jamk.fi

JKLogger

.NET ohjelmien lokitiedostojen hallinta

Joni Kukko

Harjoitustyö, Windows-ohjelmointi
Huhtikuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Sisältö

Taulukot

1 Johdanto

Harjoitustyön tarkoituksena oli luoda kirjasto, mikä tarjoaa suoraviivaisen tavan kirjata tapahtumia erilaisissa ympäristöissä erilaisiin kohteisiin. Muokattavuus ja helppokäyttöisyys olivat pääkriteerit kirjastoa suunnitellessa ja tavoitteena oli saada kirjasto sellaiseksi, että sitä pystyttäisiin käyttämään hyvin erilaisissakin projekteissa hyödyksi.

Kirjaston peruskäyttö on vaivatonta, mutta se tarjoaa kuitenkin muutamia edistyksellisiä toimintoja, joiden avulla kirjaston toimintaa voidaan muokata myös sen ulkopuolelta.

2 Asennus

Kirjaston käyttämiseksi ja asentamiseksi riittää, että lisää dll-tiedoston projektin referensseihin. Lisäämisen jälkeen kaikki kirjaston julkiset luokat ovat käytettävissä projektin eri osissa. Käyttöä voi kuitenkin helpottaa lisäämällä using ilmaisun niihin tiedostoihin, joissa kirjastoa käytetään ja osoittamalla sille kirjaston nimiavaruuden juuren.

```
using JKLog;
```

Erillisiä hakemistoja kirjasto ei tarvitse. Mikäli lokien kohteena kuitenkin on tiedostojärjestelmä, kirjasto luo käytettävän hakemistorakenteen automaattisesti joko oletusarvoin tai App.config-tiedostossa määritetyllä tavalla.

3 Kirjaston toiminnallisuudet

3.1 Toteutetut toiminnallisuudet

Kirjaston päätoiminnallisuudet löytyvät oheisesta listasta. Kirjastoa tehdessä panostettiin yksinkertaiseen rakenteeseen ja suoraviivaisuuteen.

- Tietueiden kirjoittaminen useaan kohteeseen
- Tietueiden lukeminen kohteesta
- Kohteiden konfigurointi App.config-tiedoston avulla
- Käsittelemättömien poikkeuksien nappaaminen

Näiden lisäksi kirjasto tukee myös muutamia monipuolisempia ominaisuuksia mitkä löytyvät alla olevasta listauksesta. Ominaisuudet ovat mietitty tarkkaan, että kirjastoa pystyttäisiin hyödyntämään monenlaisissa projekteissa. Nämä ominaisuudet ovat mahdollistettu täsmällisten rajapintojen avulla.

- Konfiguroitujen kohteiden ohittaminen
- Omien kohteiden lisääminen kirjaston käyttöön
- Oman tietueen käyttäminen
- Oman kirjoittimen ja lukijan käyttäminen

3.2 Toteuttamatta jääneet toiminnallisuudet

Toteuttamatta jäi yksi suunniteltu kohde ja määriteltävissä olevat raportoitavien tietueiden tasot kohde kohtaisesti App.config-tiedostossa.

Toteuttamatta jäänyt kohde oli SQLite-tietokanta. Se oli alun perinkin pienellä prioriteetilla, sillä kirjaston toiminta tulee selvästi esiin myös ilman sitä. Lisäksi sen lisääminen järjestelmään myöhemmin on hyvin yksinkertainen toimenpide.

Määriteltävissä olevat, kohde kohtaiset, tasot olisivat toimineet niin, että App.config-tiedostoon pystyttäisiin määrittämään minkä tason tietueet mikäkin kohde raportoi. Se on kyllä mahdollista toteuttaa jo nykyiselläkin kirjastolla, mutta tarkoitus oli tehdä siitä helpompaa. Sen toteuttaminen nyt vaatisi jokaisen kohteen muokkaamista mikä ei ensinnäkään ole mielekästä ja toisekseen alun perin suunniteltu tapa, jossa järjestelmä hoitaisi sen taustalla automaattisesti, olisi huomattavasti järkevämpi kokonaisuus.

4 Käyttö

Kirjaston käyttö on pyritty saamaan mahdollisimman yksinkertaiseksi ja tässä kappaleessa on tarkoitus käydä pääkäyttötapaukset läpi.

4.1 Kirjoittaminen oletuskohteisiin

Oletuskohteet ovat ne, mitkä ovat määriteltynä App.config-tiedostoon. Mikäli tiedostoon ei ole määritelty mitään, käyttää kirjasto oletuksenaan JKConsole-kohdetta. Staattinen JKLogger käyttää aina oletuskohteita ja se sisältää useita käyttöä helpottavia metodeja. Ohjelman on hyvä kutsua JKLogger-luokan Dispose-metodia ennen ohjelmasta poistumista, vaikka käytössä ei sillä hetkellä olisikaan yhtään hävittämistä tarvitsevaa kohdetta.

```
static void Main(string[] args)
{
    JKLogger.Information("Hello world!");
    JKLogger.Dispose();
}
```

4.2 Oletuskohteiden määrittäminen

Oletuskohteet määritellään App.config-tiedostossa. Kohteet määritellään Mapper elementein laittamalla ne JKLog-elementin sisään. Kohteelle annetaan attribuutiksi type, minkä arvona tulee olla halutun kohteen nimi. Kohteelle voidaan antaa lisämääreitä add-elementillä, mikä toimii yksinkertaisesti key-value parina. Käytettävissä olevat lisämääreet vaihtelevat kohteen mukaan. Alla näkyy kaikki käytettävissä olevat kohteet ja osa mahdollisista lisämääreistä.

```
<JKLog>
  <mapper type="Managed" />
  <mapper type="JKConsole">
    <add key="verbose" value="false" />
  </mapper>
  <mapper type="LogFile">
    <add key="path" value="kansio" />
    <add key="filename" value="tiedosto" />
    <add key="extension" value="txt" />
  </mapper>
  <mapper type="WindowsEvent">
    <add key="source" value="JKLogTest" />
  </mapper>
</JKLog>
```

Näiden lisäksi LogFile-kohde sisältää lisämääreitä, mitkä mahdollistavat käytettävän polun ja tiedostonimen linkittämistä sen hetkiseen aikaan, antamalla arvoksi DateTime:n hyväksymä formaatti. LogFile käyttää ensisijaisesti yksinkertaista arvoa, mikäli App.config-tiedostoon on määritelty sekä yksinkertainen, että formaatissa oleva lisämääre. Kaikki käytössä olevat lisämääritteet ja niiden oletusarvot löytyvät taulukosta.

Kohde	Lisämääre	Oletus
JKConsole	verbose	true
LogFile	path formatPath filename formatFilename extension	JKLog g JKLog g log
Managed		
WindowsEvent	source	JKLog

Taulukko 1 Kohdeiden lisämääreet

4.3 Tietueiden lukeminen oletuskohteesta

Kirjastolla pystytään lukemaan mitä tahansa kohdetta, mikä implementoi IReadable-rajapinnan. Toistaiseksi ainoastaan Managed-kohde implementoi kyseisen rajapinnan. Oletuskohteesta lukeminen vaatii kohteen instanssin pyytämistä MapperManager-luokalta. Se tapahtuu antamalla kyseiselle luokalle kohteen tyyppi mitä halutaan lukea. Mikäli saman tyyppisiä kohteita on määritelty useita, käytetään ensimmäiseksi vastaan tulevaa tyypin mukaista kohdetta. Huomaa, että Managed-kohde tulee olla määriteltynä App.config-tiedostossa.

```
static void Main(string[] args) {
   JKLogger.Information("Hello world!");
   var readable = MapperManager.GetDefaultMapper(typeof(Managed)) as IReadable;
   if (readable != null) {
       using (var reader = new JKReader(readable))
       {
           foreach (IEntry entry in reader)
           {
               Console.WriteLine(entry.Message);
           }
       }
   }
    JKLogger.Dispose();
}
```

4.4 Käsittelemättömien poikkeuksien nappaaminen

Kirjasto sisältää yksinkertaisen tavan kirjata lokijärjestelmään käsittelemättömät poikkeukset. Tämä ei estä ohjelman suorittamisen loppumista, mutta tarjoaa mahdollisuuden kirjata tulleet poikkeukset kohteisiin. Käsittelijä käyttää JKLogger-luokkaa poikkeuksien kirjoittamiseksi kohteisiin, joten käytössä on aina oletuskohteet. Käsittelijä rekisteröidään ohjelmaan seuraavalla tavalla.

```
static void Main(string[] args)
{
    AppDomain.CurrentDomain.UnhandledException += JKExceptionHandler.Unhandle-
dException;

    // aiheutetaan unhandled exception
    string s = null;
    s.Trim();
}
```

4.5 WindowsEventin rekisteröiminen

Mikäli ohjelmassa on tarkoitus käyttää WindowsEvent-kohdetta, pitää pitää huolta siitä, että sen tarvitsema source tulee rekisteröityä oikein. Tämä tarvitsee järjestelmänvalvojan oikeudet ja usein tarvittava source rekisteröidäänkin ohjelman asennuksen aikana. Kirjasto sisältää kuitenkin tavan tehdä rekisteröinnin myös ohjelman sisäisesti. Tämäkin tapa kuitenkin vaatii, että ohjelmaa ajetaan järjestelmänvalvojan oikeuksin. Tämän lisäksi source tulee käyttöön vasta seuraavalla ohjelman käynnistyskerralla. Rekisteröinti on huolehdittu niin, että sitä voidaan kutsua jokaisella käynnistyskerralla eikä se aiheuta ongelmia ohjelman toimimisessa. Tämä rekisteröinti käyttää aina App.config-tiedostossa määritettyjä kohteen lisämääreitä.

```
static void Main(string[] args)
{
    var events = MapperManager.GetDefaultMapper(typeof(WindowsEvent)) as Win-
dowsEvent;
    if (events != null)
        events.RegisterSource();
}
```

5 Ongelmat ja kehitysideat

Varsinaisia ongelmia kirjastossa ei ole. Kaikki ominaisuudet toimivat kuten kuuluu, eikä virhetilanteetkaan aiheuta ohjelmaa kaatavia ongelmia. Ainoastaan WindowsEventin hankalahko rekisteröiminen jäi vaivaamaan, mutta kehittelemäni ratkaisu hoitaa asian kuitenkin suhteellisen siististi.

Jatkoa ajatellen, tulisi ensitöiksi tehdä toteuttamatta jääneet ominaisuudet valmiiksi. Näiden lisäksi myös IReadable-rajapinnan implementointi kohteisiin olisi järkevää. IDisposable-rajapintojen implementoinnit tulisi tarkistaa ja tarvittaessa suunnitella uudestaan. Tällä hetkellä MapperManager-luokka hoitaa varsinaisen Dispose-metodin kutsumisen ja sen logiikka ei ole niin selkeä kuin haluaisin.

Tarpeelliset jatkokehityskohteet tulevat tietenkin vasta esiin, kun kirjastoa käyttää oikean projektin kanssa. Sillä se saattaa paljastaa kirjaston ennalta arvaamattomia heikkouksia.

6 Analyysi

Harjoitustyössä tuli jälleen opittua enemmän arkkitehtuurisesta suunnittelusta ja varsinkin rajapintojen järkevästä käytöstä. Rajapintojen pilkkominen pienempiin osiin mahdollisti erilaisten kohteiden luomisen ilman turhia metodeja, jolloin luokat pysyivät erittäin siisteinä.

Suunnittelun lisäksi tuli opittua paljon WindowsEvent-lokitietueiden käytöstä, käsittelemättömien poikkeuksien hallinnasta ja CompilerServicen tarjoamista mahdollisuuksista. Nämä kyseiset asiat olivat avainasemassa onnistuneen kirjaston rakentamisessa.

Haasteiksi listaan ainoastaan arkkitehtuurin ja kirjaston suunnittelun. Oli erityisen mielekästä rakentaa kirjasto muiden tekemiä ohjelmia ja projekteja varten. Opiskelin muun muassa jo olemassa olevia loggereita kartoittaen kaikkia mahdollisia käyttökohteita mihin muut loggerit ovat osanneet varautua. Olen erityisen tyytyväinen siihen, miten App.config-tiedostoa luetaan ja miten sieltä kohteet parsitaan esiin, sillä siellä on mahdollista rekisteröidä myös kirjaston ulkopuolisia luokkia kohteiksi käytettäväksi kirjaston sisällä.

7 Loppusanat

Loppusanoiksi esitän perustelut harjoitustyöstä annettaviin pisteisiin.

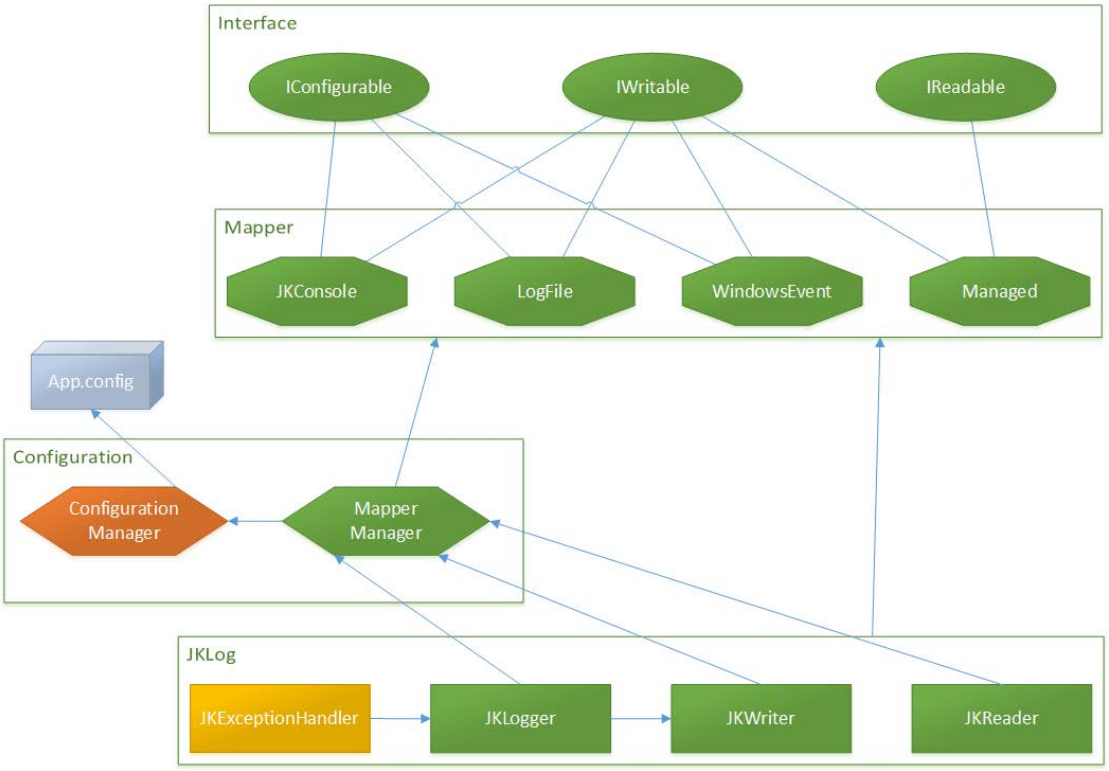
Harjoitustyö oli rivimäärältään suhteellisen pieni, mutta mielestäni sain toteutettua kirjaston varsin järkevästi. Pidin onnistuneesti mielessäni kirjaston varsinaisen päämäärän, enkä lähtenyt toteuttamaan turhia ominaisuuksia vaan pidin lopputuloksen siistinä.

Sain suunniteltua kirjaston ominaisuuksiltaan monipuoliseksi, mutta kuitenkin helppokäyttöiseksi. Sen käyttö on suoraviivaista ja selkeää, sekä sen pystyy ottamaan mukaan projektiin käyttöön ilman minkäänlaista konfigurointia. Konfiguroinnit voidaan tehdä vasta jälkikäteen, mikä osaltaan nopeuttaa sen käyttöönottoa kiireisissä projekteissa.

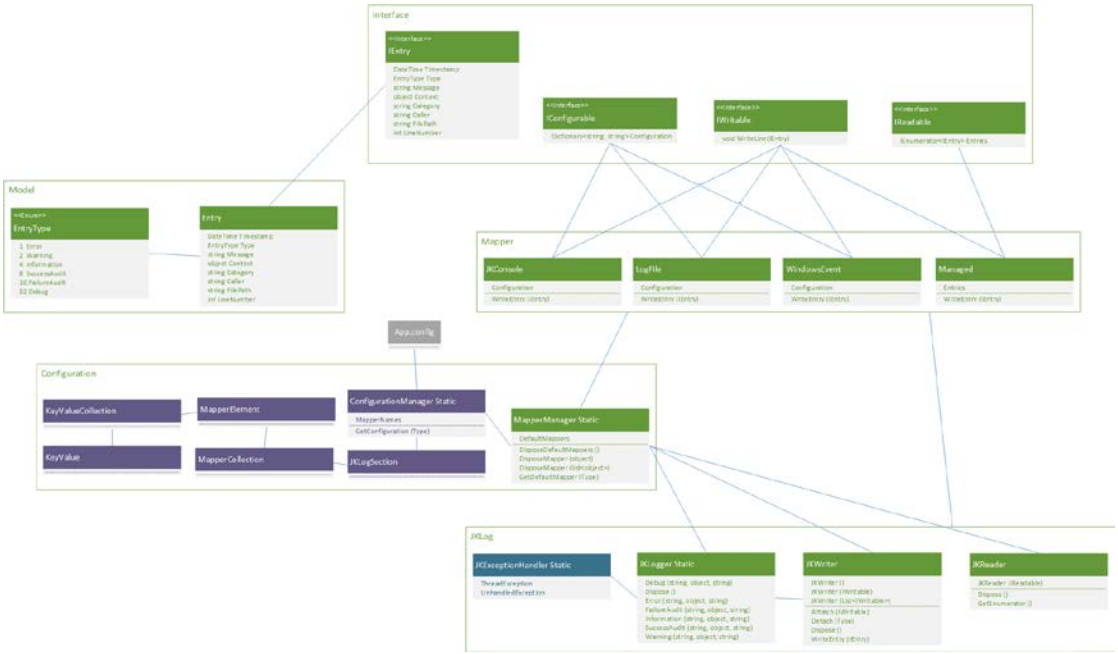
Varsinainen koodi on luettavaa, selkeää ja hyvin kommentoitua käyttäen Visual Studion merkintätapaa niin, että IntelliSense osaa näyttää vihjeet koodatessa oikein.

Mielestäni onnistuneesta toteutuksesta huolimatta työ oli pieni ja muutama suunniteltu toiminnallisuus jäi toteuttamatta. Näistä syistä ehdotan harjoitustyön pisteiksi 25/30.

Liite 1. Yksinkertainen arkkitehtuuri



Liite 2. Tarkempi arkkitehtuuri



Liite 3. Windows event kuvakaappaus