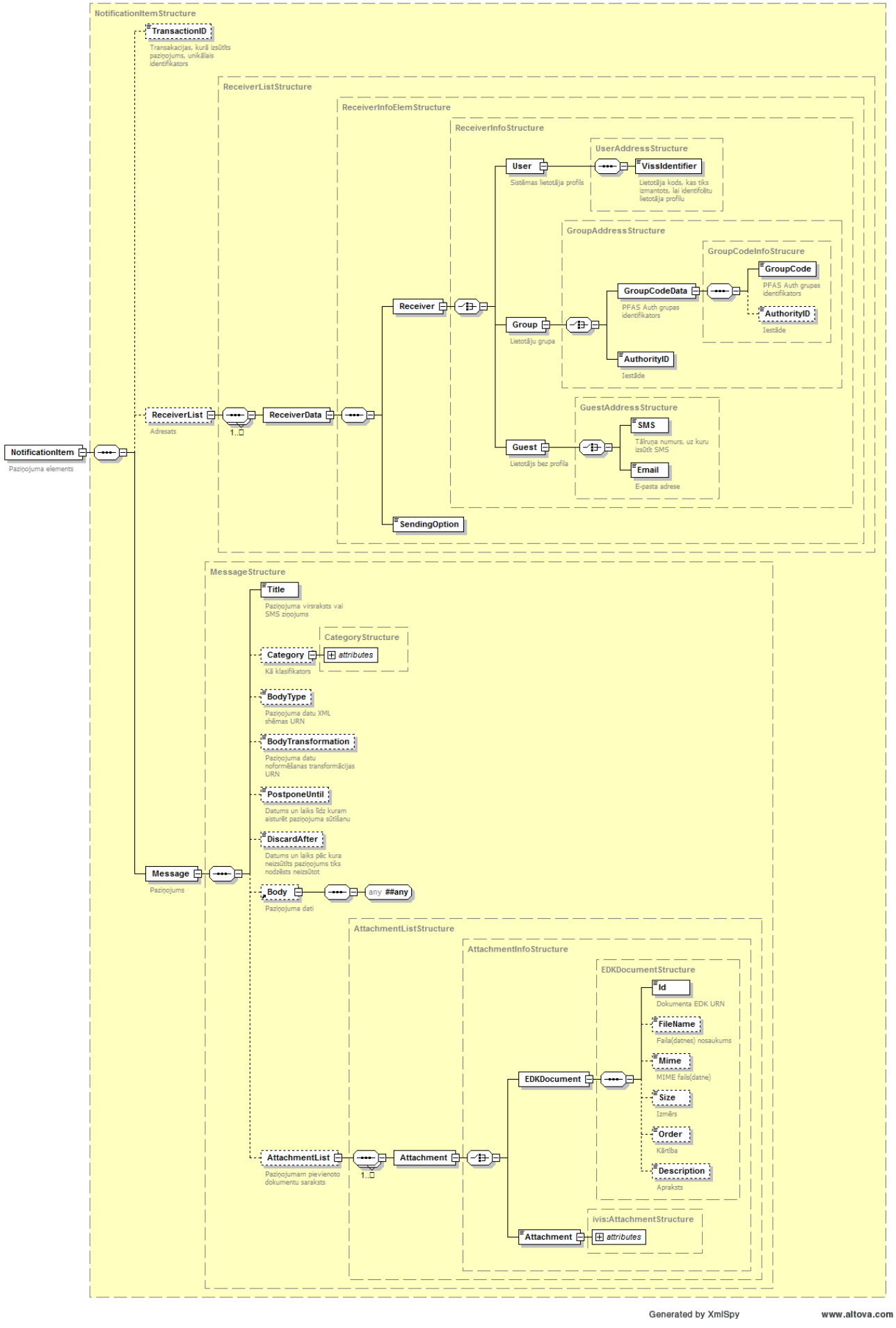
2.4. Paplašinājumu palīgklase

Satur papildu metodes notifikāciju sūtīšanai un audita rakstīšanai.

4.attēls. Paplašinājuma palīgklasē iekļautās metodes

2.4.1. Paziņojumi (Notifikācijas)

Notifikācijas tiek nosūtītās, izmantojot notifikāciju servisu, vai arī testēšanas nolūkos rakstītās teksta datnēs. Notifikāciju sūtīšanu nodrošina vairākas metodes, kas atrodas *IVIS.Diagnostics.dll* 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 nosaka 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 identificējošā 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, tad 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 saglabāto dokumentu, vai ziņojumam pievienoto dokumentu.	EDKDocument (EDK dokuments)	Skatīt EDKDocument entītiju.
	Attachment	Pievienotais dokuments.
EDKDocument (EDK dokuments) – VISS elektronisko dokumentu krātuvē saglabātais dokuments	Id (identifikators)	VISS elektronisko dokumentu krātuvē saglabātā 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 VISS elektronisko dokumentu krātuvē.
ReceiverList (Adresāti) – informācija par paziņojuma adresātiem – tie var būt konkrēti lietotāji, anonīmu lietotāju kontaktinformācija vai norāde uz lietotāju grupu un/vai iestādi	ReceiverData (Saņēmēja 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).

ENTĪTIJA UN TĀS APRAKSTS	ENTĪTIJAS ATRIBŪTI	ATRIBŪTA APRAKSTS
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.

2.4.1.1. Metodes

Izsaucot notifikācijas metodes, jāievēro šādi 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.

> ***Visi Notifikācijas servisa atgriežamo ziņojumu laika atribūti ir noradīti UTC.***

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	

2.4.1.1.1. Metode „SendGroupNotification”

Metodes apraksts:

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

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

Parametru apraksts:

NOSAUKUMS	APRAKSTS
groupCode	saņēmēja lietotāju grupas identifikators
authorityId	saņēmēja iestādes identifikators, kas ir reģistrēts VISS Klasifikatoru kataloga Iestāžu klasifikatorā
title	paziņojuma virsraksts vai SMS ziņojums
sendingType	Paziņojuma sūtīšanas iespēja: e-pasts, IDDV vai sms. Pēc noklusējuma šis parametrs vienmēr ir e-pasts.
bodyType	paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification”.
body	paziņojuma dati
bodyTransformation	paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga)
transactionId	transakcijas, kurā izsūtīts paziņojums, unikālais identifikators, tikai e-pakalpojumiem
category	paziņojuma kategorija, kas ir jebkāda ārēja vai iekšēja klasifikatora vērtība ar/bez to identificējošā koda
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

2.4.1.1.2. Metode „SendNotification”

Metodes apraksts:

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

Nosūta lietotājam bez profila paziņojumu uz e-pastu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
address	saņēmēja e-pasts
title	paziņojuma virsraksts vai SMS ziņojums
bodyType	paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification”
body	paziņojuma dati
bodyTransformation	paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga)
transactionId	transakcijas, kurā izsūtīts paziņojums, unikālais identifikators, tikai e-pakalpojumiem
category	paziņojuma kategorija

NOSAUKUMS	APRAKSTS
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

2.4.1.1.3. Metode „SendUserNotification"

Metodes apraksts:

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

Nosūta sistēmas lietotājam paziņojumu pēc tā personas koda.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
userCode	personas kods vai cits saņēmēja identifikators
title	paziņojuma virsraksts vai SMS ziņojums
sendingType	Paziņojuma sūtīšanas iespēja: e-pasts, IDDV vai sms. Pēc noklusējuma šis parametrs vienmēr ir e-pasts
bodyType	paziņojuma datu XML shēmas URN (no Resursu kataloga) ar norādītu elementa nosaukumu, piemēram:„URN:IVIS:100001:XSD-Viss-Notification-v1-0-TYPE-Notification".
body	paziņojuma dati
bodyTransformation	paziņojuma datu noformēšanas transformācijas URN (no Resursu kataloga)
transactionId	transakcijas, kurā izsūtīts paziņojums, unikālais identifikators, tikai e-pakalpojumiem
category	paziņojuma kategorija
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

2.4.2. Audits

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

2.4.2.1. Metodes

2.4.2.1.1. Metode „WriteAudit"

Metodes apraksts:

```
public static void WriteAudit(string actionCode, string message, IDictionary<string, object>
details = null, [int eventId = -1], [IEnumerable<ObjectInfo> objects = null], [RetentionInfo
retention = null], [string applicationId = null], [IPrincipal subject = null], [DateTime?
timeStamp = null])
```

Raksta veiktās darbības auditu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
actionCode	darbības veids, skat. 2.4.2.2.paragrāfā, kas ir reģistrēts VISS Klasifikatoru kataloga Auditācijas darbību klasifikatorā.
message	Notikuma apraksts
details	notikuma papildu atribūti
eventId	sistēmas stāvokļa identifikators, skat. 2.4.2.2. paragrāfā.
objects	saraksts ar tipiem un identifikatoriem objektiem (objectId un objectTypeId), ar kuriem notika auditējamā darbība
retention	Arhivēšanas grupa un arhivēšanas datums
applicationId	lietojuma identifikators, kas ir reģistrēts VISS Klasifikatoru kataloga Informācijas sistēmas notikumu klasifikatorā. Pēc noklusējuma, ja netiek aizpildīts, tiek ņemts no konfigurācijas datnes.
subject	Notikuma veidotajs.
timeStamp	Notikuma laiks UTC formāta.

2.4.2.1.2. Metode „WriteAudit" Novecojusi: DAIRM2 vairs neatbalsta datus XML formā

Metodes apraksts:

```
public static void WriteAudit(string actionCode, IXPathNavigable body, [string bodyType = null], [int eventId = 0], [string objectId = null], [string objectTypeId = null], [string applicationId = null])
```

Raksta veiktās darbības auditu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
actionCode	darbības veids, skat. 2.4.2.2.paragrāfā, kas ir reģistrēts VISS Klasifikatoru kataloga Auditācijas darbību klasifikatorā.
body	paziņojuma dati
bodyType	paziņojuma XML shēmas URN
eventId	sistēmas stāvokļa identifikators, skat. 2.4.2.2. paragrāfā.
objectId	objekta, ar kuru notiek darbība identifikators
objectTypeId	objekta, ar kuru notiek darbība tips, kurš ir reģistrēts VISS Klasifikatoru kataloga Auditācijas objektu klasifikatorā
applicationId	lietojuma identifikators kas ir reģistrēts VISS Klasifikatoru kataloga Informācijas sistēmas notikumu klasifikatorā.

2.4.2.1.3. Metode „WriteAudit"

Metodes apraksts:

```
public static void WriteAudit(string actionCode, string message, [int eventId = -1], string objectId = null], [string objectTypeId = null], [string applicationId = null])
```

Raksta veiktās darbības auditu.

Parametru apraksts:

NOSAUKUMS	APRAKSTS
actionCode	darbības veids, skat. 2.4.2.2. paragrāfā, kas ir reģistrēts VISS Klasifikatoru kataloga Auditācijas darbību klasifikatorā.
message	paziņojuma teksts
eventId	sistēmas stāvokļa identifikators, skat. 2.4.2.2. paragrāfā.
objectId	objekta, ar kuru notiek darbība identifikators
objectTypeId	objekta, ar kuru notiek darbība tips, kurš ir reģistrēts VISS Klasifikatoru kataloga Auditācijas objektu klasifikatorā

applicationId	lietojuma identifikators, kas ir reģistrēts VISS Klasifikatoru kataloga Informācijas sistēmu klasifikatorā

2.4.2.2. Parametru vērtību specificēšana

Izsaucot *WriteAudit* metodes jāievēro šādi priekšnosacījumi uzdodot parametra eventId vērtības:

NOTIKUMS	IDENTIFIKATORS	APRAKSTS
Unspecified	-1	Notikuma identifikators netiek norādīts
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
Login	4	Tiek veikta pieteikšanās
Logout	5	Tiek veikta atteikšanās

Parametru vērtību piešķiršana:

PARAMETRS	VĒRTĪBA	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

3. Klašu izmantošanas piemēri

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

3.1. 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 šādus 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"*).

3.2. LogUtility

Žurnalēšanas funkcionalitāti piesaista esošam projektam, izmantojot *LogUtility* klasi. Izpildītais risinājums, uz kā balstīta šī instrukcija, atrodams katalogā *„~\Diagnostic\CS\WindowsApp\ex01Log"*.

1. Iezīmējiet projekta failu *Calc\Calculator*.cs un izpildiet konteksta izvēlnes komandu *View Code*;
2. Pievienojiet vārdtelpu *Diagnostic* faila augšdaļā;

```
using Diagnostic;
```

3. Modificējiet konfigurācijas datni app.config, pievienojot šādus 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* definējiet koplietojamos žurnalēšanas failus, norādot šādus 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* definējiet ziņojumu klases source ar atbilstošiem identifikatoriem *name*. Katrai klasei piesaistīt vismaz vienu žurnalēšanas failu *listeners*, norādot kādu no koplietojamajiem failiem pēc to 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* definējiet atribūtu *trace* ar īpašību *autoflush* un vērtību *true*. Kas norāda, ka rakstīšana žurnalēšanas failā tiks veikta uzreiz.

```
        <trace autoflush="true" />
</system.diagnostics>
```

4. Failā *Calc\Calculator.cs* izveidojiet *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");
```

5. Nepieciešamajās vietās pievienojiet projekta kodam rakstīšanas metodes *Write*. Kategorijā norādiet žurnalēšanas klases identifikatoru;

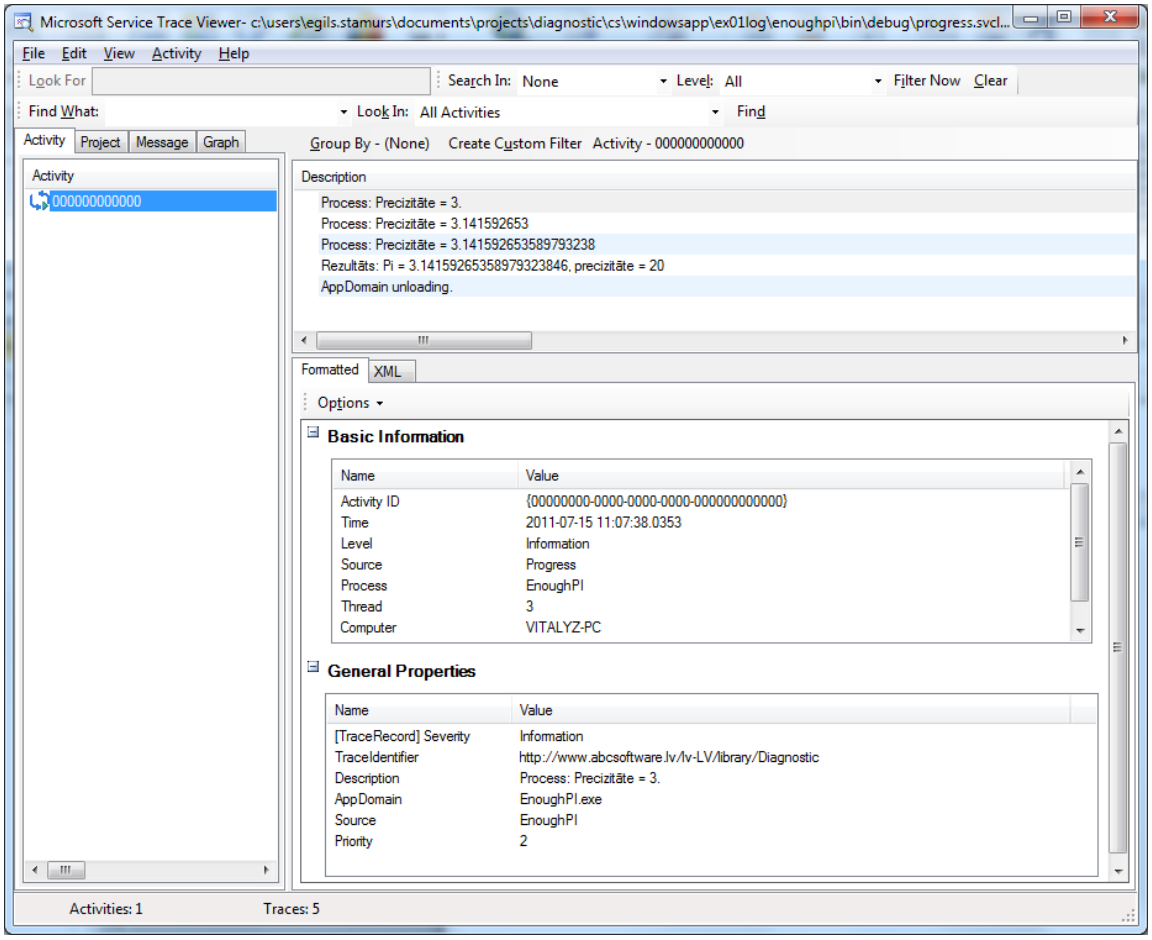
```
// TODO: Log final result
string message = string.Format("Rezultāts: Pi = {0}, precizitāte = {1}", args.Pi, args.Digits);
logwriter.Write(message, Category.General, Priority.Normal, 1,
System.Diagnostics.TraceEventType.Information);
```

```
// TODO: Log exception
logwriter.Write("Izņēmums", Category.General, Priority.High, 4,
System.Diagnostics.TraceEventType.Error, args.Exception);
```

6. Nepieciešamības gadījumā modificējiet 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";
}
```

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



6.attēls. Žurnalēšanas rezultāti

3.3. 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. Iezīmējiet projekta failu *Calc\Calculator.cs* un izpildiet konteksta izvēlnes komandu *View Code*;
2. Pievienojiet vārdtelpu *Diagnostic* faila augšdaļā;

```
using Diagnostic;
```

3. Modificējiet konfigurācijas datni app.config, pievienojot šādus 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* definējiet ziņojumu klases *source,* ar atbilstošiem identifikatoriem *name*. Katrai klasei piesaistiet vismaz vienu *listeners* failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora *name*, vai definējiet jaunu failu, norādot šādus 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* definējiet atribūtu *trace* ar īpašību *autoflush* un vērtību *true*. Kas norāda, ka rakstīšana failā tiks veikta uzreiz.

```
      <trace autoflush="true" />
</system.diagnostics>
```

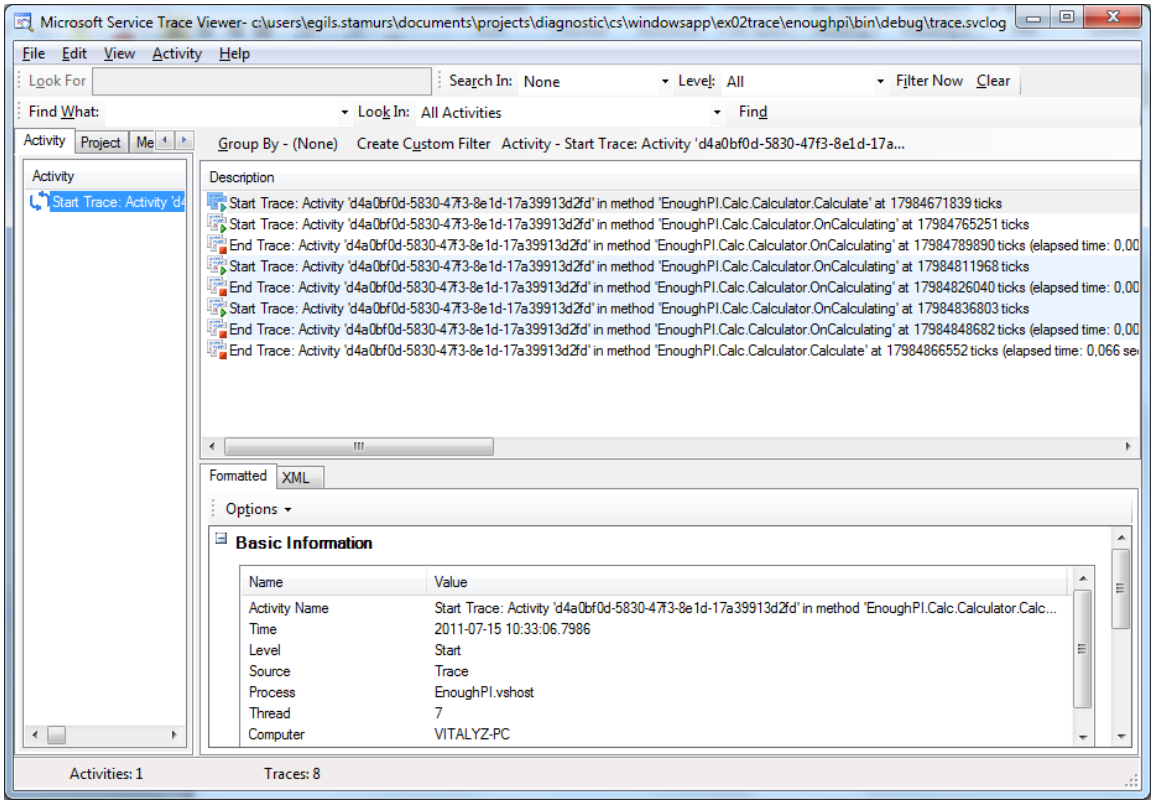
4. Projekta kodā izvietojiet trasēšanas metodes *StartTrace*, iekļaujot trasējamo koda fragmentu *using* operatora figūriekavās, un kā parametru norādot ziņojumu klases *source* identifikatoru name;

```
protected void OnCalculating(CalculatingEventArgs args)
{
    using (TraceUtility.StartTrace(Category.Trace))
    {
            if (Calculating != null)
           Calculating(this, args);
    }
}
```

5. Nepieciešamības gadījumā modificējiet 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";
}
```

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



7.attēls. Trasēšanas rezultāti

3.4. ExceptionUtility

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

1. Iezīmējiet projekta failu *Calc\Calculator.cs* un izpildiet konteksta izvēlnes komandu *View Code*;
2. Pievienojiet vārdtelpu *Diagnostic* faila augšdaļā;

```
using Diagnostic;
```

3. Modificējiet konfigurācijas datni app.config, pievienojot šādus 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* definējiet ziņojumu klasi *source* ar identifikatoru name = *„General”*. Klasei piesaistiet vismaz vienu *listeners* failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora *name*, vai definējot jaunu failu norādot šādus 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* definējiet atribūtu *trace* ar īpašību *autoflush* un vērtību *true*. Kas norāda, ka rakstīšana failā tiks veikta uzreiz.

```
       <trace autoflush="true" />
</system.diagnostics>
```

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

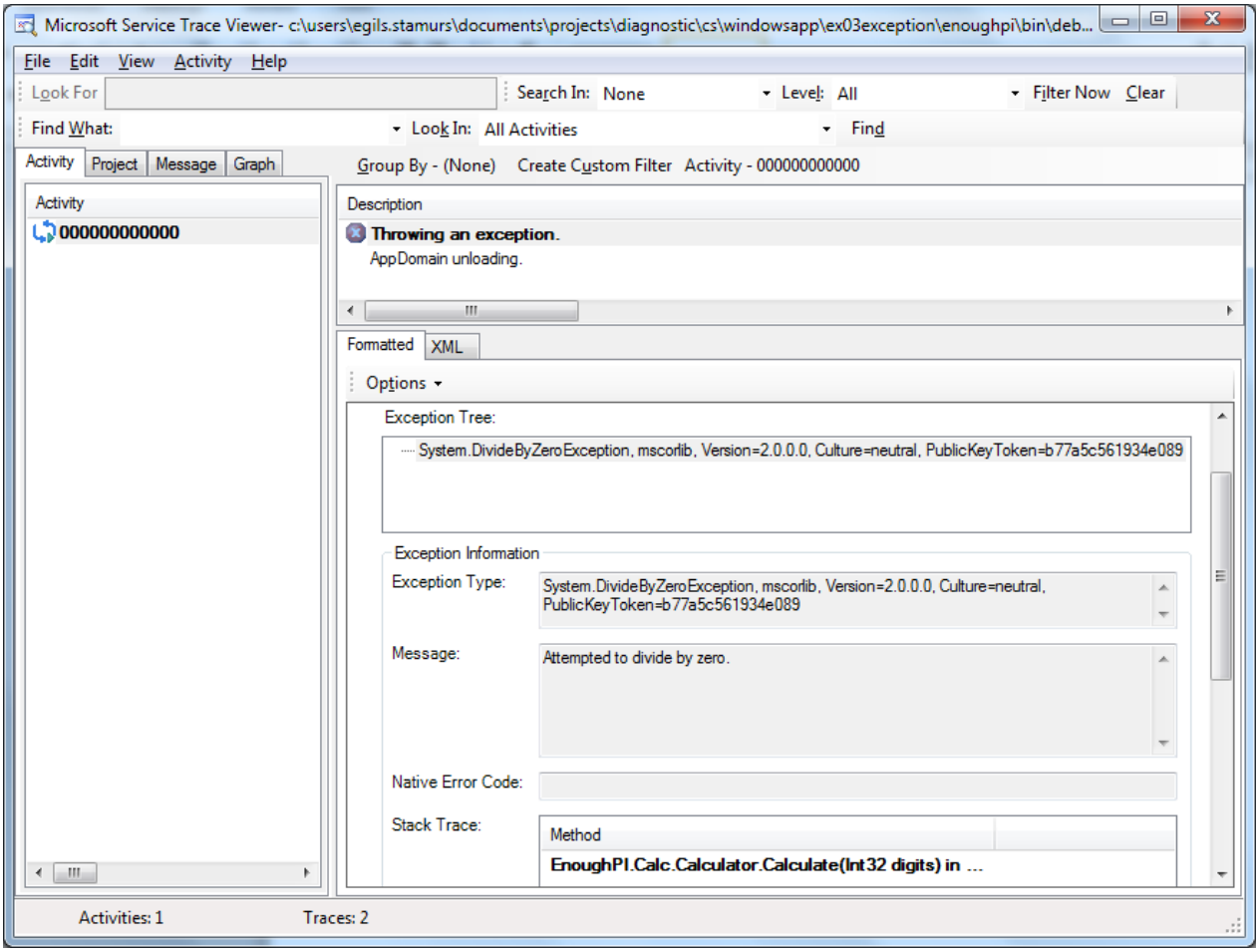
```
ExceptionUtility exceptionWriter = new ExceptionUtility(new LogUtility("EnoughPI"));
```

5. Projekta kodā pēc nepieciešamības izvietojiet izņēmumu rakstīšanas metodes, piemēram *ThrowHelperError* vai citas;

```
protected void OnCalculatorException(CalculatorExceptionEventArgs args)
{
    // TODO: Log exception
    exceptionWriter.ThrowHelperError(args.Exception);

    if (CalculatorException != null)
    CalculatorException(this, args);
}
```

6. Šajā gadījumā netiek specificēta ziņojumu kategorija, bet lietota noklusētā kategorija „General”, tāpēc jāizveido atbilstoša ziņojumu klase *app.config* failā (3.solis);

7. Rezultāta iegūšanai tika radīta izņēmuma situācija un iegūts šāds rezultāts (skat. 8.attēlu).



8.attēls. Izņēmumu žurnalēšanas rezultāti

3.5. LogActivity

Piemērā 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. Iezīmējiet projekta failu *Calc\Calculator*.cs un izpildiet konteksta izvēlnes komandu *View Code*;
2. Pievienojiet vārdtelpu *Diagnostic* faila augšdaļā;

```
using Diagnostic;
```

3. Modificējiet konfigurācijas datni app.config, pievienojot šādus 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* definējiet koplietojamos trasēšanas failus, norādot šādus 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* definējiet ziņojumu klases *source* ar identifikatoriem name = *„General”* un „Activity”. Klasēm piesaistiet vismaz vienu *listeners* failu, norādot kādu no koplietojamajiem failiem pēc tā identifikatora *name*, vai definējiet 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* definējiet atribūtu *trace* ar īpašību *autoflush* un vērtību *true*. Kas norāda, ka rakstīšana failā tiks veikta uzreiz.

```
        <trace autoflush="true" />
</system.diagnostics>
```

4. Failā *Calc\Calculator*.cs izsauciet *LogActivity* metodi *UseDiagnosticTrace*, lai iespējotu aktivitāšu trasēšanu.

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

5. Projekta kodā pēc nepieciešamības izvietojiet aktivitāšu trasēšanas metodes *CreateBoundedActivity*. Norādiet 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 arī lietot metodes *Start*() un *Dispose*();

```
protected void OnCalculatorException(CalculatorExceptionEventArgs args)
{
    LogActivity.CreateBoundedActivity(false).Start("CalculatorException",
    "type");
   if (CalculatorException != null)
      CalculatorException(this, args);

   LogActivity.Current.Dispose();
}
```

6. Asinhrono metožu izsaukumu gadījumā:

- Definējiet *LogActivity* tipa mainīgo klases *Calculator* ietvaros;

```
private LogActivity la;
```

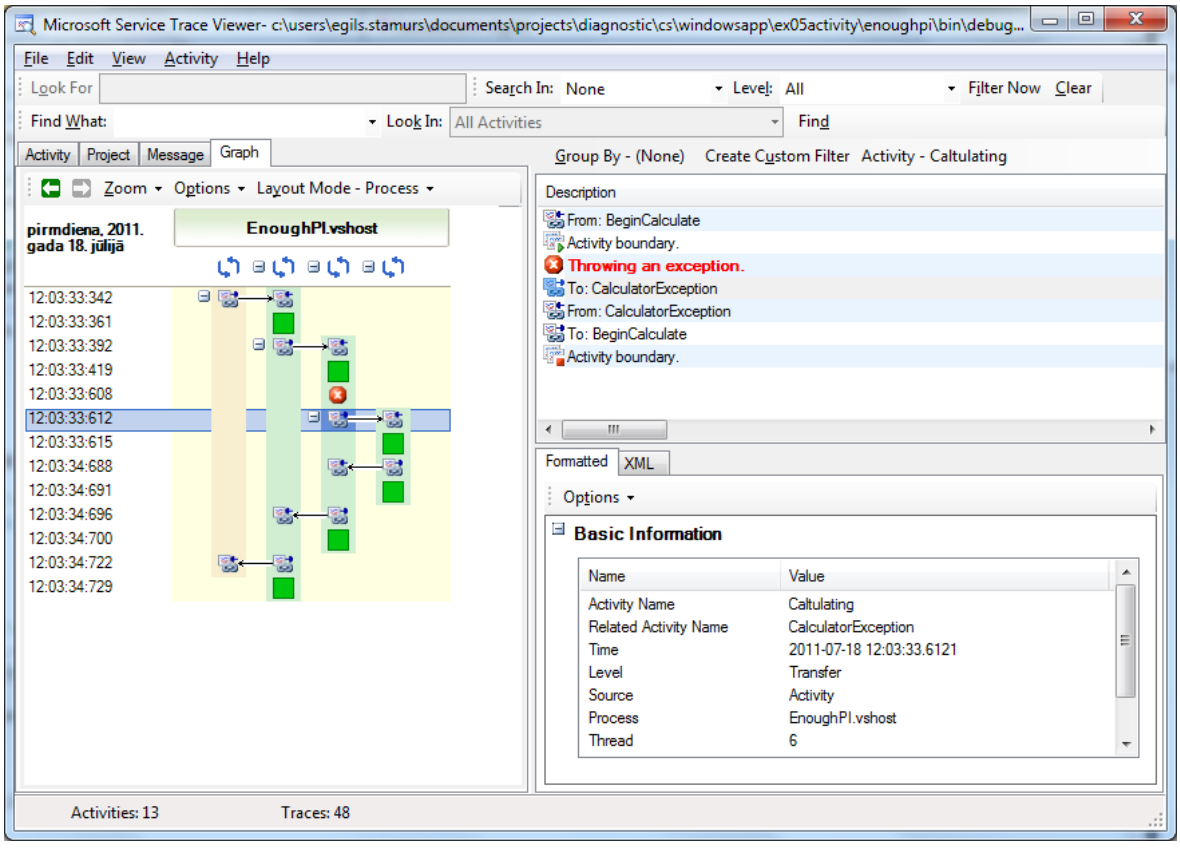
- Izveidojiet aktivitāti *CreateBoundedActivity* asinhronajā izsaukumā un uzsāciet tās trasēšanu izsaucot metodi *Start*();

```
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});
}
```

- Apturet 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();
}
```

7. Trasējot aktivitātes, netiek specificēta ziņojumu kategorija, bet lietotas noklusētās kategorijas „Activity” un „General”, tāpēc izveidojiet atbilstošas ziņojumu klases *app.config* failā (3.solis);
8. Rezultāta iegūšanai tika radīta arī izņēmuma situācija un veikta tās žurnalēšana, izmantojot *ExceptionUtility* un iegūts šāds rezultāts (skat. 9.attēls.).



9.attēls. Aktivitāšu žurnalēšanas rezultāti

3.6. ExtraInformationProvider

Klase *ExtraInformationProvider* nodrošina lietotāja veidotu īpašību iekļaušanu žurnālu failos. Piemērā 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. Pievienojiet klasēm šādas vārdtelpas:

```
using System.Collections.Generic;
using Diagnostic.ExtraInformation;
```

2. Izveidojiet klasi ar *IExtraInformationProvider* interfeisu un implementējiet tā metodi *PopulateDictionary*. Metode aizpilda *Dictionary* tipa mainīgo ar īpašībām un to vērtībām. Izveidojiet 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);
            }
      }
}
```

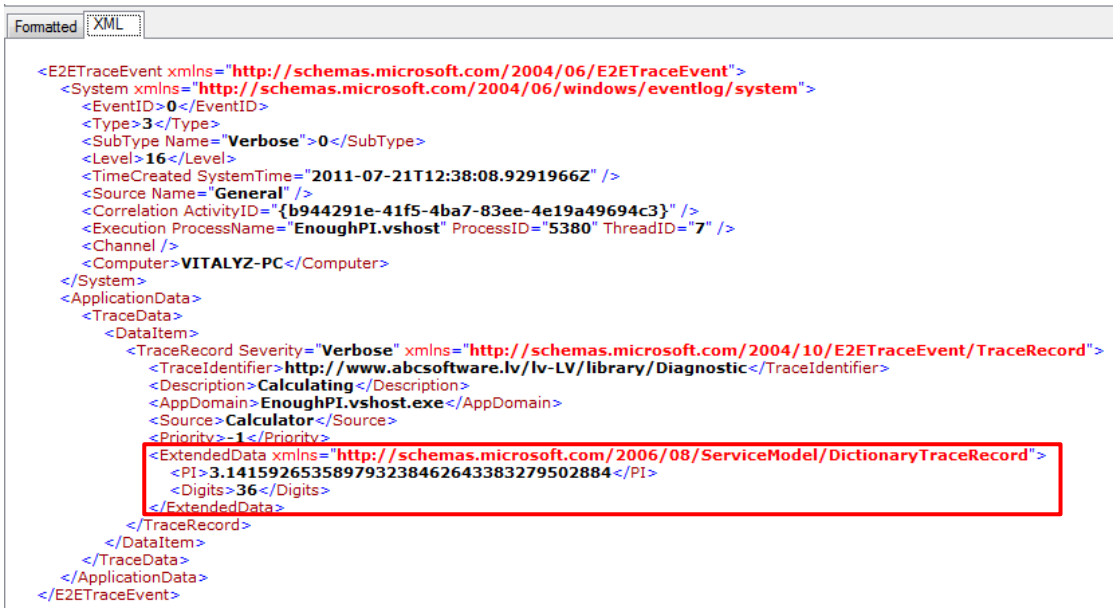
3. Lai ierakstītu žurnalēšanas failā vēlamās īpašības, nepieciešams veikt šādas darbības:
   - Definējiet *Dictionary* tipa mainīgo;
   - Izveidojiet klases eksemplāru un konstruktorā padot failā rakstāmās vērtības;
   - Izsauciet klases eksemplāra metodi *PopulateDictionary* un tās parametrā uzdodiet iepriekš definēto *Dictionary* tipa mainīgo;
   - Izsauciet *LogUtility* metodi *Write* un parametros norādiet ziņojuma tekstu un definēto *Dictionary* tipa mainīgo.

```
Dictionary<string, object> p = new Dictionary<string, object>();
var x = new ExtraInfoProvider(args.Pi, args.Digits);
x.PopulateDictionary(p);
logWriter.Write("Result", p);
```

4. To pašu rezultātu iespējams sasniegt neveidojot atsevišķu klasi *Dictionary* tipa mainīgā aizpildīšanai, bet gan veicot šādas darbības:
   - Definējiet *Dictionary* tipa mainīgo;
   - Pievienojiet mainīgajam nepieciešamās īpašības un to vērtības, izsaucot metodi *Add*;
   - Izsauciet *LogUtility* metodi *Write,* parametros norādot ziņojuma tekstu un definēto *Dictionary* tipa mainīgo.

```
Dictionary<string,object> p = new Dictionary<string,object>();
p.Add("Pi",args.Pi);
logWriter.Write("Result", p);
```

5. Iegūtais rezultāts pievienojot īpašības *Pi* un *Digits* (skat. 10.attēls.).



10.attēls. ExtraInformationProvider pielietošanas rezultāti

3.7. 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, tas mainīsies un tiks izmantota *Enterprise Library*.

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. Atveriet projektu EnoughPI un iezīmējiet katalogu References, izpildot konteksta izvēlnes komandu Add Reference un pievienojiet projektam šādas bibliotēkas (tipiski atrodamas katalogā *„~\Diagnostic\Lib”*):
   - Microsoft.Practices.EnterpriseLibrary.Logging.dll;
   - Microsoft.Practices.EnterpriseLibrary.Common.dll;
   - Microsoft.Practices.ObjectBuilder2.dll.
2. Modificējiet konfigurācijas datni *app.config*, papildus pievienojot šādus 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 norādot *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>
```

3. Izveidojiet *Enterprise Library* konfigurācijas datni, piemēram – *„Enterprise.config”*, un pievienojiet šādus 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* definējiet šādas 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>
```

3.8. 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 papildu ziņojumu žurnalēšanu ar Diagnostic.dll bibliotēku.

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

3.8.1. Klienta konfigurācijas datne

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

- Sadaļu *system.diagnostics* kas satur sadaļas *sharedListeners* (pēc izvēles) un *sources*. Sadaļā *sharedListeners* definējiet koplietojamos trasēšanas failus, norādot šādus 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* definējiet ziņojumu klases *source* ar identifikatoriem *name*. Klasēm piesaistiet 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* definējiet 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* iespējojiet vai atspējojiet ziņojumu žurnalēšanas iespējas;

```
<system.serviceModel>
   <diagnostics>
        <messageLogging logEntireMessage="true"
                        logMessagesAtTransportLevel="true" />
   </diagnostics>
```

- Sadaļā *bindings* uzstādiet 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* piesaistiet servisu – *adress* un norādiet piesaistes tipu – *binding* un tā konfigurāciju – *bindingConfiguration*.

```
<client>
   <endpoint address="http://localhost:64775/Service1.svc"
             binding="basicHttpBinding"
             bindingConfiguration="BasicHttpBinding_IService1"
             contract="ServiceRef.IService1"
             name="BasicHttpBinding_IService1" />
</client>
</system.serviceModel>
```

3.8.2. Servisa konfigurācijas datne

Modificējiet servisa konfigurācijas datni *web.config*, izmantojot WCF *Configuration Editor* vai manuāli pievienojot šādus atribūtus:

- Sadaļu *system.diagnostics* kas satur sadaļas *sharedListeners* (pēc izvēles) un *sources*. Sadaļā *sharedListeners* definējiet koplietojamos trasēšanas failus, norādot šādus 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* definējiet ziņojumu klases *source* ar identifikatoriem *name*. Klasēm piesaistiet 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* definējiet 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* iespējojiet vai atspējojiet ziņojumu žurnalēšanas iespējas;

```
<system.serviceModel>
   <diagnostics>
      <messageLogging logEntireMessage="true"
                      logMessagesAtTransportLevel="true" />
   </diagnostics>
```

- Sadaļā *services* uzstādiet 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* uzstādiet servisa konfigurāciju.

```
<behaviors>
  <serviceBehaviors>
      <behavior name="WcfService.Service1Behavior">
         <serviceMetadata httpGetEnabled="true"/>
         <serviceDebug includeExceptionDetailInFaults="false"/>
      </behavior>
   </serviceBehaviors>
</behaviors>
</system.serviceModel>
```

3.8.3. Žurnalēšana ar Diagnostic.dll

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

- Papildiniet konfigurācijas failus *web.config* un *app.config*;

```
<!--for Diagnostic.dll logUtility-->
<source name ="Activity" switchValue="All">
        <listeners>
                <add name="ServiceModelTraceListener"></add>
        </listeners>
</source>
```

- Pievienojiet klienta un servisa risinājumiem Diagnostic.dll bibliotēku un atbilstošas vārdtelpas;
- Izveidojiet žurnalēšanas metodes nepieciešamajās vietās gan servisa, gan klienta pusē;

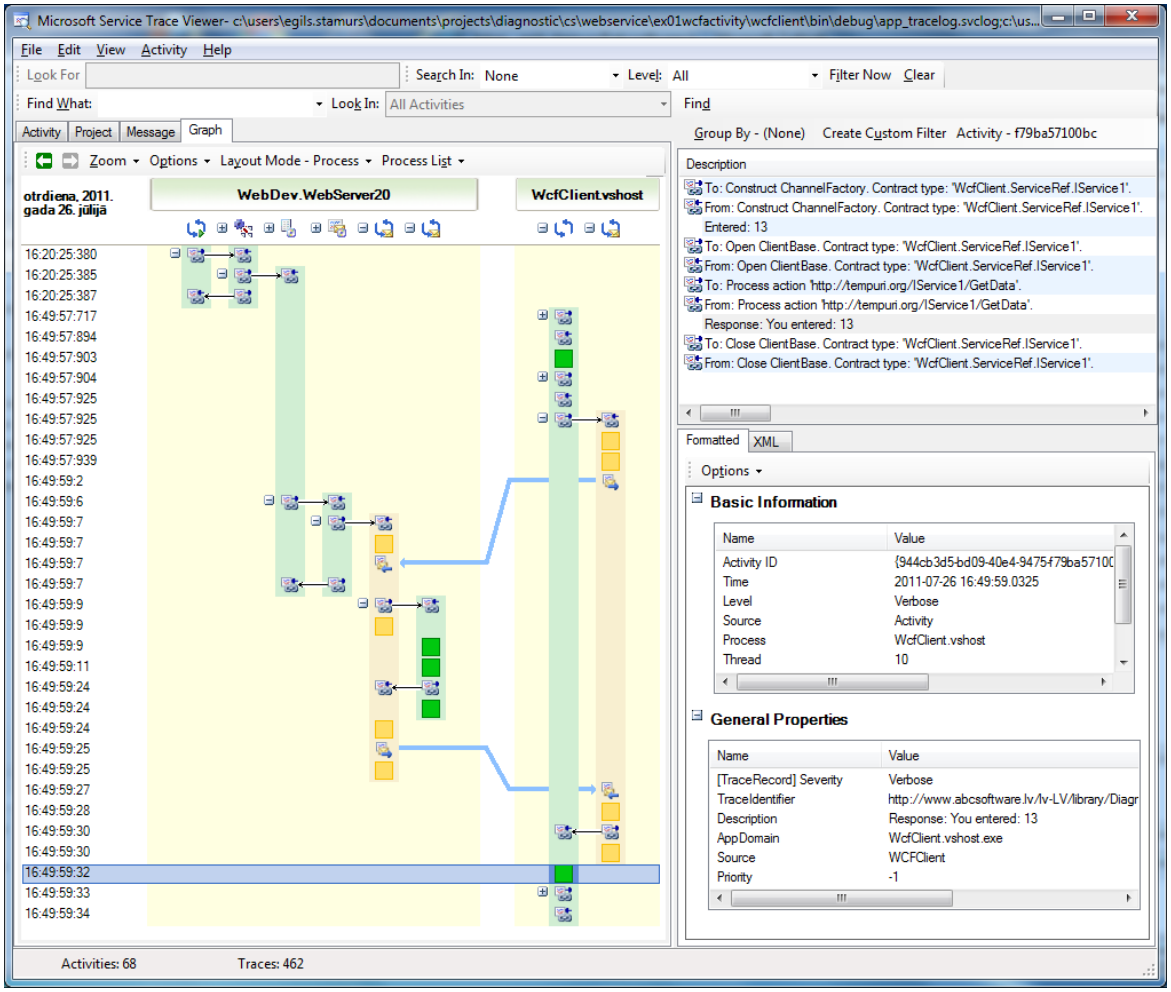
```
LogUtility logWriter = new LogUtility("WCFService");

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;
```

3.8.4. Sinhrons servisa izsaukums

Sinhroni servisa izsaukumi var tikt automātiski apstrādāti bez papildu 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

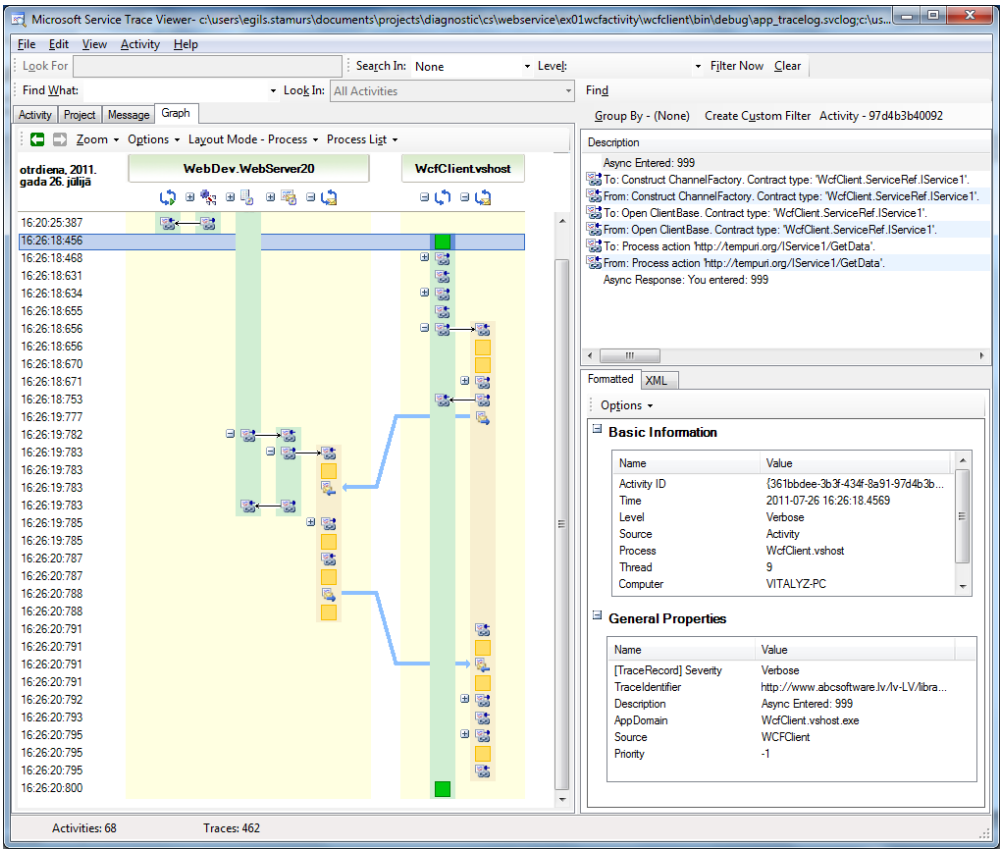
3.8.5. Asinhrons servisa izsaukums

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

3.9. Paplašinājumu palīgklase

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

1. Atveriet projektu *EnoughPI* un iezīmējiet katalogu References, izpildot konteksta izvēlnes komandu *Add Reference*, un pievienojiet projektam IVIS.*Diagnostic.dll* bibliotēku (tipiski atrodama katalogā *„~\Diagnostic\Lib”*), kas satur izmantojamās paplašinājuma metodes.
2. Iezīmējiet projekta failu *Calc\Calculator.cs* un izpildiet konteksta izvēlnes komandu *View Code*;
3. Pievienojiet vārdtelpas *Diagnostic* un *Ivis.Diagnostic* faila augšdaļā;

```
using Diagnostic;
using IVIS.Diagnostics;
```

4. Modificējiet konfigurācijas datni *app.config*, pievienojot šādus 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* definējiet koplietojamos žurnalēšanas failus, norādot šādus atribūtus: atrašanās vieta – *initializeData*, šajā gadījumā lietojuma katalogā, tips – *type*, identifikators – *name*, papildus žurnalēšanas iespējas – *traceOutputOptions* un citu atribūtus 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* definējiet ziņojumu klases *source* ar atbilstošiem identifikatoriem *name*. Katrai klasei piesaistiet 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*.diagnostics definējiet atribūtu *trace* ar īpašību *autoflush* un vērtību true. Kas norāda, ka rakstīšana žurnalēšanas failā tiks veikta uzreiz.

```
       <trace autoflush="true" />
</system.diagnostics>
```

5. Failā *Calc\Calculator*.cs izveidojiet *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");
```

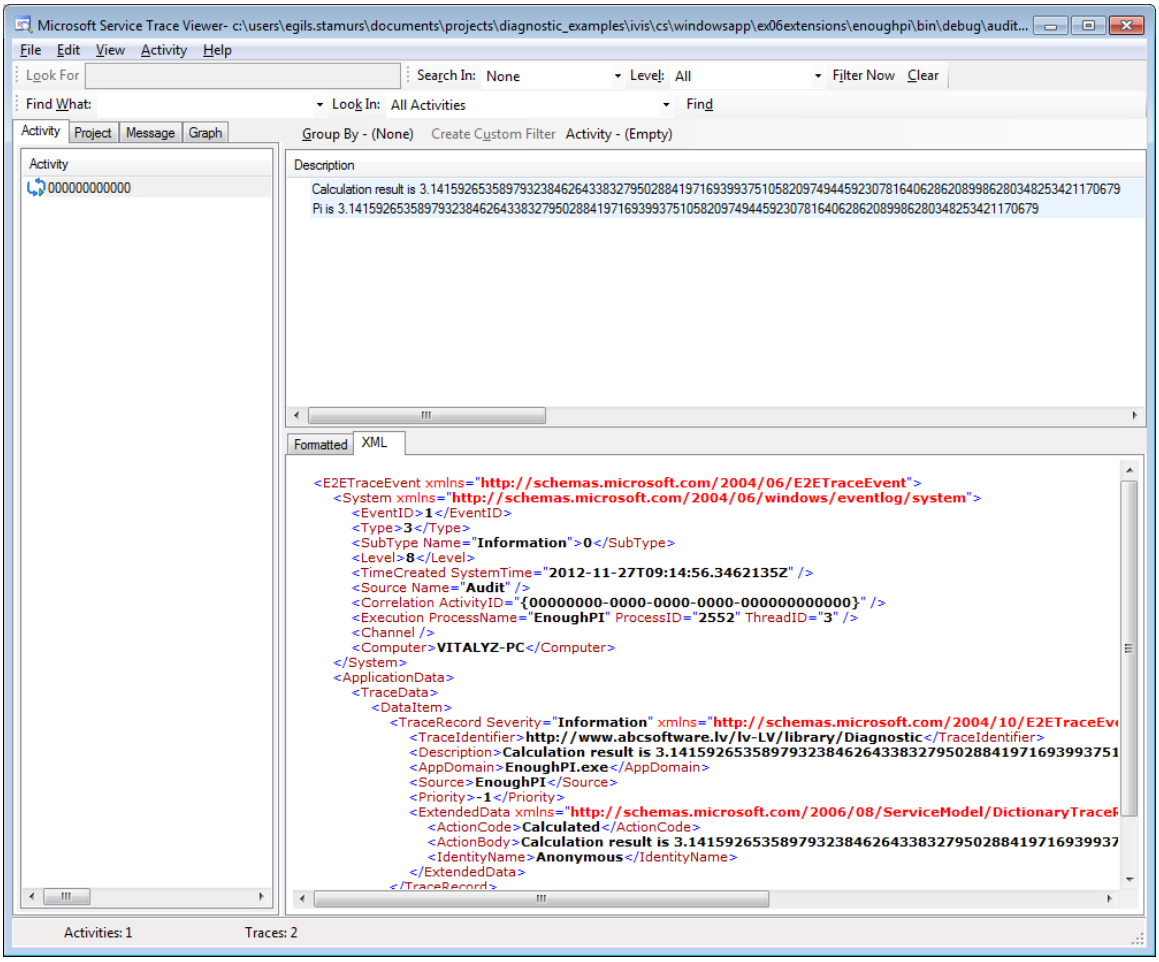
6. Nepieciešamajās vietās pievienojiet projekta kodam nepieciešamās rakstīšanas metodes. Audita rakstīšanai:

```
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));
 }
```

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



13.attēls. Ziņojumu un audita rezultāti

3.9.1. 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. Modificējiet konfigurācijas sadaļas *diagnosticConfiguration* atribūtu *type;*

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

2. Papildiniett 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>
```

3. Ja tiek izmantots .Net 3.5, pievienojiet 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="WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12BasicSecurity
Profile10" 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>
```

4. Ja tiek izmantots .Net 4.0 vai jaunāks, pievienojiet 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>
```

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

4. Diagnostic sekcijas konfigurācija VISS vidēm – instrukcija administratoriem

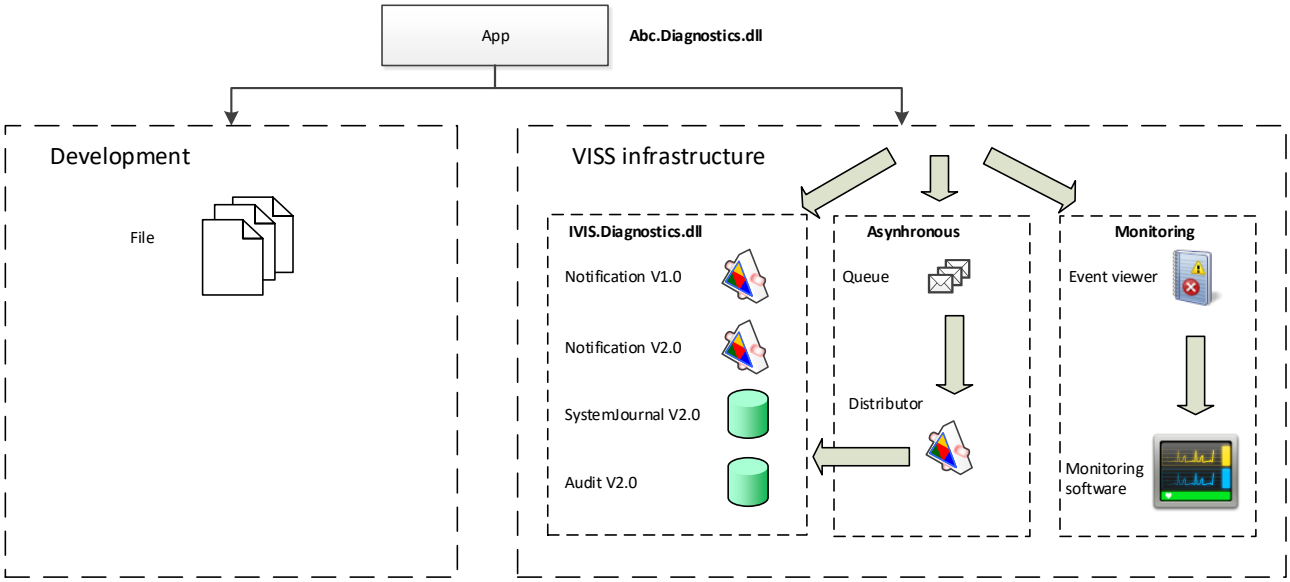
4.1. Konfigurācijas varianti atkarībā no vides

Diagnostikas konfigurācijā mainās atkarība no vides. Paredzēti divi varianti

- izstrāde un testēšana;
- VRAA vide.

Izstrādei un testiem logošana, notifikācijas un audits tiek rakstīti failos.

VRAA vidē logi tiek rakstīti sistēmas žurnālā datubāzē, audits tiek rakstīts audita datubāzē, notifikācijas izmanto notifikācijas servisu. Iespējams, diagnostiku rakstīt *asinhronā* veidā, izmantojot Enterprice Library un rindas (MSMQ un RabbitMQ). Papildus monitoringa programmatūrai iespējams konfigurēt kļūdu rakstīšanu uz windows Event Viewer.



14.attēls. Konfigurācijas varianti atkarība no vides

4.2. Konfigurācija izstrādes un testēšanas vidēm

4.2.1. Konfigurācija izmantojot System.Diagnostics

Konfigurējiet sekcijas lietojuma konfigurācijas datnē:

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics, Version=1.2.0.0"/>
</configSections>
```

Izmantojiet *Abc.Diagnostics.DefaultLogWriter*, lai rakstītu XML formāta:

```
<diagnosticConfiguration type="Abc.Diagnostics.DefaultLogWriter, Abc.Diagnostics"
defaultCategory="category" />
```

Izmantojiet *System.Diagnostics.XmlWriterTraceListener,* lai rakstītu failā:

```
<system.diagnostics>
    <sources>
```

```
        <source name="category" switchValue="All">
            <listeners>
                <add name="listener" initializeData="trace.svclog"
type="System.Diagnostics.XmlWriterTraceListener, System.Diagnostics" />
            </listeners>
        </source>
    </sources>
</system.diagnostics>
```

4.2.2. Konfigurācija izmantojot microsoft enterprise library

Konfigurējiet sekcijas lietojuma konfigurācijas datnē:

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics, Version=1.2.0.0"/>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
    <section name="dataConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Data.Configuration.DatabaseSettings,
Microsoft.Practices.EnterpriseLibrary.Data, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
</configSections>
```

Izmantojiet žurnāla rakstītāju saskaņā ar zemāk redzamo tabulu

ENTRLIB VERSION	TYPE
V3.0	Abc.Diagnostics.EntrLib30LogWriter
V4.0	Abc.Diagnostics.EntrLib40LogWriter
V5.0, V5.0 Upd 1	Abc.Diagnostics.EntrLib50LogWriter
V6.0	Abc.Diagnostics.EntrLib60LogWriter

```
<diagnosticConfiguration type="Abc.Diagnostics.EntrLib40LogWriter, Abc.Diagnostics"/>
```

Konfigurējiet Microsoft Enterprise Library bibliotēkas sadaļu. Izmantojiet *Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.XmlTraceListener*, lai rakstītu failā.

```
<loggingConfiguration name="Logging Application Block" defaultCategory="category">
    <listeners>
      <add name="listener"
type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.XmlTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging"/>
    </listeners>

    <categorySources>
      <add name="category" switchValue="All">
        <listeners>
          <add name="listener"/>
        </listeners>
      </add>
    </categorySources>
</loggingConfiguration>
```

4.2.3. Konfigurācija, izmantojot Serilog

Serilog izmantošanas apraksts žurnalēšanas rakstīšanai datnē ir dots 4.3.5. sadaļā.

4.3. Konfigurācija sekcijas lietojumā - Konfigurācija VISS vidē

Konfigurējiet sekcijas lietojuma konfigurācijas datnē:

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics, Version=1.2.0.0"/>
</configSections>
```

Izmantojiet *IVIS.Diagnostics.IvisLogWriter,* lai rakstītu VISS infrastruktūras formātā:

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

4.3.1. Sistēmas žurnāls V1 (atcelts)

Sekcijā *connectionStrings* pievienojiet elementu *add* ar nosaukumu ***LogConnectionString*** un norādiet savienojuma rindu pie *Log* datubāzes.

```
<connectionStrings>
  <add name="LogConnectionString" connectionString="{$connectionStringToLogDatabase}"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Sekcijā *system.diagnostics* pievienojiet elementu *source* un norādiet vērtību *swithValue*. Pie *source* elementa pievienojiet *LogTraceListener* un norādiet parametru *ApplicationIdentity*.

```
<system.diagnostics>
  <sources>
    <source name="{$LogName}" switchValue="{$LogSwitchValue}">
      <listeners>
         <add name="LogTraceListener" type="IVIS.Diagnostics.LogDatabaseTraceListener,
IVIS.Diagnostics" initializeData="LogConnectionString"
applicationIdentity="{$ApplicationIdenttity}"/>
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.3.2. Sistēmas žurnāls V2

Sekcijā *connectionStrings* pievienojiet elementu *add* ar nosaukumu ***SystemJournalConnectionString*** un norādiet savienojuma rindu pie *SystemJournal* datubāzes.

```
<connectionStrings>
  <add name="SystemJournalConnectionString"
connectionString="{$connectionStringToSystemJournalDatabase}"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Sekcijā *system.diagnostics* pievienojiet elementu *source* un norādiet atribūtam vērtību *swithValue*.

Pie *source* elementa pievienojiet *SystemJournalTraceListener* un norādiet parametru *ApplicationIdentity*.

```
<system.diagnostics>
  <sources>
    <source name="{$LogName}" switchValue="{$LogSwitchValue}">
      <listeners>
         <add name="SystemJournalTraceListener"
type="IVIS.Diagnostics.SystemJournalDatabaseTraceListener, IVIS.Diagnostics"
initializeData="SystemJournalConnectionString" applicationIdentity="{$ApplicationIdenttity}"/>
      </listeners>
    </source>
  </sources>
```

```
</system.diagnostics>
```

4.3.3. Audits V1

Sekcijā *connectionStrings* pievienojiet elementu *add* ar nosaukumu ***DAIRMConnectionString*** un norādiet savienojuma rindu pie *DAIRM* datubāzes.

```
<connectionStrings>
  <add name="DAIRMConnectionString" connectionString="{$connectionStringToDairmDatabase}"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Sekcijā *system.diagnostics* pievienojiet elementu *source* ar nosaukumu ***Audit*** un norādiet vērtību *swithValue* uz ***All***.

Pie *source* elementa pievienojiet *AuditDatabaseTraceListener* un norādiet parametru *ApplicationIdentity*.

```
<system.diagnostics>
  <sources>
    <source name="Audit" switchValue="All">
      <listeners>
        <add name="AuditDatabaseTraceListener"
type="IVIS.Diagnostics.AuditDatabaseTraceListener, IVIS.Diagnostics"
initializeData="DAIRMConnectionString" applicationIdentity="{$ApplicationIdentity}"/>
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.3.4. Audits V2

Sekcijā *connectionStrings* pievienojiet elementu *add* ar nosaukumu ***DAIRM2ConnectionString*** un norādiet savienojuma rindu pie *DAIRM v2* datubāzes.

```
<connectionStrings>
  <add name="DAIRM2ConnectionString" connectionString="{$connectionStringToDairm2Database}"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Sekcijā *system.diagnostics* pievienojiet elementu *source* ar nosaukumu ***Audit*** un norādiet vērtību *swithValue* uz ***All***.

Pie *source* elementa pievienojiet *DairmDatabaseTraceListener* un norādiet parametru *ApplicationIdentity*.

```
<system.diagnostics>
  <sources>
    <source name="Audit" switchValue="All">
      <listeners>
        <add name="DairmDatabaseTraceListener"
type="IVIS.Diagnostics.DairmDatabaseTraceListener, IVIS.Diagnostics"
initializeData="DAIRM2ConnectionString" applicationIdentity="{$ApplicationIdentity}"/>
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.3.5. Audits ar Serilog

4.3.5.1. Jauna lietojuma izstrāde vai esoša, kas izmanto Abc.Diagnostics v1.2.x bibliotēkas, pārkonfigurēšana

Lai rakstītu auditu RabbitMQ rindā, izmantojot Serilog, lietojumi jāveido vai jāpārkonfigurē vismaz uz .Net 4.5.2 versiju.

Audita rakstīšanai uz rindu tiek izmantota RabbitMQ.Client.dll bibliotēka, bet tā var tikt izmantota arī citu funkciju nodrošināšanai, piemēram kontrolieru datu apmaiņai. Šādās situācijas, kad projekts jau satur šo bibliotēku, ir jāsazinās ar komponentes izstrādātāju pirms to aizstāt ar jaunāku vai vecāku versiju un jāveic notikumu maršrutēšanas konfigurēšana.

Lai pieslēgtu lietojumam auditu ir nepieciešams izmantot aktuālākās bibliotēku versijas, bet jāņem vērā to savstarpējās atkarības. Lai atvieglotu bibliotēku savstarpējo atkarību risināšanu var izmantot rīku NuGetPackageExplorer, skatīt https://github.com/NuGetPackageExplorer/NuGetPackageExplorer.

Audita rakstīšanai ir nepieciešamas šādas bibliotēkas:

Diagnostikas bibliotēkas:

- Abc.Diagnostics versija v1.2.6 vai jaunāka (`Install-Package Abc.Diagnostics -Version 1.2.6 -Source https://nexus.vraa.gov.lv/repository/eservices-nuget/`)
- Ivis.Diagnostics v1.2.4 vai jaunāka. (`Install-Package Ivis.Diagnostics -Version 1.2.4 -Source https://nexus.vraa.gov.lv/repository/eservices-nuget/`)
- Viss.Diagnostics.Serilog.dll v1.2.1 vai jaunāka (`Install-Package Viss.Diagnostics.Serilog -Version 1.2.1 -Source https://nexus.vraa.gov.lv/repository/eservices-nuget/`)
  Zemāk uzskaitītas atkarības – instalējot Viss.Diagnostics.Serilog no nuget tiks pievienotas automātiski:
  1. Serilog.dll (piemēram, `Install-Package Serilog -Version 2.9.0 -Source https://www.nuget.org/api/v2/`)
  2. Serilog.Settings.AppSettings.dll (piemēram, `Install-Package Serilog.Settings.AppSettings -Version 2.2.2 -Source https://www.nuget.org/api/v2/`)

Rakstīšanai datnē, izmantojot Serilog, nepieciešams pievienot:

- Serilog.Sinks.File.dll (piemēram, `Install-Package Serilog.Sinks.File -Version 4.1.0 -Source https://www.nuget.org/api/v2/`)

Ziņojumu filtrēšanai pirms audita ierakstīšanas ir nepieciešama bibliotēka:

- Serilog.Expressions.dll (piemēram, `Install-Package Serilog.Filters.Expressions -Version 3.4.0 -Source https://www.nuget.org/api/v2/`)

Rakstīšanai RabbitMQ rindā ir nepieciešamas bibliotēkas:

- Serilog.Sinks.RabbitMQ.dll (piemēram, `Install-Package Serilog.Sinks.RabbitMQ -Version 6.1.0-with-audit03 -Source https://nexus.vraa.gov.lv/repository/eservices-nuget/`)

  Zemāk uzskaitītas atkarības – instalējot Serilog.Sinks.RabbitMQ no nuget tiks pievienotas automātiski:
  1. RabbitMQ.Client.dll (piemēram, `Install-Package RabbitMQ.Client -Version 6.0.0 -Source https://www.nuget.org/api/v2/`)

2. Serilog.Sinks.PeriodicBatching.dll (piemēram, `Install-Package Serilog.Sinks.PeriodicBatching -Version 2.3.1 -Source https://www.nuget.org/api/v2/`)

Pievienojot bibliotēkas no NuGet repozitorijiem, mainīsies arī app.config/web.config datne. Norādītās bibliotēku versijas ir aktuālas apraksta veidošanas brīdī. Patstāvīgi instalējot pakotnes ir jāievēro to savstarpējās versiju atkarības, un katrai no tām nepieciešamā .Net ietvara atkarības.

Audita rakstīšanai jāizmanto Metode „WriteAudit", skat. 2.4.2.1.3. paragrāfu.

Lietojuma konfigurēšanu rakstīšanai rindā vai datnēs skatīt nākamajās nodaļās.

4.3.5.2. Lietojuma konfigurēšana audita rakstīšanai datnē ar Serilog

Sekcijā *configSections* pievienojiet elementu ar nosaukumu ***diagnosticConfiguration***.

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics"/>
</configSections>
```

Aizvietojiet elementu *diagnosticConfiguration*, piemēram:

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

ar elementu *diagnosticConfiguration*:

```
<diagnosticConfiguration type="Viss.Diagnostics.Serilog.VissSerilogWriter,
Viss.Diagnostics.Serilog" />
```

Sekcijā *appSettings*, norādiet parametrus Serilog bibliotēkai (https://github.com/serilog/serilog-settings-appsettings):

```
<appSettings>
    <add key="serilog:using:File" value="Serilog.Sinks.File" />

    <add key="serilog:audit-to:File.path" value="c:\Logs\cfg-audit.json" />
    <add key="serilog:audit-to:File.formatter" value="Viss.Diagnostics.Serilog.JsonFormatter,
Viss.Diagnostics.Serilog" />
  </appSettings>
```

Auditējamo ziņojumu filtrēšanai pirms ierakstīšanas, konfigurācijai pievienojiet parametrus:

```
<appSettings>
    <add key="serilog:using:FilterExpressions" value="Serilog.Expressions" />
    <add key="serilog:filter:ByIncludingOnly.expression" value="@Properties['auditLogEntry'] is
not null"/>
  </appSettings>
```

Minētajā piemērā aizvietojot `is not null` ar `is null` audita ieraksta datne tiks izveidota, bet rindā un datnē audita notikumi netiks ierakstīti, jo tie neatbildīs norādītajam kritērijam.

4.3.5.3. Lietojuma konfigurēšana audita rakstīšana RabbitMQ rindā

Lai rakstītu auditu RabbitMQ rindā, lietotnei ir jāizmanto vismaz .Net 4.6.1 ietvars.

Sekcijā *configSections* pievienojiet elementu ar nosaukumu ***diagnosticConfiguration***.

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics"/>
</configSections>
```

Aizvietojiet elementu *diagnosticConfiguration*, piemēram:

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

ar elementu *diagnosticConfiguration*:

```
<diagnosticConfiguration type="Viss.Diagnostics.Serilog.VissSerilogWriter,
Viss.Diagnostics.Serilog" />
```

Ir jāpapildina sekciju *appSettings* ar RabbitMQ rindas parametriem:

```
<appSettings>
    <add key="serilog:using:RabbitMQ" value="Serilog.Sinks.RabbitMQ"/>
    <add key="serilog:audit-to:RabbitMQ.hostname" value="host.abc"/>
    <add key="serilog:audit-to:RabbitMQ.vHost" value="rabbitHostName"/>
    <add key="serilog:audit-to:RabbitMQ.username" value="dairm2"/>
    <add key="serilog:audit-to:RabbitMQ.password" value="fTr%45^YR&^4"/>
    <add key="serilog:audit-to:RabbitMQ.exchange" value="dairm2_exchange_name"/>
    <add key="serilog:audit-to:RabbitMQ.formatter"
value="Viss.Diagnostics.Serilog.JsonFormatter, Viss.Diagnostics.Serilog"/>
</appSettings>
```

Sīkāku parametru aprakstu var atrast https://github.com/gekiss/serilog-sinks-rabbitmq/blob/master/README.md

Ja nokonfigurējāt audita rakstīšanu arī datnē atbilstoši iepriekš aprakstītajam, tad tagad audits tiks rakstīts gan datnē, gan arī RabbitMQ.

4.3.5.3.1. TLS pieslēguma konfigurācija

Aktivizējot TLS pieslēgumu sekcijā *appSettings* jāpieliek parametrus:

```
<appSettings>
    ...
    <add key="serilog:audit-to:RabbitMQ.sslEnabled" value="true"/>
</appSettings>
```

Atkarīgi no izmantojamā TLS sertifikāta un porta jāpieliek nepieciešamos parametrus:

```
<appSettings>
    ...
    <add key="serilog:audit-to:RabbitMQ.sslServerName" value="host.abc"/>
    <add key="serilog:audit-to:RabbitMQ.sslVersion" value="Tls3"/>
    <add key="serilog:audit-to:RabbitMQ.sslAcceptablePolicyErrors" value="None"/>
    <add key="serilog:audit-to:RabbitMQ.sslCheckCertificateRevocation" value="false"/>
</appSettings>
```

Parametru apraksts:

- `sslEnabled` – norāda ka izmanto TLS pieslēgušu;
- `sslServerName` – servera nosaukums (CA);
- `sslAcceptablePolicyErrors` – pielaupāmas kļūdas. Enum `SslPolicyErrors`. https://learn.microsoft.com/en-us/dotnet/api/system.net.security.sslpolicyerrors;
- `sslVersion` – pieļaujamie protokoli. Enum `SslProtocols`. https://learn.microsoft.com/en-us/dotnet/api/system.security.authentication.sslprotocols ;
- `sslCheckCertificateRevocation` – norāda ka pārbaudīt sertifikātā atcelšanu.

4.3.5.4. Žurnalēšanas notikumu maršrutēšana

Izmantojot `RoutedLogWriter` iespējams nokonfigurēt lietojumu, lai notikumus no dažādām kategorijām, piemēram, Audit, Log, Error, General apstrādā dažādi rakstītāji, piemēram, LogFlatFileTraceListener, XmlWriterTraceListener vai citi.

Sekcijā *configSections* pievienojiet elementu ar nosaukumu ***diagnosticConfiguration***.

```
<configSections>
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics"/>
</configSections>
```

Aizvietojiet elementu *diagnosticConfiguration*, piemēram:

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

ar šādu elementu *diagnosticConfiguration,* norādot nepieciešamos filtrus, piemēram*:*

```
<diagnosticConfiguration type="Abc.Diagnostics.RoutedLogWriter, Abc.Diagnostics">
   <filters>
      <filter categories="Audit" type="Viss.Diagnostics.Serilog.VissSerilogWriter,
Viss.Diagnostics.Serilog" applicationIdentity="testing.app.my"/>
      <filter categories="Log" type="IVIS.Diagnostics.IvisLogWriter, IVIS.Diagnostics"/>
      <filter categories="Audit,*" type="Abc.Diagnostics.DefaultLogWriter, Abc.Diagnostics"/>
   </filters>
</diagnosticConfiguration>
```

Šāda konfigurācija nosaka, ka:

- Notikumi no kategorijas Audit tiks rakstīti izmantojot Serilog uz rindu vai datnē;
- Notikumus no kategorijas Log būs iespējams apstrādāt ar kādu no IVIS.Diagnostics nodrošinātajiem rakstītājiem:
  - SystemJournalDatabaseTraceListener – sistēmas žurnāla rakstīšana SQL datu bāzē;
  - LogEventLogTraceListener – notikumu rakstīšana event logā;
  - LogFlatFileTraceListener – notikumu rakstīšana teksta datnē;
  - NotificationService2TraceListener – notifikāciju sūtīšana;
  - u.c.
- Notikumus no kategorijas Audit un visām citām iepriekš nenorādītajām kategorijām varēs apstrādāt ar kādu no šiem rakstītājiem:
  - System.Diagnostics.XmlWriterTraceListener – notikumu rakstīšana svclog datnēs;
  - u.c.

4.3.5.4.1. Konfigurācijas un koda piemērs:

Konsoles lietojumam pievainotas šādas bibliotēkas:

```
<packages>
  <package id="Abc.Diagnostics" version="1.2.6" targetFramework="net461" />
  <package id="Ivis.Diagnostics" version="1.2.4" targetFramework="net461" />
  <package id="RabbitMQ.Client" version="6.0.0" targetFramework="net461" />
  <package id="Serilog" version="2.9.0" targetFramework="net461" />
  <package id="Serilog.Expressions" version="3.4.0" targetFramework="net461" />
  <package id="Serilog.Settings.AppSettings" version="2.2.2" targetFramework="net461" />
  <package id="Serilog.Sinks.File" version="4.1.0" targetFramework="net461" />
  <package id="Serilog.Sinks.PeriodicBatching" version="2.2.1" targetFramework="net461" />
  <package id="Serilog.Sinks.RabbitMQ" version="6.1.0-with-audit03" targetFramework="net461" />
  <package id="Viss.Diagnostics.Serilog" version="1.2.1" targetFramework="net461" />
</packages>
```

Konsoles lietojuma kods – veic notikumu rakstīšanu Audit, Log un General kategorijās:

```
        Abc.Diagnostics.DiagnosticTools.LogUtil.WriteAudit("test.new.net.audit.action", "Audits
no .net 4.6.1 lietojuma " + DateTime.Now.ToString(), -1, null, null, "Audit.test.App");
            Abc.Diagnostics.DiagnosticTools.LogUtil.Write("test log msg " +
DateTime.Now.ToString(), "Log", 1, -1, System.Diagnostics.TraceEventType.Error, null,
Guid.NewGuid());
            Abc.Diagnostics.DiagnosticTools.LogUtil.Write("test General msg " +
DateTime.Now.ToString(), "General", 1, -1, System.Diagnostics.TraceEventType.Error, null,
Guid.NewGuid());
```

Konsoles lietojuma konfigurācija:

```
<configuration>
    <configSections>
        <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics"/>
    </configSections>
```

```
    <diagnosticConfiguration type="Abc.Diagnostics.RoutedLogWriter, Abc.Diagnostics">
        <filters>
            <filter categories="Audit" type="Viss.Diagnostics.Serilog.VissSerilogWriter,
Viss.Diagnostics.Serilog" applicationIdentity="testing.app.my"/>
            <filter categories="Log" type="IVIS.Diagnostics.IvisLogWriter, IVIS.Diagnostics"/>
            <filter categories="Audit,*" type="Abc.Diagnostics.DefaultLogWriter,
Abc.Diagnostics"/>
        </filters>
    </diagnosticConfiguration>

    <appSettings>
        <add key="serilog:using:RabbitMQ" value="Serilog.Sinks.RabbitMQ"/>
        <add key="serilog:audit-to:RabbitMQ.hostname" value="ivis-2k11.abc"/>
        <add key="serilog:audit-to:RabbitMQ.vHost" value="testRabbitHost"/>
        <add key="serilog:audit-to:RabbitMQ.username" value="user123"/>
        <add key="serilog:audit-to:RabbitMQ.password" value="password123"/>
        <add key="serilog:audit-to:RabbitMQ.exchange" value="DAIRM2Exchange"/>
        <add key="serilog:audit-to:RabbitMQ.formatter"
value="Viss.Diagnostics.Serilog.JsonFormatter, Viss.Diagnostics.Serilog"/>
    </appSettings>

    <system.diagnostics>
        <trace autoflush="true"/>
        <sources>
            <source name="Audit" switchValue="All">
                <listeners>
                    <add name="AuditXmlTraceListener"/>
                </listeners>
            </source>

            <source name="Log" switchValue="All">
                <listeners>
                    <add name="LogListener" type="IVIS.Diagnostics.LogFlatFileTraceListener,
IVIS.Diagnostics" initializeData="log.txt" />
                </listeners>
            </source>

            <source name="General" switchValue="All">
                <listeners>
                    <add name="LogListener" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="general.svclog" />
                </listeners>
            </source>
        </sources>
        <sharedListeners>
            <add name="AuditXmlTraceListener" type="System.Diagnostics.XmlWriterTraceListener"
traceOutputOptions="DateTime" initializeData="viss_audit.svclog"/>
        </sharedListeners>
    </system.diagnostics>
</configuration>
```

4.3.5.5. Papildu auditējamie parametri

Atkarībā no lietotnes realizācijas platformas, Serilog bibliotēkai ir iespējams norādīt auditā iekļaut papildu parametrus:

- Kopējie (https://github.com/serilog/serilog-enrichers-environment, https://github.com/serilog/serilog-enrichers-thread, https://github.com/serilog/serilog-enrichers-process);
- ASP.NET (https://github.com/serilog-web/classic);
- MVC (https://github.com/serilog-web/classic-mvc);
- WebApi (https://github.com/serilog-web/classic-webapi);
- ASPNetCore (https://github.com/serilog/serilog-aspnetcore).

Kopējie iespējamie parametri:

```
<appSettings>
    <add key="serilog:using:Process" value="Serilog.Enrichers.Process"/>
    <add key="serilog:using:Thread" value="Serilog.Enrichers.Thread"/>
    <add key="serilog:using:Environment" value="Serilog.Enrichers.Environment"/>
    <add key="serilog:enrich:WithProcessId"/>
    <add key="serilog:enrich:WithProcessName"/>
    <add key="serilog:enrich:WithThreadId"/>
    <add key="serilog:enrich:WithMachineName"/>
    <add key="serilog:enrich:WithEnvironmentUserName"/>
</appSettings>
```

ASP.NET iespējamie parametri:

```
<appSettings>
    <add key="serilog:using:SerilogWeb.Classic" value="SerilogWeb.Classic"/>
    <add key="serilog:enrich:WithClaimValue.claimProperty" value="MyClaimPropertyName"/>
    <add key="serilog:enrich:WithHttpRequestClientHostIP"/>
    <add key="serilog:enrich:WithHttpRequestClientHostName"/>
    <add key="serilog:enrich:WithHttpRequestId"/>
    <add key="serilog:enrich:WithHttpRequestNumber"/>
    <add key="serilog:enrich:WithHttpRequestRawUrl"/>
    <add key="serilog:enrich:WithHttpRequestTraceId"/>
    <add key="serilog:enrich:WithHttpRequestType"/>
    <add key="serilog:enrich:WithHttpRequestUrl"/>
    <add key="serilog:enrich:WithHttpRequestUrlReferrer"/>
    <add key="serilog:enrich:WithHttpRequestUserAgent"/>
    <add key="serilog:enrich:WithHttpSessionId"/>
    <add key="serilog:enrich:WithUserName"/>
</appSettings>
```

MVC iespējamie parametri:

```
<appSettings>
    <add key="serilog:using:SerilogWeb.Classic.Mvc" value="SerilogWeb.Classic.Mvc"/>
    <add key="serilog:enrich:WithMvcActionName"/>
    <add key="serilog:enrich:WithMvcControllerName"/>
    <add key="serilog:enrich:WithMvcRouteData"/>
    <add key="serilog:enrich:WithMvcRouteTemplate"/>
</appSettings>
```

WebApi iespējamie parametri:

```
<appSettings>
    <add key="serilog:using:SerilogWeb.Classic.WebApi" value="SerilogWeb.Classic.WebApi"/>
    <add key="serilog:enrich:WithWebApiActionName" />
    <add key="serilog:enrich:WithWebApiControllerName" />
    <add key="serilog:enrich:WithWebApiRouteData" />
    <add key="serilog:enrich:WithWebApiRouteTemplate" />
</appSettings>
```

Pievienojiet bibliotēku:

- SerilogWeb.Classic (`Install-Package SerilogWeb.Classic -Version 4.2.42 -Source https://www.nuget.org/api/v2/`)

un norādiet papildu parametrus ASP.NET lietotnes konfigurācijā:

```
<appSettings>
    <add key="serilog:using:SerilogWeb.Classic" value="SerilogWeb.Classic"/>
    <add key="serilog:enrich:WithHttpRequestClientHostIP"/>
    <add key="serilog:enrich:WithHttpRequestUserAgent"/>
</appSettings>
```

Tiek iegūts RabbitMQ Serilog audita ziņojums, ja tiek izmantota tīmekļa lietotne, piemēram:

```
{
  "Timestamp": "2018-09-26T07:51:52.1919403Z",
  "MessageTemplate": "",
```

```
  "Level": "Information",
  "Properties": {
    "messageId": "cbe531e6-1c17-4c42-a271-f69d1d98ab0c",
    "HttpRequestId ": null,
    "HttpRequestClientHostIP": null,
    "auditLogEntry": {
      "sessionId": "504B3A333138313236373538353835382D55523A343030303333330303030303126092018",
      "timestamp": "2018-09-26T07:51:52.1919403Z",
      "activityId": "00000000-0000-0000-0000-000000000000",
      "messageId": "31e8bf86-35d4-410f-9a39-cce7442f44b3",
      "eventId": 0,
      "description": "URN:IVIS:100001:XSD-ErrorReport-IVISErrorReport-v1-1-TYPE-SystemMetadata",
      "serviceMetadata": { "machineName": "APP1-DEV-VRAA" },
      "endpointIdentitfier": "IdentitySelector.LVP.STS",
      "details": "",
      "sender": {
        "senderId": "PK:31812675858-UR:40003300001",
        "senderType": 3,
        "senderMetadata": {
          "firstName": "Custom",
          "lastName": "Tester"
        }
      },
      "actionType": "IdentityRequestResponse",
      "subjects": [
        {
          "subjectId": "192.168.100.163",
          "subjectType": "IPAddress"
        }
      ]
    }
  }
}
```


4.3.5.6. Lietojumu, kas izmanto v1.0.x bibliotēkas pārkonfigurācija uz auditēšanu ar Serilog

Lai novirzītu audita rakstīšanu RabbitMQ rindā lietotnēm, kuras nav paredzēts pārstrādāt, un kuras tika veidotas uz .NET2.0, .NET3.0, .NET3.5, .NET4.0, .NET4.5 bāzes, izmantojot Diagnostics v1.0.x bibliotēkas ir jāveic šādas darbības:

1. Atjaunojiet žurnalēšanas bibliotēkas šādām vai jaunākām v1.0.x versijām (pieejamas VRAA nuget https://nexus.vraa.gov.lv/repository/eservices-nuget /):
   i. Abc.Diagnostics v1.0.16 (Diagnostic.dll);

```
Install-Package Abc.Diagnostics -Version 1.0.16 -Source
https://nexus.vraa.gov.lv/repository/eservices-nuget/
```

   ii. Ivis.Diagnostics v1.0.12-rc03 (IVIS.Diagnostics.dll).

```
Install-Package IVIS.Diagnostics -Version 1.0.12-rc03 -Source
https://nexus.vraa.gov.lv/repository/eservices-nuget/
```

2. Pievienojiet Viss.Diagnostics.Serilog bibliotēka ar šādu vai jaunāku v1.0.x versiju (pieejama VRAA nuget https://nexus.vraa.gov.lv/repository/eservices-nuget/):
   i. Viss.Diagnostics.Serilog v1.0.0-rc03 (Viss.Diagnostics.Serilog.dll).
3. Pievienojiet šādas Serilog un Rabbit bibliotēkas:
   i. RabbitMQ.Client v4.1.3;
   ii. Serilog v2.7.1;
   iii. Serilog.Filters.Expressions v2.0.0;
   iv. Serilog.Settings.AppSettings v2.2.2;
   v. Serilog.Sinks.PeriodicBatching v2.1.1;
   vi. Serilog.Sinks.RabbitMQ v2.0.3-with-audit00;
   vii. Superpower v2.1.0;

viii. Serilog.Sinks.File v4.0.0.

4. Lai vecajām lietotnēm nokonfigurētu rakstīt RabbitMQ rindā *tikai* **auditu**, bet pārējo žurnalēšanas konfigurāciju neskartu, izmainiet sekcija *diagnosticConfiguration*:

```
<diagnosticConfiguration type="Diagnostic.RoutedLogWriter, Diagnostic"
defaultCategory="category">
    <filters>
        <filter categories="Audit" type="Viss.Diagnostics.Serilog.VissSerilogWriter,
Viss.Diagnostics.Serilog" />
        <filter categories="*" type="Diagnostic.DefaultLogWriter, Diagnostic" />
    </filters>
</diagnosticConfiguration>
```

Ja auditu ir nepieciešams novirzīt rakstīšanai arī svclog datnē, ir jāizmaina filtrēšanas nosacījumi uz šādiem, papildus informāciju par ziņojumu maršrutēšanu skatīt 4.3.5.3.1. paragrāfā:

```
<filter categories="*,Audit" type="Diagnostic.DefaultLogWriter, Diagnostic" />
```

5. konfigurējiet savienojumu ar rindu, skatīt 4.3.5.3.paragrāfā.
6. Ar konfigurācijas palīdzību pārslēdziet lietojumu, lai izmantotu vismaz .NET4.5.2. Katra lietojuma konfigurācija var atšķirties atkarībā no tā tipa un funkcionalitātes, piemēram:
    i. konsoles lietojumā:

```
<configuration>
    <startup>
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"/>
    </startup>
</configuration>
```

    i. web lietojumā:

```
<configuration>
    <compilation targetFramework="4.0">
    ...
</configuration>
```

4.3.6. Notifikācija V1

Sekcijā *system.diagnostics* pievienojiet elementu *source* ar nosaukumu ***Notification*** un norādiet vērtību *swithValue* uz ***All***.

Pie *source* elementa pievienojiet *NotificationTraceListener* un norādiet parametrus *host, TransformationUrn, MessageTitle*.

```
<system.diagnostics>
  <sources>
    <source name="Notification" switchValue="All">
      <listeners>
        <add name="NotificationTraceListener"
type="IVIS.Diagnostics.NotificationServiceTraceListener, IVIS.Diagnostics"
initializeData="https://{host}/Notification/v1-0/soap11"
transformationSchemaURN="{$TransformationUrn}" defaultMessageTitle="{$MessageTitle}" />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.3.7. Notifikācija V2

Sekcijā *system.diagnostics* pievienojiet elementu *source* ar nosaukumu ***Notification*** un norādiet vērtību *swithValue* uz ***All***.

Pie *source* elementa pievienojiet *Notification2TraceListener* un norādiet parametrus *TransformationUrn, MessageTitle*.

```
<system.diagnostics>
  <sources>
    <source name="Notification" switchValue="All">
      <listeners>
        <add name="Notification2TraceListener"
type="IVIS.Diagnostics.NotificationService2TraceListener, IVIS.Diagnostics"
initializeData="NotificationEndpoint" defaultMessageTitle="{$MessageTitle}" />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

Konfigurējiet NotificationEndpoint pie notifikācijas servisa V2, saskaņā WCF vadlīnijām.

4.4. Žurnalēšana sekošanas programmatūrai

System.diagnostics sekcijā pievienojiet *source* elementu un norādiet *swithValue* uz **Error**. *Source* elementam pievienojiet *LogEventLogTraceListener,* norādot parametru *initializeData*.

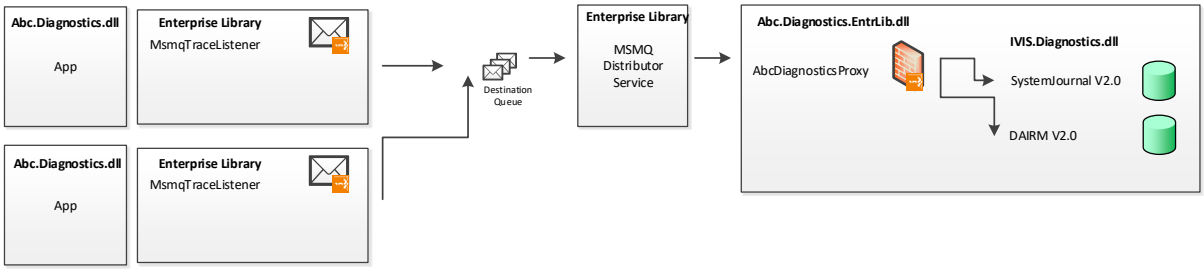
```
<system.diagnostics>
  <sources>
    <source name="{$LogName}" switchValue="Error">
      <listeners>
        <add name="LogEventLogTraceListener" type="IVIS.Diagnostics.LogEventLogTraceListener,
IVIS.Diagnostics" initializeData="TraceListenerLog" />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

Ja notikumu žurnāla avots, kas ir saistīts ar EventLogTraceListener nepastāv, tad tiek izveidots jauns notikumu avots. Lai izveidotu notikumu avotu operētājsistēmā Windows Vista, Windows XP Professional vai Windows Server 2003, jums ir jābūt administratora tiesībām.

Lai izvairītos no iespējas rakstīt lielu datu apjomu, notikumu žurnālam EventLogTraceListener neizsniedz izvēles datus, kas definēti raceOutputOptions parametrā.

4.5. Asinhronā žurnalēšana

Asinhronā logošana un audita rakstīšana ļauj atslogot datubāzes serverus, tādējādi uzlabojot sistēmas ātrdarbību.



15.attēls. Asinhronas logošanas un audita diagramma

4.5.1. Asinhronas logošanas konfigurācijas scenārija izvēle

Iespējami dažādi konfigurācijas scenāriji, kuri atkarīgi no .NET Framework un Enterprise Library versijām. Šajā dokumenta tiks izskatīti izplatītākie.

Izvēles scenārijs:

1. Aizejiet uz programmas palaišanas mapi;

2. Meklējiet datni *Diagnostic.dll* vai *Abc.Diagnostics.dll;*
3. Ja ir *Diagnostic.dll*, tad konfigurācija būs kā .NET3.5 projektiem, ja tomēr ir *Abc.Diagnostics.dll*, tad konfigurācijai jābūt kā .NET4.5 projektiem;
4. Scenārijs kad .NET4.5 projektiem nokonfigurē Enterprise Library 5.0 nav izskatīts.

4.5.2. Asinhronā logošana .NET4.5 projektiem, izmantojot Microsoft Enterprise Library 6.0

Pievienojiet *configSections* elementu lietojuma konfigurācijas datnē, kā pirmo elementu pēc *configuration* elementa:

```
<configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics, Version=1.2.0.0"/>
</configSections>
```

Nokonfigurējiet ABC diagnostics konfigurācijas sekciju *diagnosticConfiguration*

```
  <diagnosticConfiguration type="Abc.Diagnostics.EntrLib.EntrLibLogWriter,
Abc.Diagnostics.EntrLib60, Version=1.0.0.0" applicationIdentity="{$ApplicationIdentity}" />
```

Nokonfigurējiet Enterprise Library logošanas konfigurācijas sekciju *loggingConfiguration*

```
<loggingConfiguration name="Logging Application Block" tracingEnabled="true"
defaultCategory="Log" logWarningsWhenNoCategoriesMatch="true">
  <listeners>
    <add name="AuditTraceListener"
         traceOutputOptions="None"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.MsmqTraceListenerD
ata, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.MsmqTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         queuePath="{$MSMQAuditQueuePath}"
         formatter="BinaryFormatter"
         useDeadLetterQueue="true" />
    <add name="LogTraceListener"
         traceOutputOptions="None"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.MsmqTraceListenerD
ata, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.MsmqTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         queuePath="{$MSMQLogQueuePath}"
         formatter="BinaryFormatter"
         useDeadLetterQueue="true" />
  </listeners>
  <categorySources>
    <add name="Audit" switchValue="All">
      <listeners>
        <add name="AuditTraceListener" />
      </listeners>
    </add>
    <add name="Log" switchValue="All">
      <listeners>
        <add name="LogTraceListener" />
      </listeners>
```

```
    </add>
  </categorySources>
  <specialSources>
    <allEvents switchValue="All" name="All Events"/>
    <notProcessed switchValue="All" name="Unprocessed Category" />
    <errors switchValue="All" name="Logging Errors &amp;amp; Warnings" />
  </specialSources>
</loggingConfiguration>
```

Pievienojiet bibliotēkas programmas palaišanas mapē:

- Microsoft.Practices.Unity.dll
- Microsoft.Practices.Unity.Interception.dll
- Microsoft.Practices.ServiceLocation.dll
- Microsoft.Practices.EnterpriseLibrary.Common.dll
- Microsoft.Practices.EnterpriseLibrary.Logging.dll
- Abc.Diagnostics.EntrLib60.dll

Kā alternatīvais variants ievietot šis bibliotēkas Global Assembly Cache, tad bibliotēkas būs pieejamas visiem lietojumiem uz mašīnas. https://msdn.microsoft.com/en-us/library/ex0ss12c(VS.80).aspx

4.5.3. Entriprise Library 6.0 MSMQDistributor to ABC diagnostics

Piemērs parāda, kā nokonfigurēt Enterprise Library 6.0 MSMQDistributer, izmantojot ABC Diagnostics.

Pievienojiet *configSections* elementu ***MsmqDistributor.exe.config*** datnē, ka pirmo elementu pēc *configuration* elementa:

```
<configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
    <section name="diagnosticConfiguration"
type="Abc.Diagnostics.Configuration.DiagnosticSettings, Abc.Diagnostics, Version=1.2.0.0"/>
</configSections>
```

Nokonfigurējiet Enterprice Library logošanas sekciju *loggingConfiguration*

```
<loggingConfiguration name="" tracingEnabled="true" defaultCategory="category0"
logWarningsWhenNoCategoriesMatch="true">
  <listeners>
    <add name="Abc.Diagnostic.EntLib"
type="Abc.Diagnostics.EntLib.AbcDiagnosticsProxyTraceListener, Abc.Diagnostics.EntLib60,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
         listenerDataType="Abc.Diagnostics.EntLib50.AbcDiagnosticsProxyTraceListenerData,
Abc.Diagnostics.EntLib50, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
  </listeners>
  <categorySources>
    <add switchValue="All" name="category0">
      <listeners>
        <add name="Abc.Diagnostic.EntLib" />
      </listeners>
    </add>
  </categorySources>
  <specialSources>
    <allEvents switchValue="All" name="All Events" />
    <notProcessed switchValue="All" name="Unprocessed Category" />
    <errors switchValue="All" name="Logging Errors &amp;amp; Warnings" />
  </specialSources>
</loggingConfiguration>
```

Pievienojiet *diagnosticConfiguration* elementu **MsmqDistributor.exe.config** datnē.

```
<diagnosticConfiguration type="Abc.Diagnostics.DefaultLogWriter, Abc.Diagnostics"/>
```

Pievienojiet *system.diagnostics* elementu **MsmqDistributor.exe.config** datnē.

```
<system.diagnostics>
  <trace autoflush="true"/>
  <sources>
    <source name="category0" switchValue="All">
      <listeners>
        <add name="Log" initializeData="log.svclog"
type="System.Diagnostics.XmlWriterTraceListener" traceOutputOptions="Timestamp"/>
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.5.4. Asinhronā logošana .NET3.5 projektiem, izmantojot Microsoft Enterprise Library 5.0

Pievienojiet *configSections* elementu lietojuma konfigurācijas datnē kā pirmo elementu pēc *configuration* elementa:

```
<configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
    <section name="diagnosticConfiguration" type="Diagnostic.Configuration.DiagnosticSettings,
Diagnostic, Version=1.0.0.0"/>
</configSections>
```

Nokonfigurējiet ABC diagnostics konfigurācijas sekciju *diagnosticConfiguration*

```
  <diagnosticConfiguration type="Abc.Diagnostics.EntrLib.EntrLibLogWriter,
Abc.Diagnostics.EntrLib50, Version=1.0.0.0" applicationIdentity="{$ApplicationIdentity}" />
```

Nokonfigurējiet Enterprise Library logošanas konfigurācijas sekciju *loggingConfiguration*

```
<loggingConfiguration name="Logging Application Block" tracingEnabled="true"
defaultCategory="Log" logWarningsWhenNoCategoriesMatch="true">
  <listeners>
    <add name="AuditTraceListener"
         traceOutputOptions="None"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.MsmqTraceListenerD
ata, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.MsmqTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         queuePath="{$MSMQAuditQueuePath}"
         formatter="BinaryFormatter"
         useDeadLetterQueue="true" />
    <add name="LogTraceListener"
         traceOutputOptions="None"

listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.MsmqTraceListenerD
ata, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.MsmqTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
         queuePath="{$MSMQLogQueuePath}"
         formatter="BinaryFormatter"
         useDeadLetterQueue="true" />
```

```
  </listeners>
  <categorySources>
    <add name="Audit" switchValue="All">
      <listeners>
        <add name="AuditTraceListener" />
      </listeners>
    </add>
    <add name="Log" switchValue="All">
      <listeners>
        <add name="LogTraceListener" />
      </listeners>
    </add>
  </categorySources>
  <specialSources>
    <allEvents switchValue="All" name="All Events"/>
    <notProcessed switchValue="All" name="Unprocessed Category" />
    <errors switchValue="All" name="Logging Errors &amp; Warnings" />
  </specialSources>
</loggingConfiguration>
```

Pievienojiet bibliotēkas programmas palaišanas mapē:

- Microsoft.Practices.Unity.dll
- Microsoft.Practices.Unity.Interception.dll
- Microsoft.Practices.ServiceLocation.dll
- Microsoft.Practices.EnterpriseLibrary.Common.dll
- Microsoft.Practices.EnterpriseLibrary.Logging.dll
- Abc.Diagnostics.EntrLib50.dll

4.5.5. Entriprise Library 5.0 MSMQDistributor to ABC diagnostics

Piemērs parāda, kā nokonfigurēt Enterprise Library 5.0 MSMQDistributer, izmantojot ABC Diagnostics.

Pievienojiet *configSections* elementu ***MsmqDistributor.exe.config*** datnē, ka pirmo elementu pēc *configuration* elementa:

```
<configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.505.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
    <section name="diagnosticConfiguration" type="Diagnostic.Configuration.DiagnosticSettings,
Diagnostic, Version=1.0.0.0"/>
</configSections>
```

Nokonfigurējiet Enterprice Library logošanas sekciju *loggingConfiguration*

```
<loggingConfiguration name="" tracingEnabled="true" defaultCategory="category0"
logWarningsWhenNoCategoriesMatch="true">
  <listeners>
    <add name="Abc.Diagnostic.EntLib"
type="Abc.Diagnostics.EntLib50.AbcDiagnosticsProxyTraceListener, Abc.Diagnostics.EntLib50,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
         listenerDataType="Abc.Diagnostics.EntLib50.AbcDiagnosticsProxyTraceListenerData,
Abc.Diagnostics.EntLib50, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
  </listeners>
  <categorySources>
    <add switchValue="All" name="category0">
      <listeners>
        <add name="Abc.Diagnostic.EntLib" />
      </listeners>
    </add>
  </categorySources>
```

```
  <specialSources>
    <allEvents switchValue="All" name="All Events" />
    <notProcessed switchValue="All" name="Unprocessed Category" />
    <errors switchValue="All" name="Logging Errors &amp;amp; Warnings" />
  </specialSources>
</loggingConfiguration>
```

Pievienojiet *diagnosticConfiguration* elementu **MsmqDistributor.exe.config** datnē.

```
<diagnosticConfiguration type="Diagnostic.DefaultLogWriter, Diagnostic"/>
```

Pievienojiet *system.diagnostics* elementu **MsmqDistributor.exe.config** datnē.

```
<system.diagnostics>
  <trace autoflush="true"/>
  <sources>
    <source name="category0" switchValue="All">
      <listeners>
        <add name="Log" initializeData="log.svclog"
type="System.Diagnostics.XmlWriterTraceListener" traceOutputOptions="Timestamp"/>
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

4.6. Lietojuma identifikācija

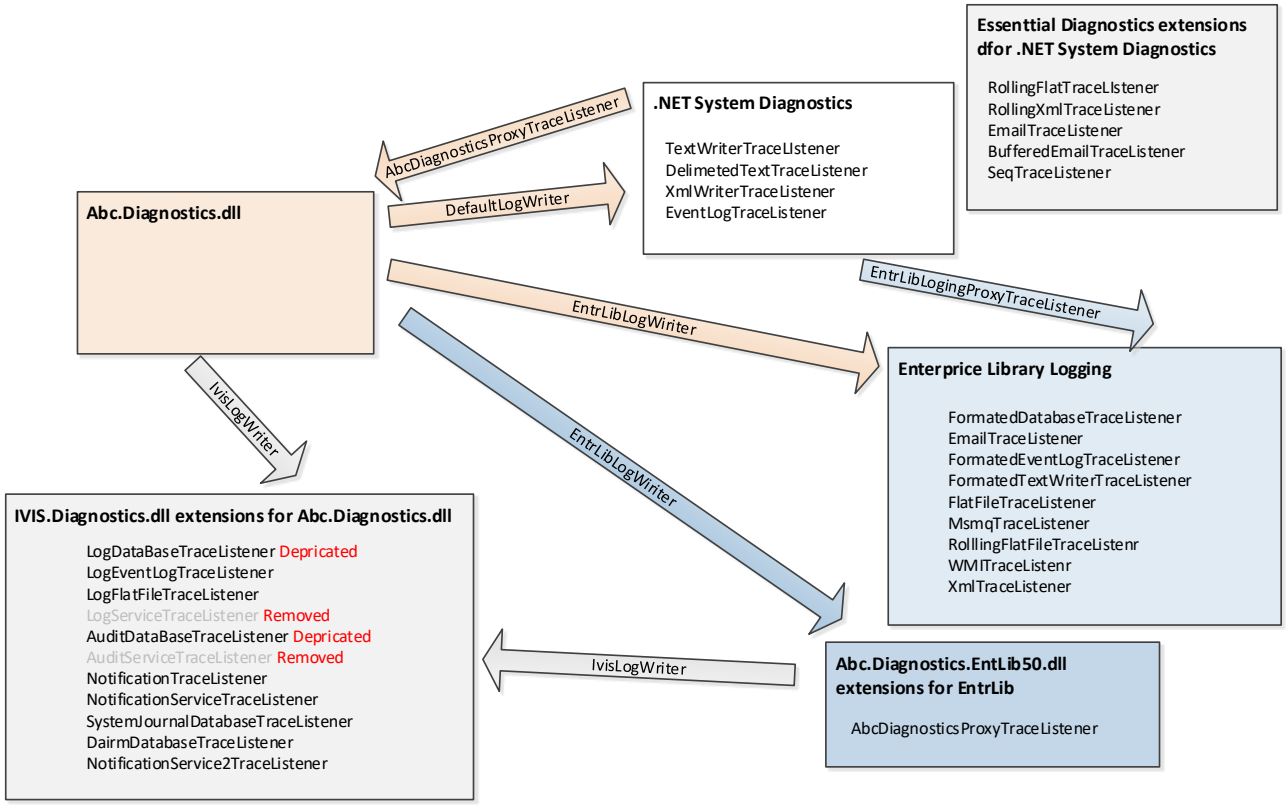
Rakstot auditu un žurnālu datubāzē, jānorāda lietojuma identifikators, lai vieglāk meklētu ierakstus.

Rakstot datubāze, lietojuma identifikators jānorāda *ApplicationIdentity* parametrā, skat. 4.3.2, 4.3.4. sadaļās.

Ja audits un žurnāls tiek rakstīts asinhroni, izmantojot Enterprise Library, tad lietojuma identifikatoru jānorāda *ApplicationIdentity* parametrā, skat. 4.5.2, 4.5.4. sadaļās.

4.7. Logošanas bibliotēku mijiedarbība

Logošanās bibliotēku mijiedarbība redzama 16.attēlā.



16.attēls. Datu plūsma starp bibliotēkām

5. Žurnalēšana no konteinerizētām komponentēm

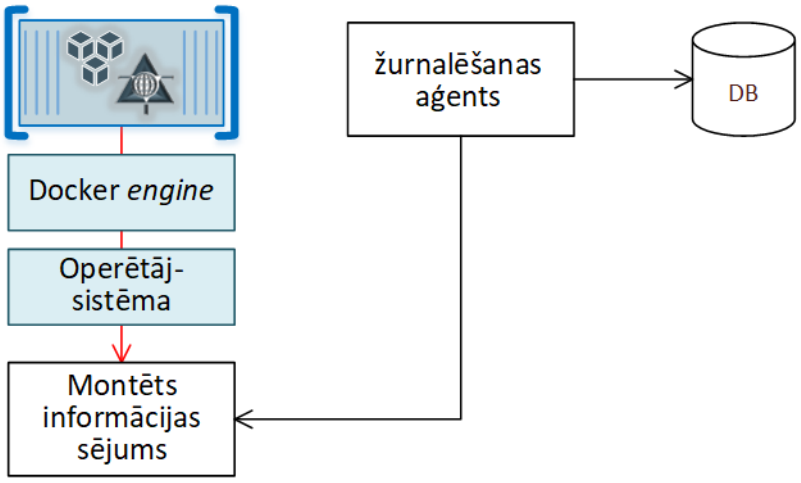
Notikumu žurnalēšanai no VRAA Kubernetes platformā izmitinātām .Net Core komponentēm ir jāizmanto Abc.Analytics.Serilog un Abc.Analytics.Serilog.AspNetCore bibliotēkas.

5.1. Notikumu žurnalēšana

Tā kā komponentes tiek izmitinātas *Docker* konteineros, lai veiksmīgi veiktu notikumu žurnalēšanu nepieciešams ņemt vērā šo konteineru darbības īpatnības. Tā kā katra *Docker* konteinera dzīves laiku pilnībā kontrolē *Docker* dzinējs (*engine*), un tas var pabeigt jebkuru konteineru potenciāli jebkurā laika momentā, konteineros darbojošos lietotņu žurnalēšanas ierakstus nedrīkst glabāt pašos konteineros — jo tie pazudīs, kad konteiners tiek pabeigts. Tāpēc izstrādātajiem jānodrošina žurnalēšana izmantojot standarta izejas plūsmas (*stdout*) tehniku, kad žurnalēšanas ierakstus raksta vienotajā plūsmā, kura, ar *Docker* dzinēja un attiecīgās operētājsistēmas palīdzību tiek novirzīta montētajā informācijas sējumā (*volume*).

> ***Docker žurnalēšanas dzinis uzskata katru rindu (rindkopu) kā atsevišķu ziņojumu — kad tiek izmantots Docker žurnalēšanas dzinis, vairākrindu ziņojumi netiek atbalstīti. Tāpēc izstrādātāju atbildībā ir nodrošināt, lai katrs atsevišķs žurnalēšanas ieraksts nesaturētu sevī rindas beigu rakstzīmi.***

No šī informācijas sējuma (tajā esošajām datnēm) žurnalēšanas aģents patstāvīgi nolasa žurnalēšanas ierakstus un saglabā tos žurnāla datubāzē (skat. 5. attēlu) *JSON* formātā (skat. 3. tabulu).



1.attēls. Žurnalēšanas ierakstu saglabāšana datubāzē

3.tabula

Žurnāla ieraksta JSON formāts

LAUKS	OBLIGĀTS	VĒRTĪBA
`Action`	Jā	"`journal`"
`Payload`	Jā	
• `Timestamp`	Jā	Notikuma reģistrēšanas datums un laiks datubāzē (tehniska informācija), kā DateTimeOffset.
• `Level`	Jā	Viena no šādām vērtībām: "`Verbose`",

LAUKS	OBLIGĀTS	VĒRTĪBA
		“Debug“, “Information“, “Warning“, “Error“, “Fatal“.
MessageTemplate	Nē	Ieraksta veidne *MessageTemplate* formātā — pēc https://messagetemplates.org/ notācijas.
Message	Jā, ja nav norādīts Exception	Vienkārša teksta (*plain text*, bez *HTML*) pilnīgi atveidots (*rendered*) ziņojums *UTF-8* kodējumā.
Exception	Jā, ja nav norādīts Message	Kļūdas detalizēts apraksts.
Properties	Nē	Visas notikuma īpašību vērtības, kas neparādās citur izvadē.

Veicot notikumu žurnalēšanu, jāievēro šādi pamatnosacījumi:

- ieraksti jāraksta lietotnes standarta izejas plūsmā (*stdout*);
- lai nodrošinātu korektu žurnālēšanas ierakstu saglabāšanu datubāzē, jānodrošina ierakstu veidošana atbilstoši vienotajam žurnalēšanas ierakstu *JSON* formātam (skat. 3. tabulu);
- lai nodrošinātu to, ka dažādās vides var tikt pielietota citādāka žurnalēšanas politika, žurnalēšanas minimālā līmeņa vērtību nepieciešams ielādēt no attiecīgā konteinera vides parametriem;
- ieraksti nesatur jaunas rindas rakstzīmi.

5.2. Žurnalēšanas klašu izmantošana

Notikumu žurnalēšana tiek nodrošināta, izmantojot pielāgotu standarta *.NET Core* žurnalēšanas abstrakcijas (`Microsoft.Extensions.Logging.ILogger<T>`) *Serilog* realizāciju. Lai pievienotu notikumu žurnalēšanu, nepieciešams:

- pievienot NUGET pakotnes
  - `Abc.Analytics.Serilog`
  - `Abc.Analytics.Serilog.AspNetCore`
- pievienot žurnalēšanas konfigurāciju, izmantojot `Microsoft.AspNetCore.Hosting.IWebHostBuilder` paplašinājuma `Abc.Analytics.Serilog.AspNetCore.UseSerilog` metodi. `JsonFormatter` klase jāņem tieši no Abc.Analytics.Serilog bibliotēkas.

Piemērs:

```
...
using Abc.Analytics.Serilog;
using Abc.Analytics.Serilog.AspNetCore;

public class Program {
...
public static IHostBuilder CreateHostBuilder( string[] args ) {
            var environmentLoggingLevelSwitch = Enum.TryParse( Environment.GetEnvironmentVariable( "Serilog__MinimumLevel" ), true, out LogEventLevel level )
                ? new LoggingLevelSwitch( level )
                : new LoggingLevelSwitch( LogEventLevel.Information );

            return Host.CreateDefaultBuilder( args )
               .ConfigureWebHostDefaults( webBuilder => {

                   webBuilder.UseStartup<Startup>();
                   webBuilder.UseSerilog(
                          configureLogger: ( provider, context, loggerConfiguration ) => {
                                loggerConfiguration
                                       .MinimumLevel.ControlledBy( environmentLoggingLevelSwitch )
                                       .MinimumLevel.Override( "Microsoft", environmentLoggingLevelSwitch.MinimumLevel )
                                       .MinimumLevel.Override( "System", environmentLoggingLevelSwitch.MinimumLevel )
                                       .Enrich.FromLogContext()
                                       .Enrich.WithHttpContext( provider )
                                       .WriteTo.Console( new JsonFormatter() );
                          } );
               } );
        }
...
}
```