KEMI-TORNION AMMATTIKORKEAKOULU
TEKNIIKAN KOULUTUSYKSIKKÖ

Kilponen Arto

Tiedoston jäsentäjän toteutus VBA:lla

Tietotekniikan koulutusohjelman opinnäytetyö
Ohjelmistotekniikka
Kemi 2009

TIIVISTELMÄ

Kemi-Tornion ammattikorkeakoulu, Tekniikan yksikkö

Koulutusohjelma	Tietotekniikka
Opinnäytetyön tekijä	Arto Kilponen
Opinnäytetyön nimi	Tiedoston jäsentäjän toteutus VBA:lla
Työn laji	Opinnäytetyö
Päiväys	9.5.2009
Sivumäärä	34
Opinnäytetyön ohjaaja	FM Teppo Aalto

Opinnäytetyössä on toteutettu ohjelma, joka analysoi annettua lähdetiedostoa, paikantaa halutut tiedot ja raportoi löydetyt tulokset haluttuun tiedostoon. Ohjelma on toteutettu Visual Basic for Applications -kielellä (VBA) Excel 2007:ssä ja ohjelma on suunniteltu mahdollisimman helposti siirrettäväksi muihin VBA:ta hyödyntäviin ohjelmiin.

Ohjelma tukee Hypertext Markup Language (HTML) –syötteitä, esimerkiksi tiedostopäätteet .htm ja .html, sekä normaalia tekstitietoa, kuten tiedostopääte .txt. Nämä ovat yleisimmät lähteet tekstimuotoiselle tiedolle. Toteutettu ohjelma hyödyntää INI-tiedostoja, joissa on etukäteen määritetty lähteestä etsittävän tiedon yksityiskohdat. INI-tiedostot voivat olla joko normaaleja tekstitiedostoja tai .INI-päätteisiä tiedostoja, joissa on tarkasti määritelty syntaksi. Ohjelma voi esittää tulokset suoraan Excelin taulukoissa tallennettuna Excelin omalle tiedostomuodolleen tai vaihtoehtoisesti HTML-muodossa. Tällöin tulokset esitetään HTML-tauluissa ja tulostiedostoa voi lukea Internet-selaimella.

Opinnäytetyön tavoitteena oli myös edistää VBA-ohjelmoinnin taitoja sekä oppia työskentelemään HTML-tiedostojen parissa. Tavoitteena oli myös tutustua Visual Studio Tools for Officen (VSTO) ominaisuuksiin.

Asiasanat: Excel, Visual Basic for Applications, ohjelmointiympäristö.

ABSTRACT

Kemi-Tornio University of Applied Sciences, Technology

Degree Programme	Information Technology
Name	Arto Kilponen
Title	Implementation of a File Parser by Using VBA
Type of Study	Bachelor's Thesis
Date	9 May 2009
Pages	34
Instructor	Teppo Aalto, MSc

This study discusses the implementation of an application that navigates through an input file, locating specified information and writing it to a result file specified by the user. The application was implemented in Visual Basic for Applications (VBA), running under Microsoft Office Excel 2007, and was designed for optimum portability to other applications supporting VBA.

The application supports Hypertext Markup Language (HTML) input (file extensions .htm and .html) as well as plain text input (file extension .txt), these being the most common file formats for textual information.

The implemented application makes use of initialization files, written prior to application execution, which detail the types of information to be extracted from the input. The initialization files may be in plain text or .INI file format, with a strictly defined syntax.

The application can output its results directly to Excel worksheets, saved in their own output file in Microsoft Excel's file format. It can also output directly to HTML, formatting its output as tables in an HTML page which may then be displayed by any Web browser.

From a development process standpoint, the improvement of VBA programming skills was among the goals of this study. Learning to use Visual Studio Tools for Office (VSTO) and HTML data format were also major considerations.

Keywords: Excel, Visual Basic for Applications, Integrated development environment.

SISÄLLYSLUETTELO

TIIVISTELMÄ I
ABSTRACT II
SISÄLLYSLUETTELO III
KÄYTETYT MERKIT JA LYHENTEET IV
1. JOHDANTO 1
2. VBA-OHJELMOINTI 2
2.1. Visual Basic Editor 2
2.1.1. VBE:n sisältö lyhyesti 3
2.1.2. VBE:n käyttötarkoitus 5
2.1.3. VBE:n tulevaisuus 6
2.1.4. VBE:n puutteita 7
2.1.5. VBE:n ohjelmoiminen 7
2.1.6. Tietolähteitä 7
2.2. Visual Basic for Applications 8
2.2.1. Syntaksi 8
2.2.2. Muuttujat 9
2.2.3. Taulukkomuuttujat 10
2.2.4. VBA:n ohjausrakenteet 11
3. OHJELMAN SUUNNITTELU JA TOTEUTUS 14
3.1. Vaatimusmäärittely 14
3.2. Ohjelman suunnittelu 14
3.3. Tagit 16
3.4. Ohjelman kehitysvaiheet 16
4. OHJELMAN OSIOT 20
4.1. INI-tiedoston käsittely 20
4.2. Tiedoston jäsentäjän toiminta 23
4.3. Dynaaminen tagilaturi 25
4.4. Tulosten esittäminen 26
4.5. Unicode-muuntajat 27
4.6. Lähdetiedoston lukeminen 28
4.7. Tietojen noutaminen Internetistä 29
4.8. Testaaminen 29
4.9. Ohjelman käyttöohje 30
4.9.1. INI-tiedoston kirjoittaminen 30
5. YHTEENVETO 32
6. LÄHDELUETTELO 33

KÄYTETYT MERKIT JA LYHENTEET

C	Standardisoitu ohjelmointikieli.
C++	Standardisoitu ohjelmointikieli.
C#	Standardisoitu ohjelmointikieli. Lausutaan C-sharp.
DAO API	Data Access Object Application Programming Interface. Yleinen ohjelmointirajapinta Microsoft Windowsille.
HTML	HyperText Markup Language. Avoimesti standardoitu kuvauskieli, jonka avulla voidaan kuvata hyperlinkkejä sisältävää tekstiä.
IntelliSense	Microsoftin toteutus automaattiselle kirjoittamiselle. Käytössä lähinnä Visual Studiossa ja Visual Basic Editorissa.
ODBC	Open Database Connectivity. Microsoftin määrittelemä rajapinta (API) tietokannoille.
RibbonX	Microsoft Office Systemin valikkojärjestelmä versiosta 2007 eteenpäin.
VBA	Visual Basic for Applications. Microsoft Office Systemin automatisointiin käytetty kieli.
VBE	Visual Basic Editor. Visual Basic for Applicationsin muokkausohjelma. Tunnetaan myös nimillä VBIDE ja VBA Editor.
VSTO	Visual Studio Tools for Office. Visual Studion osa, jonka avulla voidaan automatisoida Microsoft Office –systemiä.
WinAPI	Windows Application Programming Interface. Ryhmä rajapintoja Microsoft Windowsin ohjelmointiin.
XML	Extensible Markup Language. Merkintäkieli. Rakenteellinen kuvauskieli, joka auttaa jäsentämään suuria tietomääriä.

1. JOHDANTO

Tämä opinnäytetyö on tehty kevään 2009 aikana tekijän omaan tarpeeseen. Opinnäytetyössä on toteutettu ohjelma, jonka avulla voidaan analysoida tiedoston sisältö.

Yksi työn tavoitteista oli kirjoittaa lyhyt johdanto Visual Basic for Applicationsiin, minkä avulla opinnäytetyön ohjelman voisi ymmärtää suhteellisen helposti. Lisäksi Visual Basic Editorista esitellään oleellisimmat osat ja VBA:ta verrataan Visual Studio Tools for Officeen.

Työ aloitettiin syksyllä 2008 kirjoittamalla opas Visual Basic Editorista kirjallisen viestinnän kurssilla. Opas kirjoitettiin tätä opinnäytetyötä varten.

Työn vaativin osuus oli rakentaa ohjelma, joka tulkitsee tiedoston sisällön. Lisäksi VBE:stä kirjoittaminen tarjosi suuren haasteen, sillä tietoa löytyi pääasiassa vain VBA:ta käsittelevistä kirjoista sekä asiantuntevilta Internetin keskustelupalstoilta. Opinnäytetyön kirjoittaminen vaati paljon tutkimista ja asiantuntijoilta kysymistä, sillä VBE:stä ei koottua tietoa ollut saatavilla käytännössä ollenkaan.

Opinnäytetyö käsittelee vain edellä mainittuja asioita. Opinnäytetyöstä on rajattu pois ohjelman ylläpito ja jatkokehittely.

2. VBA-OHJELMOINTI

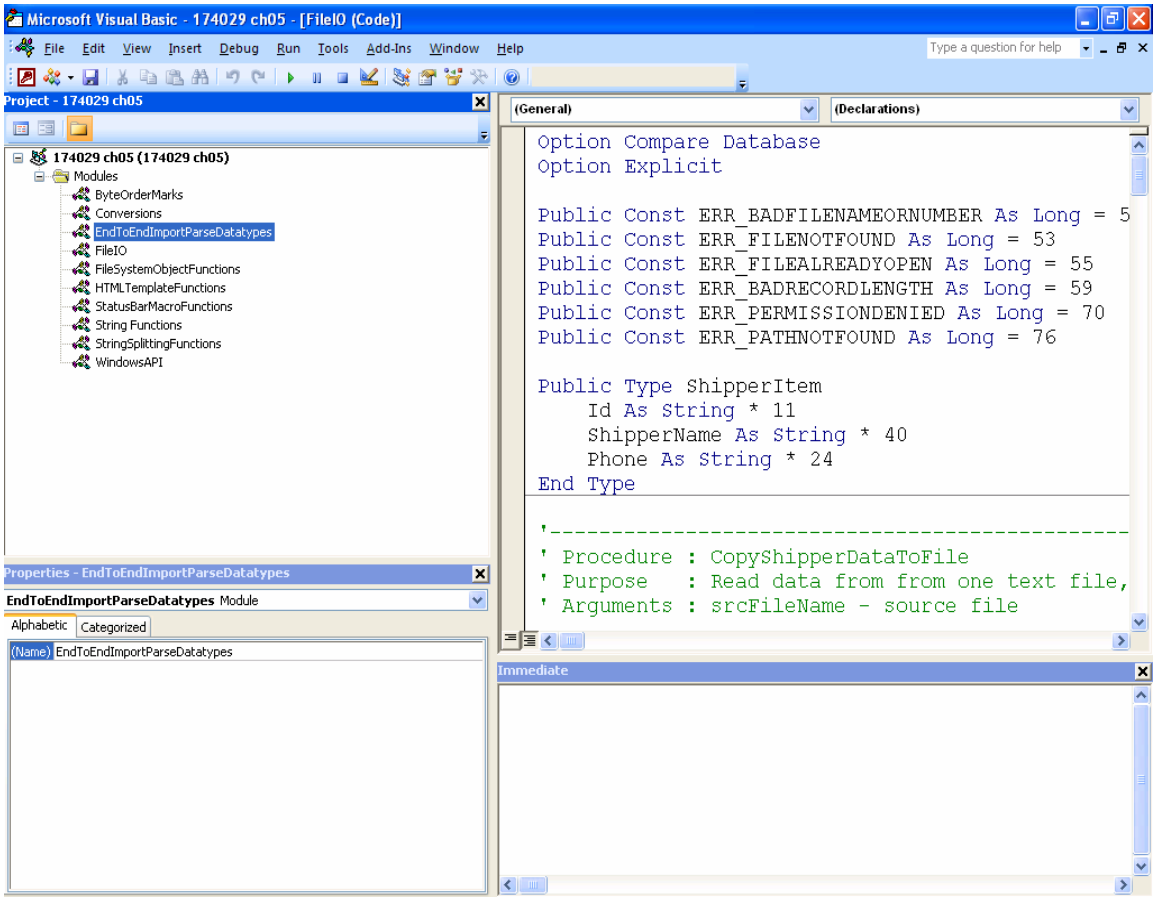
Osa Microsoft Office Systemin (myöhemmin Office) ohjelmista, kuten Microsoft Office Word (myöhemmin Word) ja Microsoft Office PowerPoint (myöhemmin PowerPoint), toimivat varsin hyvin, vaikka Visual Basic for Applicationsia (myöhemmin VBA) ei käytettäisi lainkaan. Sen sijaan Microsoft Office Accessin (myöhemmin Access) käyttöön VBA-ohjelmointi on käytännössä vaatimus, jotta pystyttäisiin tekemään mitään ammattimaisempaa. /4, s. 345–346/

Microsoft Office Exceliä (myöhemmin Excel) voidaan käyttää ilman VBA:n hyödyntämistä, sillä Excelissä työkirjafunktioiden avulla suoriudutaan varsin monimutkaisistakin laskuista. VBA:n suurimmat hyödyt työkirjafunktioihin nähtynä tulevat VBA:n monenlaisesta vapaudesta ja mahdollisuudesta käyttää sekä muuttujia että erilaisia ohjausrakenteita.
/12, s. 623/

Esiselvityksissä tutustutaan ohjelmointiympäristönä käytettyyn VBE:hen sekä VBA-ohjelmointikieleen. VBE:stä tarkastellaan ohjelman roolia VBA-ohjelmoinnissa ja VBA:sta käydään läpi opinnäytetyön kannalta oleellisimmat perusteet.

2.1. Visual Basic Editor

Visual Basic Editorilla, joka on osa Officea, kirjoitetaan VBA-ohjelmia. Kuten kuvasta 1 nähdään, VBE sisältää tekstieditorin. VBE on käytännössä identtinen kaikissa Officen osissa, joten kuvasta 1 on hankala nopeasti päätellä, mistä Officen osasta kuvassa olevaa VBE:tä käytetään. Tässä tapauksessa oikea vastaus on Microsoft Access, sillä vasemmassa yläkulmassa on Accessin punainen avainlogo. Lisäksi ensimmäinen ohjelmakoodirivi, Option Compare Database, on nimenomaan Accessin komento.



Kuva 1. Visual Basic Editor

2.1.1. VBE:n sisältö lyhyesti

VBE:hen pääsee helpoiten pikanäppäinyhdistelmällä Alt+F11, joka toimii niin Office 2007:ssä kuin myös vanhemmissa Officen versioissa. VBE:hen pääsee käsiksi valikoiden kautta. Office 2007:ään uutena ominaisuutena tulleen RibbonX-valikkojärjestelmän myötä navigointi VBE:hen on kuitenkin täysin erilainen kuin Office 2003:ssa. VBE:ssä ei ole mitään uutta Office 2003:een nähtynä, joten VBE:ssä ei ole RibbonX-valikkojärjestelmää. RibbonX-valikkojärjestelmä ei ole täydellisesti käytössä koko Office 2007:ssa. Esimerkkinä tästä mainittakoon Microsoft Office Outlook (myöhemmin Outlook). /9, 13, s. 30/

RibbonX-valikkojärjestelmää ei ole tehty VBA:lla, vaan Extensible Markup Languagella (myöhemmin XML). Siten valikoiden ohjelmointi ei onnistu kovin vaivattomasti VBE:stä käsin. VBE:n avulla voidaan kirjoittaa tekstitiedostoja, jotka voidaan tallentaa mille tahansa normaalille tekstitiedostomuodolle. VBE:stä ei siis ole täysin mahdotonta ohjelmoida RibbonX:llä. Samalla tekniikalla voidaan kirjoittaa myös HTML-tiedostoja. /8, s. 56/

VBE:hen sisältyy myös paljon muita ominaisuuksia, kuten Immediate-ikkuna. Immediate-ikkunan avulla voidaan suorittaa ohjelmakomentoja suoraan VBE:stä ohjelmaa sen enem-

pää kirjoittamatta. Yleensä tätä tehdään kokeilumielessä, kun halutaan nähdä, miten tietty komento toimii. Immediate-ikkunaa voidaan käyttää kokeiluluontoiseen tulostamiseen Debug.Print -komennon avulla. Immediate-ikkuna näkyy kuvassa 1 oikealla alhaalla. VBE:stä käytetään myös nimityksiä VBA Editor ja VBIDE, eli Visual Basic Integrated Development Environment. Kaikki kolme tarkoittavat samaa asiaa.
/13, s. 141/, /3, s. 571/

VBE:ssä on myös yleensä auki projektien hallintaikkuna, kuten kuvassa 1 vasemmalla keskellä. Projektien hallintaikkunan avulla voidaan käsitellä montaa Office-tiedostoa samanaikaisesti. Käytännössä VBE on avattava jokaiselle Office-ohjelmalle erikseen, eli haluttaessa ohjelmoida esimerkiksi Wordia ja PowerPointia samanaikaisesti myös VBE:n pitää olla kahteen kertaan auki. VBE:ssä ei myöskään näy muiden Officen osien projektit. Mikäli Excelissä on projekti auki, se ei kuitenkaan näy Outlookin VBE:ssä. Toisten Officen osien projekteja voidaan ohjata ulkoisten viittausten avulla. Officen lisäksi käsiksi päästään myös kaikkiin muihin VBA:ta tukeviin ohjelmiin, kuten esimerkiksi Internet Exploreriin ja
CorelDRAW X4:ään. /5/

Kuvassa 1 vasemmalla alhaalla on Properties-ikkuna, jossa näkyy projektitiedoston tai moduulin ominaisuuksia, kuten nimi. Mikäli VBA on asennettu tietokoneelle, jokaisella Office-tiedostolla on oma projektinsa. Projektit ovat kuitenkin käyttäjän kannalta käytännössä tyhjiä, kunnes tiedostoa käsitellään VBA:n avulla. Tällaista käsittelyä on esimerkiksi makron nauhoitus. Taustalla projekti on kuitenkin olemassa alusta asti.

Ohjelmakoodi sijaitsee proseduureissa, jotka puolestaan sijaitsevat moduuleissa. Moduuleita on kahdenlaisia: luokkamoduuleita, joissa toteutetaan luokkia sekä normaaleita moduuleja, jotka sisältävät aliproseduureja ja funktioita. Aliproseduureilla eli subproseduureilla pystyy tekemään melkein mitä vain, kuten muuttamaan muotoiluja. Funktiot puolestaan palauttavat vain arvon. Esimerkiksi Excelissä tämä tarkoittaa sitä, että solua ei voi värjätä funktiossa. Vastaavasti kun halutaan kirjoittaa kaava, siihen on helpompi käyttää funktiota. Funktiota voi sitten käyttää solussa esimerkiksi kirjoittamalla *=yhteen(1,2,3)*. Tässä *yhteen* on VBA:lla toteutettu funktio. /3, s. 21/

Käytännössä jokaisessa Office-tiedostossa on myös valmiina moduuleita tapahtumien käyttöön. Tapahtumia ovat esimerkiksi tiedoston tallennus ja avaus. Tapahtumia voidaan myös käyttää herätyskellon tapaan. Näin VBA-ohjelmaa voidaan esimerkiksi käskeä tallentamaan Office-tiedosto tunnin välein. Tapahtumasarjat ovat toisinaan hyvin monimutkaisia. Tällöin on tärkeää tietää tapahtumien suoritusjärjestys. Esimerkiksi ohjelmaa suljettaessa kysytään ensin tiedoston tallennuksesta ja sitten vasta ohjelma suljetaan, eikä toisinpäin. Monesti tapahtumien järjestyksestä pääsee selville loogisella päättelyllä. Luokkamoduulien avulla voi toteuttaa myös omia tapahtumia. /3, s. 199/

Visual Basic Editorilla voidaan luoda myös käyttöliittymiä, kuten raportteja ja lomakkeita. Esimerkiksi Excelissä käyttöliittymiä voidaan suunnitella käyttäen ActiveX-kontrolleja ja lomakkeita. Kun ohjelmisto toteutetaan tarpeeksi pitkälle, käyttäjä ei välttämättä edes tiedä käyttävänsä Office-ohjelmaa.

Office 2007:ssa automatisoinnin tason näkee hyvin pitkälti tiedoston tallennusmuodosta. Vanhemmissa Office-tiedostoissa tallennusmuodosta ei voinut päätellä mitään automatisoinnin tasosta. Taulukossa 1 on listattuna Microsoft Wordin yleisimmät tiedostomuodot.

Taulukko 1. Wordin tiedostomuodot VBA-koodin kannalta /11, s. 771/

Ohjelma	Versioissa	Tiedoston pääte	Tiedostotyypin kuvaus	VBA-koodi käytössä?
Word	2007–	.docx	Normaali Wordin dokumentti.	Ei
Word	2007–	.docm	Normaali Wordin dokumentti, jossa makrot ovat sallittuja.	Luultavasti
Word	2007–	.dotx	Wordin mallipohja dokumentti.	Ei
Word	2007–	.dotm	Wordin mallipohja dokumentti, jossa makrot ovat sallittuja.	Luultavasti
Word	97–2003	.doc	Word dokumentti versioista 97–2003	Ei voi päätellä
Word	97–2003	.dot	Word mallipohja dokumentti versioista 97–2003	Ei voi päätellä

2.1.2. VBE:n käyttötarkoitus

VBE:tä käytetään VBA:lla ohjelmoimiseen, mutta käyttämällä WinAPI:a voidaan ohjelmoida myös Windowsia, kuten Vistan dialogeja. WinAPI:n dokumentaatio on yleisesti ottaen tarkoitettu C/C++ -kielille. VBA:han dokumentaatiota joudutaan soveltamaan muun muassa muuttujien tietotyyppien osalta. WinAPI:lla ohjelmoiminen on jokseenkin vaarallista, sillä sen avulla voi helposti saada käyttöjärjestelmän vähintäänkin kaatumaan. Myös DAO API:n avulla voidaan laajentaa VBA-ohjelmaa Officen ulkopuolelle, muun muassa ODBC-tietokantojen käyttöön. /7/

Yksi suurimmista uudistuksista VBA:n viimeisimmissä versioissa on mahdollisuus luoda viittauksia ulkoisiin tyyppikirjastoihin myöhäisen sitomisen (late binding) avulla. Sitomisen kautta objekteja voidaan käyttää aivan kuten Office-ohjelmien omia objekteja. Viittausten avulla voidaan myös liittää Office-ohjelmia yhteen, jolloin esimerkiksi Access-tietokannan osoitetiedoista voidaan tehdä postituslista Wordilla ja lähettää se sähköpostilla Outlookin kautta. Tätä kutsutaan myös automaatioksi. /3, s. 411–412/**,** /1, s. 387/

Viittauksia voidaan tehdä kahdella tavalla, aikaisella ja myöhäisellä sitomisella. Kumpikin tapa tehdään esittelemällä objektimuuttuja, joka viittaa haluttuun ohjelmaan. Tämä objektimuuttuja voi sitten käyttää ominaisuuksia ja metodeja halutusta ohjelmasta. Erona aikaisella ja myöhäisellä sitomisella on se, että myöhäisessä sitomisessa objektimuuttuja tehdään ennen linkin tekemistä tyyppikirjastoon. Aikaisessa sitomisessa järjestys on päinvastainen. Kun ohjelma suunnitellaan julkaistavaksi, on käyttäjän kannalta helpointa päätyä myöhäiseen sitomiseen, sillä aikaista sitomista käytettäessä käyttäjä joutuu itse tekemään viittaukset VBE:stä. Ohjelmaa kirjoitettaessa on kuitenkin helpompaa käyttää aikaista sitomista, sillä silloin se mahdollistaa IntelliSensen käytön. /1, s. 584/

Uutta VBA-ohjelmaa tehdessä voidaan periaatteessa käyttää kahta tapaa. Ensimmäinen tapa on nauhoittaa makro, joka käytännössä tapahtuu käskemällä Office-ohjelmaa kirjoittamaan ylös suoritetut toiminnot. Kun haluttu toimintosarja on tehty, nauhoitus lopetetaan ja VBE:stä voi lukea valmiin VBA-koodin. Tämä ei kuitenkaan ole kovin tehokasta, sillä makroa nauhoitettaessa VBA-koodiin tallentuu suuri määrä ylimääräistä ohjelmakoodia. Esimerkiksi kun Excelillä nauhoitetaan makro, jossa soluun A1 tallennetaan teksti ”testi”, niin sen sijaan, että VBA-koodissa muutettaisiin yksinkertaisesti solun arvoa, tallentuu myös mahdollinen solun valitseminen ja aktivointi. Makrojen nauhoituksesta onkin lähinnä hyötyä käytettyjen objektien selvittämisessä. Käytännössä nauhoitettua makroa tulisi aina muokata nauhoituksen jälkeen. Toinen tapa on ohjelmoida perinteisellä tavalla, eli kirjoittaa ohjelma riippumatta siitä, tehdäänkö se Officea varten vai ei.

On syytä huomata, että nimitys makro on monesti harhaanjohtava. Makroilla tarkoitetaan käytännössä VBA-moduulissa sijaitsevaa ohjelmakoodia. Sen sijaan Accessissa makro ei ole VBA-koodia, vaan pieni erillinen ohjelma. /4, s. 1133–1134/

2.1.3. VBE:n tulevaisuus

VBA:ta järeämpään käyttöön Microsoft on julkaissut muutamia vuosia sitten Visual Studio Tools for Officen (myöhemmin VSTO), joka on nykyisin integroitu osa Visual Studiota. Koska VSTO:n käyttöön tarvitaan siis toistaiseksi Officen ulkopuolinen ohjelma, VBA säilynee käytössä vielä jonkin aikaa. Lisäksi VSTO:n Access-ohjelmointiominaisuuksista ei ole juurikaan julkisuudessa puhuttu. Näin ollen VBA säilynee osana Accessia jatkossakin. Jos Accessia voidaan jatkossakin ohjelmoida VBA:lla, niin luultavasti sama pätee myös muihin Officen osiin. Access sinällään on Officen osa siinä missä PowerPoint tai Outlook, eikä VSTO:lla Accessin ohjelmoiminen juuri poikenne muiden Office-ohjelmien ohjelmoimisesta. /13. s. 134/

VBA on VSTO:a helpompi oppia, sillä VSTO:lla makrojen nauhoittaminen ei toistaiseksi onnistu vaivattomasti. VSTO:a voidaan käyttää useammalla kielellä, kuten Visual Basic .Netillä ja C#:lla. Nämä ovat kuitenkin jonkin verran VBA:ta hankalampia oppia käyttämään. Myös hinta puoltaa VBA:n valintaa, sillä VBA tulee Officen mukana. VSTO:a varten tarvitaan Visual Studio, joka helposti kaksinkertaistaa kustannukset pelkkään Office-pakettiin verrattuna. Kolmas VBA:ta puoltava asia on VSTO:n uutuus, sillä uuteen siirtyminen vie aina jonkin aikaa. VBA:lla on jo ohjelmoitu viimeisen sukupolven aikana varsin

paljon valmiita ohjelmistoja. Toisaalta VBA-ohjelmien siirtäminen VSTO:oon on suhteellisen helppoa, jolloin kynnys siirtyä VSTO:oon pienenee. Neljäs oleellinen asia on VSTO-kirjallisuuden puute. Niitä on olemassa muutamia, mutta ne ovat harvassa VBA-kirjoihin verrattuna. VSTO-kirjallisuutta ilmestyy kuitenkin jatkuvasti lisää.

VSTO on VBA:ta parempi järeässä ohjelmoinnissa, sillä VBA:ssa ei esimerkiksi luokkien periyttäminen ole täysin mahdollista. Lisäksi Visual Studio on tiimityökaluineen huomattavasti monipuolisempi, nykyaikaisempi ja ammattimaisempi ohjelmointiympäristö kuin VBE. Ohjelmoinnin ollessa vain pienimuotoista automatisointia VBA:n hyvät puolet vievät voiton, mutta aloitettaessa isoa ohjelmistotuotantoa VSTO:n käyttö on monesti järkevämpää. /14/

2.1.4. VBE:n puutteita

VBE:n suurimpia puutteita on rivinumeroinnin puuttuminen, moduulien koko ja niiden jakaminen ryhmiin. Rivinumerointi on käytössä todella monessa ohjelmassa, kuten esimerkiksi Wordissa. Rivinumeroiden avulla varsinkin ohjelmakoodissa on helpompi viitata tiettyyn paikkaan. Moduulien koko on rajoitettu 64 kiloon, jolloin se ohjaa ohjelmoijia käyttämään useampia moduuleja. Niitä on hankala jakaa ryhmiin muuten kuin moduulin sisältävän tiedoston perusteella, joten välistä ikään kuin puuttuu yksi porras. Olisi huomattavasti kätevämpää luoda projektien sisälle omia kansioita ja niiden alle taas alikansioita. Nämä kaikki ongelmat on kuitenkin korjattu Visual Studiossa ja sitä kautta VSTO:ssa. /14/

2.1.5. VBE:n ohjelmoiminen

VBE on osa Officea, joten VBE:n ohjelmointia varten tarvitaan viittaus VBE-objektiin. Yleensä tämä tapahtuu Tools-valikon References-kohdan alta. Avautuvasta listasta merkitään Microsoft Visual Basic for Applications Extensibility 5.3, minkä jälkeen seuraava ohjelmakoodi toimii.

```
Dim oVBE as VBIDE.VBE
Set oVBE = Application.VBE
```

Ensimmäinen rivi määrittää VBIDE-luokkaan kuuluvan VBE-tyyppisen objektin nimeltä oVBE. Toinen rivi asettaa oVBE-muuttujan viittaamaan VBE-ohjelmaan. /3, s. 572/

VBE:hen voidaan ohjelmoida esimerkiksi uusi toiminto, jonka avulla voidaan avata tiedostoja. Sen avulla ei tarvitse mennä takaisin ohjelmaan, josta VBE avattiin. /3, s. 580 - 588/

2.1.6. Tietolähteitä

VBE:stä on hankalaa löytää tietoa erillisistä lähteistä. Käytännössä kaikki saatava materiaali käsittelee VBA:ta. VBA-kirjoista löytyy tietoa VBE:n ohjelmoinnista, kuten valikoi-

den ja liitännäisten lisäämisestä. Internetin hakukoneilla haettaessa valtaosa VBE:tä koskevasta tiedosta kertoo VBE:n olevan VBA:n ohjelmointiin käytettävä ympäristö, mutta VBE:stä harvoin löytyy mitään sen syvällisempää tietoa. Toisaalta tämä on ymmärrettävää, sillä VBE:n tarkoitus on kuitenkin olla vain VBA:n kehitysympäristö. VBE ei loppujen lopuksi ole kovin monimutkainen ohjelma. Olisi yllättävää, jos jostain löytyisi tuhannen sivun kirja, joita esimerkiksi Visual Studiosta on kirjoitettu useita.

2.2. Visual Basic for Applications

Visual Basic for Applications on ollut osa Exceliä versiosta 5 lähtien. VBA:n voidaan ajatella olevan Microsoftin yleinen ohjelmien käyttämä script-kieli, ja se on käytössä valtaosassa Officen osista. Tästä syystä on helppoa ohjelmoida muita ohjelmia, kun VBA:ta on kerran oppinut yhdellä ohjelmalla käyttämään. Suurin ero VBA:ssa eri ohjelmien välillä on ohjelmien käyttämät erilaiset objektimallit. /13. s. 134/

2.2.1. Syntaksi

Visual Basiciin vahvasti pohjautuva VBA muistuttaa syntaksiltaan läheisesti Visual Basicia. VBA tarvitsee toimiakseen normaalisti isäntäohjelman, kuten Excelin, ympärilleen. VBA:ssa on kahdenlaisia proseduureja, aliproseduureja ja funktioita. Aliproseduureista käytetään yleisesti nimitystä sub. /3, s. 10/

VBA:ssa komennot loppuvat rivin loppuun, eli toisin kuin esimerkiksi C:ssä ja Javassa. Komentoja voi kuitenkin jakaa useammalle riville käyttämällä rivin lopussa välilyönnin ja alaviivan yhdistelmää. Muita vastaavia eroja C:hen ja Javaan on lopettamisesta erikseen ilmoittavat End- ja Next-komennot sekä aaltosulkeiden puuttuminen lähes täysin. /6/

Kommentointi VBA:ssa tapahtuu yksittäisellä heittomerkillä. Sen sijaan lohkokommentteja VBA:ssa ei ole. Taulukossa 2 on listattuna joitakin VBA:n varattuja sanoja.

Taulukko 2. Joitakin esimerkkejä VBA:n varatuista sanoista /2/

And	As	Boolean	ByRef	Byte	ByVal
Call	Case	CBool	CByte	CDate	CDbl
CInt	CLng	Const	CSng	CStr	Date
Declare	Dim	Do	Double	Each	Else
ElseIf	End	EndIf	Error	False	For
Function	Get	GoTo	If	Integer	Let
Lib	Long	Loop	Me	Mid	Mod
New	Next	Not	Nothing	Option	Or
Private	Public	ReDim	REM	Resume	Select
Set	Single	Static	Step	String	Sub
Then	To	True	Until	vbCrLf	When
Where	With	While			

2.2.2. Muuttujat

Koska VBA on heikosti tyypitetty kieli, muuttujia ei ole pakko esitellä etukäteen. Tämä on kuitenkin suositeltavaa. Normaali muuttuja esitellään käyttämällä avainsanaa Dim. Muuttujan nimen tulee alkaa kirjaimella, ja erikoismerkkien käyttöä muuttujien nimessä tulisi välttää. Muuttujan nimi voi olla korkeintaan 255 merkkiä pitkä. Seuraavassa esimerkissä esitellään muuttuja x.

Dim x

Muuttujalla x ei kuitenkaan vielä ole mitään tietotyyppiä. Tämä ei ole virhe, sillä Excel käsittelee muuttujaa tyyppinä Variant. Variant-tietotyyppi vie kuitenkin paljon muistia, joten tiedettäessä muuttujan sisältö tarkemmin voidaan se esitellä tietyntyyppiseksi. Tämä onnistuu avainsanan As avulla. Seuraavassa esimerkissä esitellään muuttuja x olemaan tietotyyppiä pitkä kokonaisluku.

Dim x As Long

Taulukossa 3 on listattu tämän opinnäytetyön kannalta merkittävimmät VBA:n tietotyypit. Kuten taulukosta nähdään, oikean tietotyypin valinnalla voidaan säästää suuria määriä muistia.

Taulukko 3. Opinnäytetyön kannalta oleellisimmat VBA:n tietotyypit /3, s. 42–43/

Tietotyyppi	Muistin käyttö	Arvoalue
Integer	2 tavua	-32,768 … 32,767
Long (pitkä kokonaisluku)	4 tavua	-2,147,483,648 … 2,147,483,647
String (muuttuva pituus)	10 tavua + merkkijonon pituus	0 … noin kaksi miljardia merkkiä
Variant (merkkejä käyttävä)	22 tavua + merkkijonon pituus	Sama alue kuin muuttuvan pituuden merkkijonolla.

2.2.3. Taulukkomuuttujat

VBA:ssa voi muuttujien tapaan tallentaa tietoa myös taulukkomuuttujiin, joiden ero muuttujiin on mahdollisuus tallentaa useampi eri arvo. Taulukkomuuttuja on periaatteessa vastaavanlainen kuin Excel-taulukon solualue. Näiden välillä tietoa voidaan myös siirtää kätevästi. Merkittävimmät erot taulukkomuuttujien ja solualueiden välillä ovat solualueiden mahdollisuus käyttää muotoiluja ja taulukkomuuttujien paljon nopeampi käsittely. Seuraavassa esimerkissä esitellään merkkijonoja sisältävä taulukkomuuttuja.

Dim aStr() As String

Tässä taulukkomuuttujassa ei kuitenkaan ole vielä yhtään paikkaa. Taulukkomuuttujan paikkojen määrää voidaan muuttaa ReDim-määritteellä. Seuraavassa esimerkissä tehdään edellisessä esimerkissä esiteltyyn taulukkomuuttujaan kaksi paikkaa.

ReDim aStr(1 To 2)

Tämä kuitenkin hävittää kaikki taulukkomuuttujassa ennestään olevat tiedot. Tiedot voidaan säilyttää käyttämällä määritettä Preserve. Tulee kuitenkin huomata, että Preserve-määritettä käytettäessä voidaan suoranaisesti muuttaa vain ylimmän ulottuvuuden elementtien määrää. Seuraavassa esimerkissä suurennetaan edellisen esimerkin taulukkomuuttujaa säilyttäen samalla tiedot.

ReDim Preserve aStr(1 To 3)

Taulukkomuuttujassa voi olla jopa 60 ulottuvuutta. Ulottuvuuksia tarvitaan harvoin kahta tai kolmea enempää. Seuraavassa esimerkissä esitellään taulukko, jossa on kaksi ulottuvuutta. /3, s. 57/

aInt(1 To 2, 2 To 3)

Kuten edellä olevasta esimerkistä nähdään, taulukkomuuttujien raja-arvot voivat olla erikoisempiakin. Tätä varten ovat olemassa funktiot UBound () ja LBound(), joiden avulla saadaan selville taulukkomuuttujan ylä- ja alarajat tässä järjestyksessä. Edellä mainitut funktiot ovat erityisen käteviä silmukoiden yhteydessä. Seuraava esimerkki tulostaa konsoliin edellisen esimerkin taulukkomuuttujan toisen ulottuvuuden ylärajan eli 3. /3, s. 57/

```
Debug.Print UBound(aInt, 2)
```

2.2.4. VBA:n ohjausrakenteet

Ohjausrakenteet ovat oleellinen osa mitä tahansa ohjelmointikieltä. Seuraavaksi on esitelty tämän opinnäytetyön kannalta oleellisimmat VBA:n ohjausrakenteet. Ohjausrakenteista on annettu yksinkertaisia esimerkkejä.

Vertailuoperaattorit

Muiden ohjelmointikielien tapaan vertailuoperaattorit ovat oleellinen osa myös VBA:ta. VBA:ssa on kuusi vertailuoperaattoria ja ne on listattu taulukossa 4.

Taulukko 4. VBA:n vertailuoperaattorit /13, s. 207/

Operaattori	Tosi jos	Epätosi jos
< (Pienempi kuin)	lauseke1 < lauseke2	lauseke1 >= lauseke2
<= (Korkeintaan)	lauseke1 <= lauseke2	lauseke1 > lauseke2
> (Suurempi kuin)	lauseke1 > lauseke2	lauseke1 <= lauseke2
>= (Vähintään)	lauseke1 >= lauseke2	lauseke1 < lauseke2
= (Samanarvoinen)	lauseke1 = lauseke2	lauseke1 <> lauseke2
<> (Eriarvoinen)	lauseke1 <> lauseke2	lauseke1 = lauseke2

If-rakenne

If-rakenteella voidaan toteuttaa ehdollisesti osa ohjelmaa. Seuraavassa esimerkissä verrataan muuttujaa x numeroon neljä ja tulostetaan konsoliin vertailun tulos.

```
If x > 4 Then
        Debug.Print "x on suurempi kuin 4"
Else
        Debug.Print "x ei ole suurempi kuin 4"
End If
```

Select Case -rakenne

Select Case -rakenteen avulla voidaan suorittaa ohjelmasta tietty osa. Select Case -ratkaisun voisi toteuttaa myös If-rakenteen avulla, mutta ohjelmasta tulee selkeämpi ja lyhyempi Select Casea käyttäen. Seuraavassa esimerkissä tarkastellaan muuttujan x arvoa ja tulostetaan konsoliin tarkastelun tulos.

```
Select Case x
        Case 0
                Debug.Print "x on nolla"
        Case 1 To 9
                Debug.Print "x on yksinumeroinen"
        Case 10 To 99
                Debug.Print "x on kaksinumeroinen"
        Case Else
                Debug.Print "x on jotain muuta"
End Select
```

For…Next -silmukkarakenne

For…Next -silmukkaa käyttäen voidaan suorittaa joku tietty osa ohjelmaa, kunnes tietty ehto täyttyy. Oletuksena silmukkamuuttujaa kasvatetaan yhdellä, mutta avainsanan Step avulla tätä voidaan muuttaa. Seuraava esimerkki tulostaa konsoliin parilliset numerot kahdesta kymmeneen.

```
For x = 2 to 10 Step 2
        Debug.Print x
Next x
```

For Each…Next -silmukkarakenne

For Each…Next -silmukan avulla voidaan käydä läpi kaikki kokoelmaan kuuluvat jäsenet. Seuraava esimerkki muuttaa nykyisen työkirjan sivun alatunnisteita, jolloin jokaisen taulukon sivun alatunnisteeksi tulee nykyisen työkirjan tiedostopolku.

```
For Each wks In Worksheets
        wks.PageSetup.CenterFooter = sFilePath
Next wks
```

With…End With -rakenne

With…End With -rakenteen avulla voidaan suorittaa useita operaatioita yksittäiselle objektille. Tämä nopeuttaa ja selkeyttää ohjelmakoodia. Seuraava esimerkki suorittaa operaatioita objektille ilman With…End With -rakennetta.

```
Selection.Font.Name = "Cambria"
Selection.Font.Bold = True
Selection.Font.Italic = True
Selection.Font.Size = 12
Selection.Font.Underline = xlUnderlineStyleSingle
Selection.Font.ThemeColor = xlThemeColorAccent1
```

Seuraava esimerkki päätyy samaan lopputulokseen, mutta nopeammin ja yksinkertaisemmin. Myös kirjoitettavaa ohjelmakoodia on huomattavasti vähemmän.

```
With Selection.Font
        .Name = "Cambria"
        .Bold = True
        .Italic = True
        .Size = 12
        .Underline = xlUnderlineStyleSingle
        .ThemeColor = xlThemeColorAccent1
End With
```

3. OHJELMAN SUUNNITTELU JA TOTEUTUS

Ohjelman toteuttaminen jakaantuu neljään eri vaiheeseen: esiselvityksiin, suunnitteluun, toteutukseen ja testaamiseen. Seuraavissa otsikoissa kuhunkin työvaiheeseen perehdytään tarkemmin.

3.1. Vaatimusmäärittely

Käyttäjän tulee pystyä mahdollisimman rajoittamattomasti määrittämään haettavat tiedot. Tätä varten käyttäjällä tulee olla mahdollisuus kirjoittaa ohjaustiedostoja ohjelmaa varten. Ohjelman tulee myös pystyä käsittelemään kommenteiksi ja merkkijonoiksi määriteltyä tietoa.

Ohjelman tulee toimia mahdollisimman täydellisesti suomeksi ja englanniksi. Lisäksi ohjelman tukemia kieliä pitää voida lisätä mahdollisimman helposti.

Excelin omia ominaisuuksia, kuten työkirjafunktioita, tulee välttää. Ohjelman tulee pystyä kirjoittamaan tulostiedostoja niin, ettei siihen vaadita Exceliä. Tulostiedostoja pitää pystyä kirjoittamaan myös muilla Officen ohjelmilla.

Tiedostojen käsittelyssä tulee huomioida virhetilanteet, kuten kirjoitussuojaukset. Käyttäjälle pitää ilmoittaa, ettei tiedostoa voida luoda, jotta käyttäjä voisi valita uuden tiedoston. Tämän tulee tapahtua mahdollisimman nopeasti haun käynnistämisen jälkeen. Ohjelman toiminnan aikana käyttäjää tulee tiedottaa mahdollisista pitkistä odotusajoista. Ohjelman pitää myös pystyä reagoimaan käyttäjän antamiin keskeytyskomentoihin hakujen aikana. Ohjelma ei kuitenkaan saa kaatua keskeytyksiin.

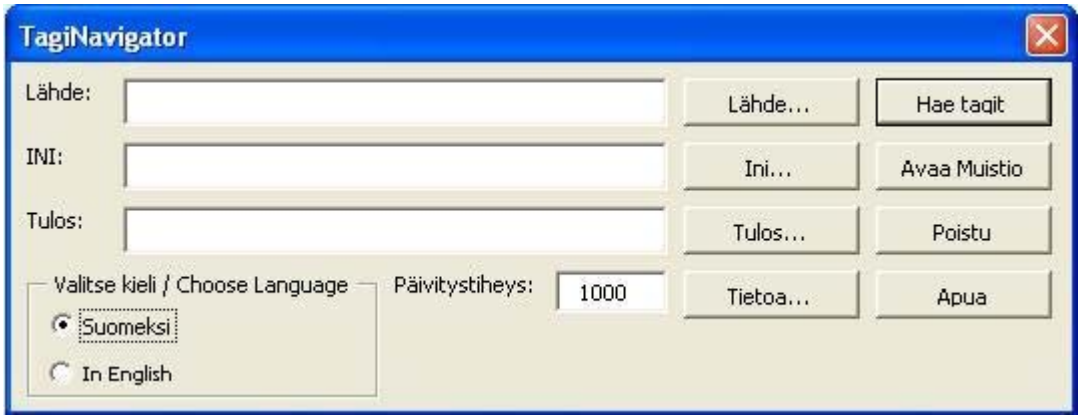
Internet-selainta voidaan ohjata lähettämällä selaimelle suoraan näppäinkomentoja SendKeys-funktion avulla. Sitä tulee kuitenkin välttää, koska Internetistä sivuja haettaessa on tärkeää pyrkiä ennakoimaan virhetilanteita. Tällainen virhetilanne voi syntyä esimerkiksi Internet-sivun latauksen hitauden takia. Internet-selainta tulee siis ohjata automatisoimalla objektimallin kautta.

3.2. Ohjelman suunnittelu

VBA on kieli, jonka avulla pystyy käsittelemään tietoa joko suoraan taulukkomuuttujissa tai epäsuorasti muualla, esimerkiksi Excel-taulukon soluissa. Jälkimmäisessä vaihtoehdossa on hyvänä puolena sen tuoma selkeys, sillä sitä kautta pystyy koko ajan seuraamaan tiedonkäsittelyn etenemistä. Huonona puolena on tavan hitaus ja Excel-sidonnaisuus. Tästä on olemassa myös välimuoto, eli ohjelma voi käsitellä tietoa taulukkomuuttujissa ja välillä

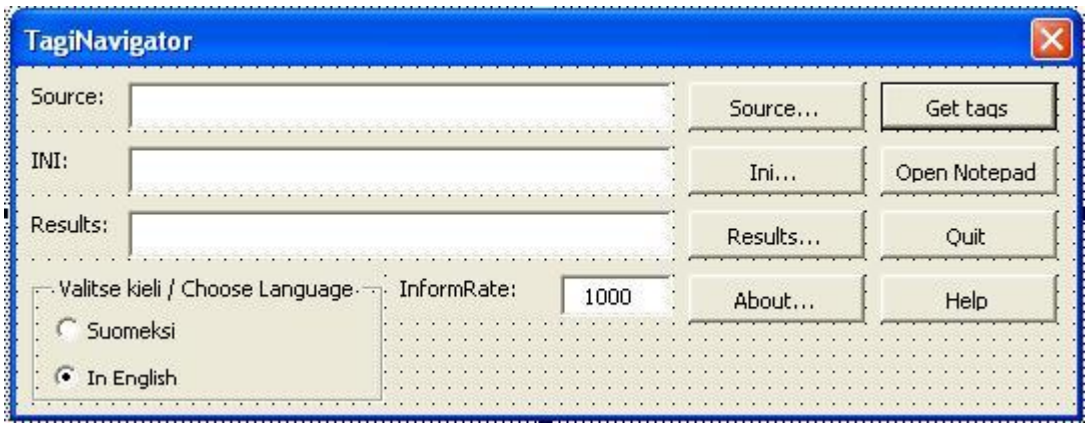
kirjoittaa tuloksia Excelin soluihin. Tätä kautta testaaminenkin on huomattavasti helpompaa.

Ohjelman suunnittelu aloitettiin käyttöliittymän suunnittelusta. Käyttöliittymän tulisi olla mahdollisimman yksinkertainen, mutta sen tulisi myös pystyä helposti reagoimaan kielen vaihtamiseen. Kuvassa 2 on esitettynä ohjelman suomenkielinen käyttöliittymä.



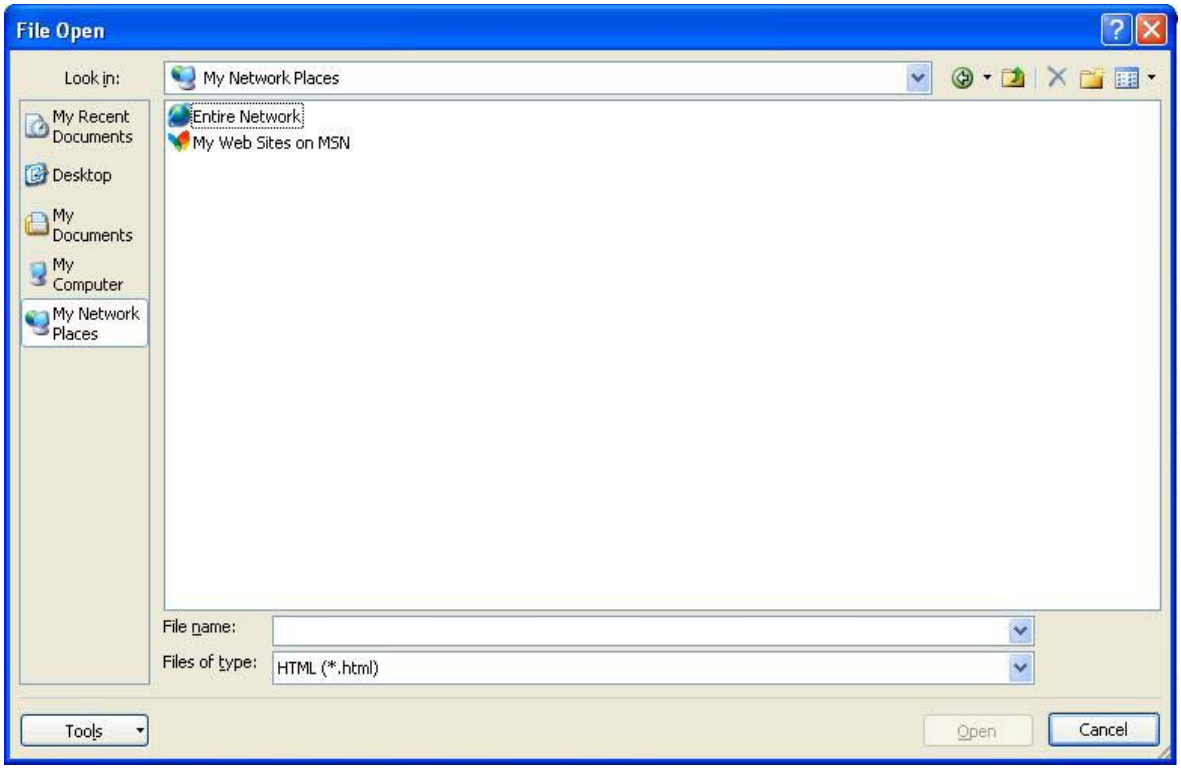
Kuva 2. Ohjelman suomenkielinen käyttöliittymä

Käyttöliittymälle on oleellista pystyä esittämään samat tiedot mahdollisimman vastaavan kokoisina myös englanniksi, jotta painonäppäimiä ei tarvitsisi siirtää lomakkeella. Kielten välillä tulee pystyä siirtymään yhden painonäppäimen painalluksella. Kuvassa 3 on esitetty ohjelman englanninkielinen käyttöliittymä.



Kuva 3. Ohjelman englanninkielinen käyttöliittymä

Ohjelman helppokäyttöisyyden kannalta tiedostoja tulisi pystyä selaamaan muiden Windows-ohjelmien tapaan. Tätä varten käytetään File Open Dialog -objektia. Kuvassa 4 on esitetty lähdetiedoston valitsemiseen tarkoitettu tiedostonavaamisikkuna.



Kuva 4. File Open Dialog -ikkuna

3.3. Tagit

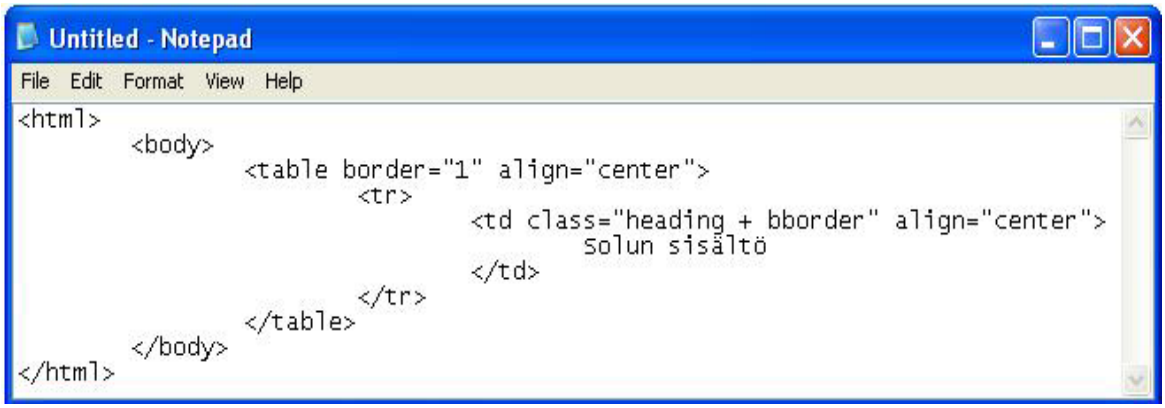
Ohjelman oleellisin osa on tagien tunnistaminen. Tagilla tarkoitetaan mitä tahansa merkkijonoa, jolle käyttäjä on määrittänyt INI-tiedostossa merkityksen. HTML:ssä tageja ovat esimerkiksi <HTML>, </TABLE> ja
. Tässä opinnäytetyössä toteutetussa ohjelmassa ennalta määriteltyjä tageja ei ole.

3.4. Ohjelman kehitysvaiheet

Ohjelman toteutus alkoi etsimällä mahdollisia tapoja käydä läpi lähdetiedostoa. Tapoja löytyi kaksi. Ohjelma voi lukea tietoa lähdetiedostosta merkki kerrallaan tai etsiä jokaisen halutun tagin, seuraavan alkukohdan. Toisessa vaihtoehdossa olisi mahdollisesti voinut ohittaa osan lähdetiedoston merkeistä, mutta ohjelma päädyttiin toteuttamaan ensimmäisellä tavalla. Tähän suurin syy oli lähestymistavan mahdollistama tapa lukea yksittäinen merkki vain kerran.

Toisena kohtana toteutettiin osio, joka palauttaa kulloinkin halutut tagit. Tagit eivät saa esiintyä kuin niille tarkasti varatuissa paikoissaan, jotka käyttäjä saa kuitenkin vapaasti määritellä. Käytännön esimerkkinä mainittakoon HTML-tiedosto, jossa esimerkiksi taulun aloittava <table>-tagi ei saa esiintyä, ennen kuin sivun aloittava <html>-tagi on löytynyt. Toteutettavassa ohjelmassa tämä ei aiheuta minkäänlaista virhettä tai ongelmaa, vaan vää-

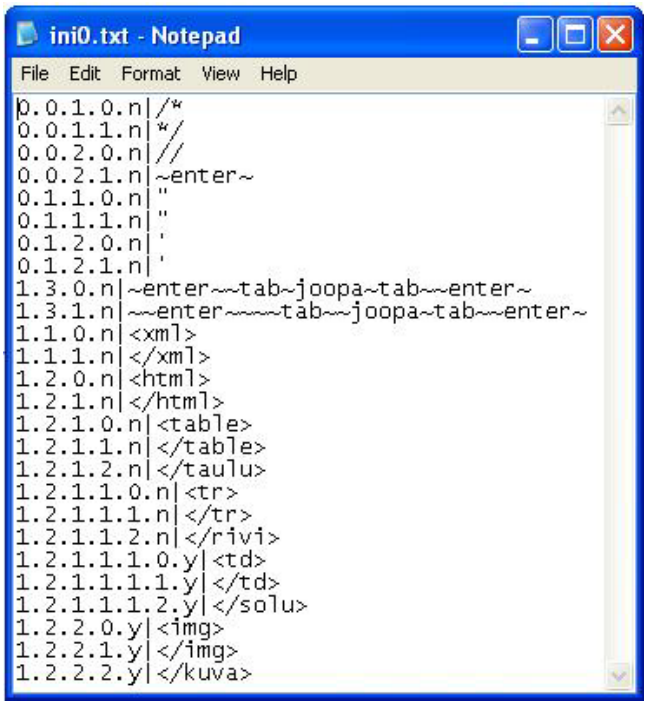
rässä paikassa sijaitsevaa tagia käsitellään normaalina merkkijonona. Tällöin sitä ei vain yksinkertaisesti tulkita tagiksi. Kuvassa 5 on esitetty tyypillinen HTML-tiedoston sisennetty rakenne.



Kuva 5. HTML-tiedoston sisennetty rakenne

Seuraavaksi ohjelmassa toteutettiin käyttäjän mahdollisuus määrittää, mitä tageja lähdetiedostosta halutaan etsiä. Tätä varten suunniteltiin oma INI-tiedostomuoto. INI-tiedostossa käyttäjä voi määritellä haettavat tiedot ja INI-tiedosto on kirjoitettava sitä varten erikseen kehitetyllä kielellä. Jokaisesta halutusta tagista annetaan tietoina tagin taso epäsuorasti, tagin vaadittu sijainti lähdetiedostossa, onko tagi avaava vai sulkeva, halutaanko tagin sisältö tallentaa ja millainen tagi on. Tässä ohjelmassa käytettävä INI-tiedosto ei kuitenkaan ole sama asia kuin Windowsin INI-tiedosto.

Yhtä tagia kohden kirjoitetaan yksi rivi INI-tiedostoon, ja INI-tiedoston rivien on oltava yksilöllisiä. Tämä ei kuitenkaan tarkoita sitä, ettei jotakin tagia voisi määrittää etsittäväksi useammasta paikasta, vaan sitä, että tagitunniste pitää vain merkitä eri tavalla. Tagitunnisteen ja tagin ero on siinä, että tagitunniste vastaa kysymykseen ”mistä haetaan?”, kun taas tagi vastaa kysymykseen ”mitä haetaan?”. Tagin tyyppi voi olla kommentti, merkkijono tai normaali. Kuvassa 6 on esimerkki INI-tiedostosta. INI-tiedoston kirjoittamista käsitellään puolestaan tarkemmin omassa kappaleessaan.



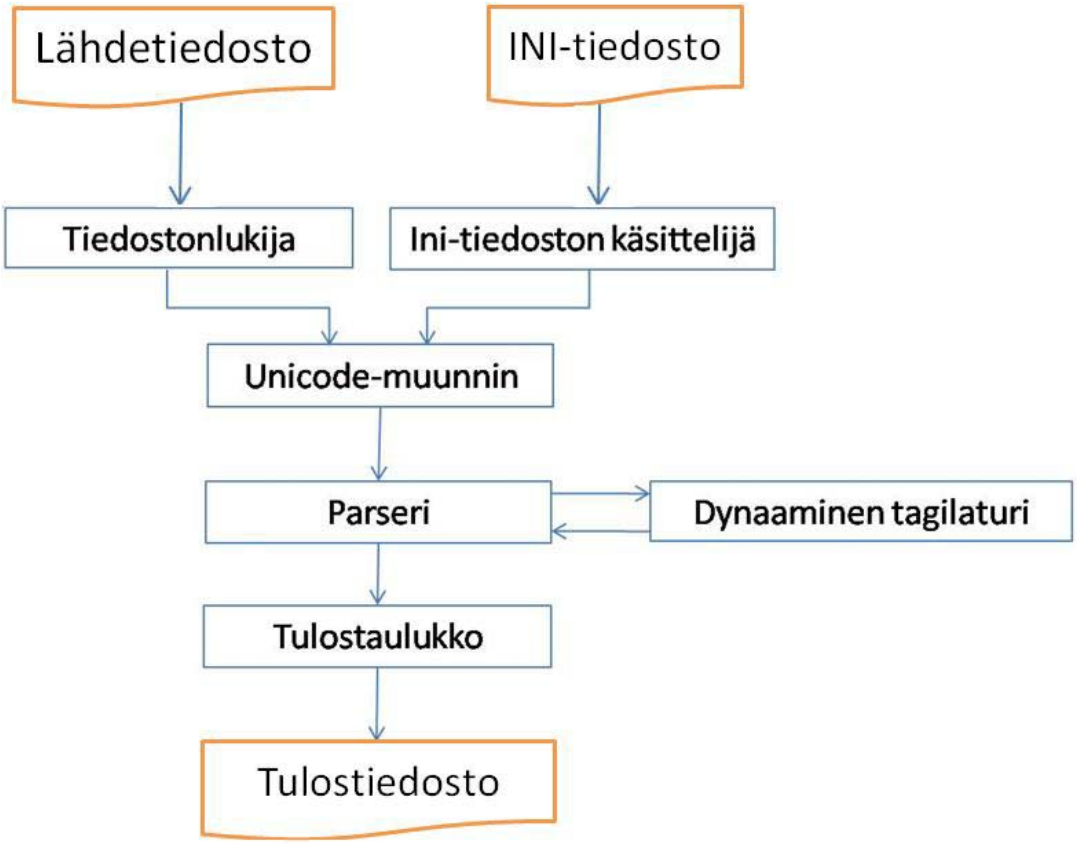
Kuva 6. Esimerkki INI-tiedostosta

Neljäntenä osiona ohjelmaan kehitettiin mahdollisuus hakea lähdetiedosto Internetistä. Tässä hyödynnettiin VBA:n mahdollisuutta viitata Internet Explorerin objektimalliin. Tuloksena saatu lähdetiedosto on merkkijono, jota käsitellään kuten tekstitiedostosta haettua lähdetietoa. Lähdetiedoston muuntamista merkkijonomuotoon käsitellään myöhemmin tässä opinnäytetyössä omana kappaleenaan.

Ohjelmaan päätettiin rakentaa myös erikoismerkkien tuki, ja tämä toteutettiin kääntämällä kaikki käsiteltävä tieto Unicode-muotoon. Unicode-muodossa kukin merkki esitetään heksadesimaalinumerosarjana, joskin tässä ohjelmassa normaalista Unicode-muodosta poiketaan hieman. Jokaista merkkiä merkitään neljällä heksadesimaalinumerolla ja alaviivalla, eli esimerkiksi merkkijono ”ja” muuttuu muotoon ”006A_0061_”. Tämä mahdollistaa sen, että tagitiedossa voidaan käyttää rivinvaihtoja ja muita erikoismerkkejä sekä tarvittaessa myös kyrillisiä aakkosia. Unicode tukee kymmeniä tuhansia merkkejä. Joitain erikoismerkkejä voidaan käyttää INI-tiedoston kirjoittamisessakin. Tästä syvällisempää tietoa löytyy INI-tiedoston kirjoittamista tarkemmin käsittelevässä kappaleessa.

Ohjelman tulostenkäsittely päätettiin toteuttaa mahdollisimman helposti laajennettavaksi. Tästä johtuen tulokset muodostetaan yhteen paikkaan, joka ei riipu millään tavalla halutusta tulostiedostomuodosta. Tämä paikka on taulukkomuuttuja, johon kaikkien löytyneiden tagiparien tiedot tallennetaan niiden löytymisen yhteydessä.

Edellämainitusta taulukkomuuttujasta, eli tulostaulukosta, muodostetaan tulostiedosto. Ohjelmaan toteutettiin kaksi erilaista tallennusvaihtoehtoa, Excel-tiedosto ja HTML-tiedosto. Excel-tiedosto luo tulosesityksensä pohjautuen Excel-työkirjan taulukoihin. HTML-tiedosto kirjoitetaan ensin tekstitiedostoon, joka sitten tallennetaan HTML-muodossa. Ohjelman yleinen toimintaperiaate on esitetty kuvassa 7.



Kuva 7. Ohjelman toimintaperiaate

4. OHJELMAN OSIOT

Ohjelma koostuu seitsemästä eri osiosta: INI-tiedoston käsittelijästä, tiedoston jäsentäjästä, eli parserista, dynaamisesta tagilaturista, tulosten esittäjästä, Unicode-muuntajasta, lähdetiedoston lukijasta sekä Internet-sivujen lukijasta. Seuraavissa otsikoissa kuhunkin osioon perehdytään tarkemmin.

4.1. INI-tiedoston käsittely

INI-tiedostosta muodostetaan lista kaikista määritellyistä tageista syöttämällä osiolle tietoa lähde- ja tulostiedostoista. INI-tiedostolle suoritetaan useita tarkistuksia, joiden avulla varmistetaan, että INI-tiedosto vastaa määriteltyä muotoa. Käytännössä kaikki virheet ovat kriittisiä ja siten ne päättävät ohjelman näytettyään käyttäjälle virheilmoituksen.

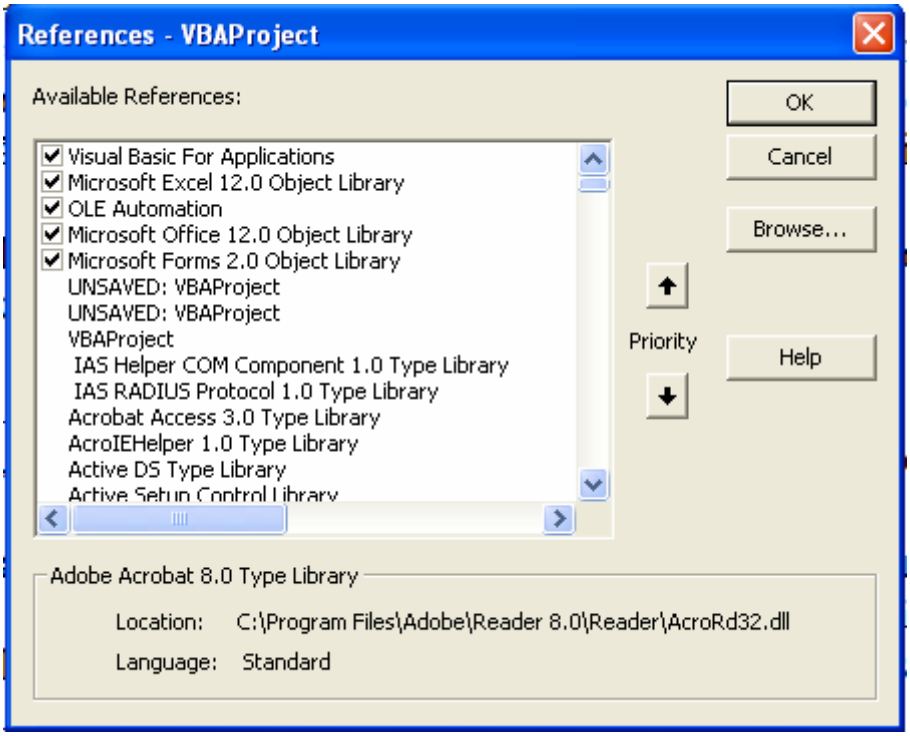
Ensimmäiset tarkistukset koskevat tiedostojen ja hakemistojen olemassaoloa. Mikäli tulostiedoston kansiota ei ole olemassa, se luodaan. Jos sitä ei pystytä luomaan, aiheutuu virhe. On erittäin todennäköistä, että tässä tapauksessa käyttäjä yritti luoda kansion paikkaan, jonne käyttäjällä ei ole oikeuksia päästä. Tällaisesta paikasta esimerkkinä mainittakoon optisen median asemat.

Seuraavaksi tarkistetaan, onko INI-tiedosto olemassa. Tähän tilanteeseen joudutaan, jos käyttäjä on esimerkiksi vahingossa kirjoittanut väärin INI-tiedoston sijainnin.

Viimeinen tiedostojen olemassaoloon liittyvä tarkistus koskee lähdetiedostoa. Lähdetiedoston tiedostopolusta tarkistetaan, onko kyseessä Internet-lähde vai paikallinen lähde.

Tiedostojen olemassaolon tarkistuksen jälkeen INI-tiedosto luetaan taulukkomuuttujaan. Tässä proseduurissa on tietoisesti käytetty tietojen myöhäistä sitomista, sillä siten käyttäjän ei tarvitse muodostaa itse viittausta FileSystemObjectiin. Tiedosto luetaan käyttämällä FileSystemObjectin metodeja. Kuvassa 8 on lista Excel-projektin viittauksista muihin ohjelmiin, jotka on sidottu käyttäen aikaista sitomista. Ohjelmien, joihin viittaus on tehty, ohjelmointi on huomattavasti helpompaa. Seuraavassa esimerkissä myöhäisen sitomisen avulla otetaan käyttöön tiedostonlukija.

```
Dim FSOone As Object
Dim FSOtwo As Object
Set FSOone = CreateObject("Scripting.FileSystemObject")
Set FSOtwo = FSOone.OpenTextFile(strFileName)
```



Kuva 8. Viittauksien muodostaminen Excel-projektin ulkopuolelle

Tiedoston lukemisen jälkeen tiedosto jaetaan osiin riveittäin. Tämän jälkeen tarkistetaan, että rivejä oli ylipäätään olemassa. Jos rivejä ei ole olemassa, aiheutuu virhe. Tämä puolestaan johtuu siitä, että jokainen rivi vastaa haluttua tagia. Mikäli rivejä ei ole, niin mitään ei haluta vastaukseksi. Siten ohjelman suorittamisen jatkaminen olisi turhaa työtä, koska tulos tiedetään etukäteen.

Kun INI-tiedosto on saatu luettua, järjestetään vastauksena tulleet rivit nousevaan aakkosjärjestykseen. Tässä tarkistetaan varmuuden vuoksi uudelleen, että jotain järjestettävää oikeasti oli.

Seuraava tarkistus koskee tagitunnisteiden ja tagien jakamista tietyllä tavalla kahteen osaan. Kieleen määritettiin, että tämä jakaminen tulee olla yksittäinen pystyviiva, eli niin sanottu putkimerkki. Tämä tapahtuu yksinkertaisesti käymällä silmukkamuuttujassa läpi jokainen rivi ja vastaavasti jokaiselta riviltä tarkistamalla löytyykö riviltä pystyviivaa. Jos pystyviivaa ei löydy, rivillä on virhe. Seuraavassa esimerkissä tarkistetaan, löytyikö riviltä pystyviivaa.

```
For l = 1 To UBound(vTagiArray)
   lSijainti = InStr(vTagiArray(l), "|")
Next l
```

Tagierottimen tarkistamisen jälkeen tagit jaetaan kahteen osaan, tagitunnisteeseen ja itse tagiin. Samalla lasketaan löytyneiden tagien määrä. Jos tageja löytyi vähemmän kuin kaksi, tiedetään, että kyseessä on jonkinlainen virhe. Tageja tulee aina olla vähintäänkin yksi pari. Virheen tyyppiä ei sen tarkemmin eritellä.

Tagien jakamisen jälkeen tagit analysoidaan. Tässä analyysissa selvitetään tagin tyyppi, tagin taso, halutaanko tagi talteen, muunnetaan tagi Unicode-muotoon ja onko tagi avaava vai sulkeva. Kuvassa 9 kuvan 6 INI-tiedosto on jaettu osiin. Kuva 9 on testausvaiheesta. Lopullisessa ohjelmassa Excel-taulukoita ei käytetä.

0.0.1.0.	/*	002F_002A_	Kommentti	y	avaava	0
0.0.1.1.	*/	002A_002F_	Kommentti	y	sulkeva	0
0.1.1.0.	"	0022_	String	y	avaava	0
0.1.1.1.	"	0022_	String	y	sulkeva	0
1.3.0.	~enter~~tab~joopa~tab~~enter~	0010_0011_006A_006F_006F_0070_0061_0011_0010_	Normaali	n	avaava	1
1.3.1.	~~enter~~~~tab~~joopa~tab~~enter~	007E_0010_007E_007E_0011_007E_006A_006F_006F_0070_0061_0011_0010_	Normaali	n	sulkeva	1

Kuva 9. INI-tiedosto jaettuna osiin

Tagien analysoinnin jälkeen tarkistetaan, että tagitunnisteet ovat yksilöllisiä. Mikäli tagitunnisteet eivät ole yksilöllisiä, aiheutuu virhe. Seuraavassa esimerkissä tarkistetaan tagien yksilöllisyys

```
For lElement = LBound(vArray, 1) To UBound(vArray, 1) 'Käydään kaikki tagit läpi
   For lCheck = LBound(strUniques) To UBound(strUniques) 'Käydään uniikit läpi
      If vArray(lElement, LBound(vArray, 2)) = strUniques(lCheck) Then 'Tupla!
           'Tiedotetaan käyttäjää
         End 'Poistutaan ohjelmasta
      End If
   Next lCheck 'Seuraava uniikki

    'Tagia ei löytynyt, joten lisätään tagi kokoelmaan
   strUniques(UBound(strUniques)) = vArray(lElement, LBound(vArray, 2))

    'Lisätään paikka uniikkien tagien taulukkoon
   ReDim Preserve strUniques(0 To UBound(strUniques) + 1)
Next lElement 'Seuraava tagi
```

Seuraavaksi avaaville ja sulkeville tageille etsitään vastaava pari. Esimerkiksi kuvassa 9 yksi pari muodostuu viidenneltä ja kuudennelta riviltä. Mikäli paria avaavalle tagille ei löydy, kyseessä on virhe. Vastaavasti pari etsitään jokaiselle tallennettavaksi halutulle tagille. Tässä viidennen ja kuudennen rivin tagit eivät muodosta paria, sillä molemmat tagit on merkitty n-kirjaimella, jolloin niitä ei haluta tallentaa. Tagien luonnetta on käsitelty tarkemmin ohjelman käyttöohjeessa (kohta 4.9.).

Kun edellä mainitut tarkistukset on suoritettu virheittä, palautetaan muodostettu tagipuu funktion tuloksena. Lopuksi ilmoitetaan käyttäjälle INI-tiedoston olevan käsitelty ja siirrytään ohjelman seuraavaan osioon.

4.2. Tiedoston jäsentäjän toiminta

Ohjelma lukee lähdetiedostosta merkin kerrallaan ja vertaa merkkiä kaikkiin sillä hetkellä etsittäviin tageihin. Kun tagi löytyy, ohjelma lähettää eteenpäin tiedon siitä. Tällöin halutut merkkijonot päivitetään. Tämä prosessi on esitettynä kuvassa 10. Seuraavassa esimerkissä lähdetiedostona on merkkijono ”konekirjoituskone”, josta käyttäjä haluaa etsiä määrittelemiään tageja ”tieto”, ”tulostin” ja ”kone”.

Lähdetiedosto: konekirjoituskone
Etsitään: tieto, tulostin, kone
Ensimmäinen merkki: k
Täsmää tageihin: kone
Etsitään: tieto, tulostin, one
Toinen merkki: o
Täsmää tageihin: one
Etsitään: tieto, tulostin, ne
Kolmas merkki: n
Täsmää tageihin: ne
Etsitään: tieto, tulostin, e
Neljäs merkki: e
Täsmää tageihin: e
Tagi ”kone” löytyi, vaihdetaan tagit.

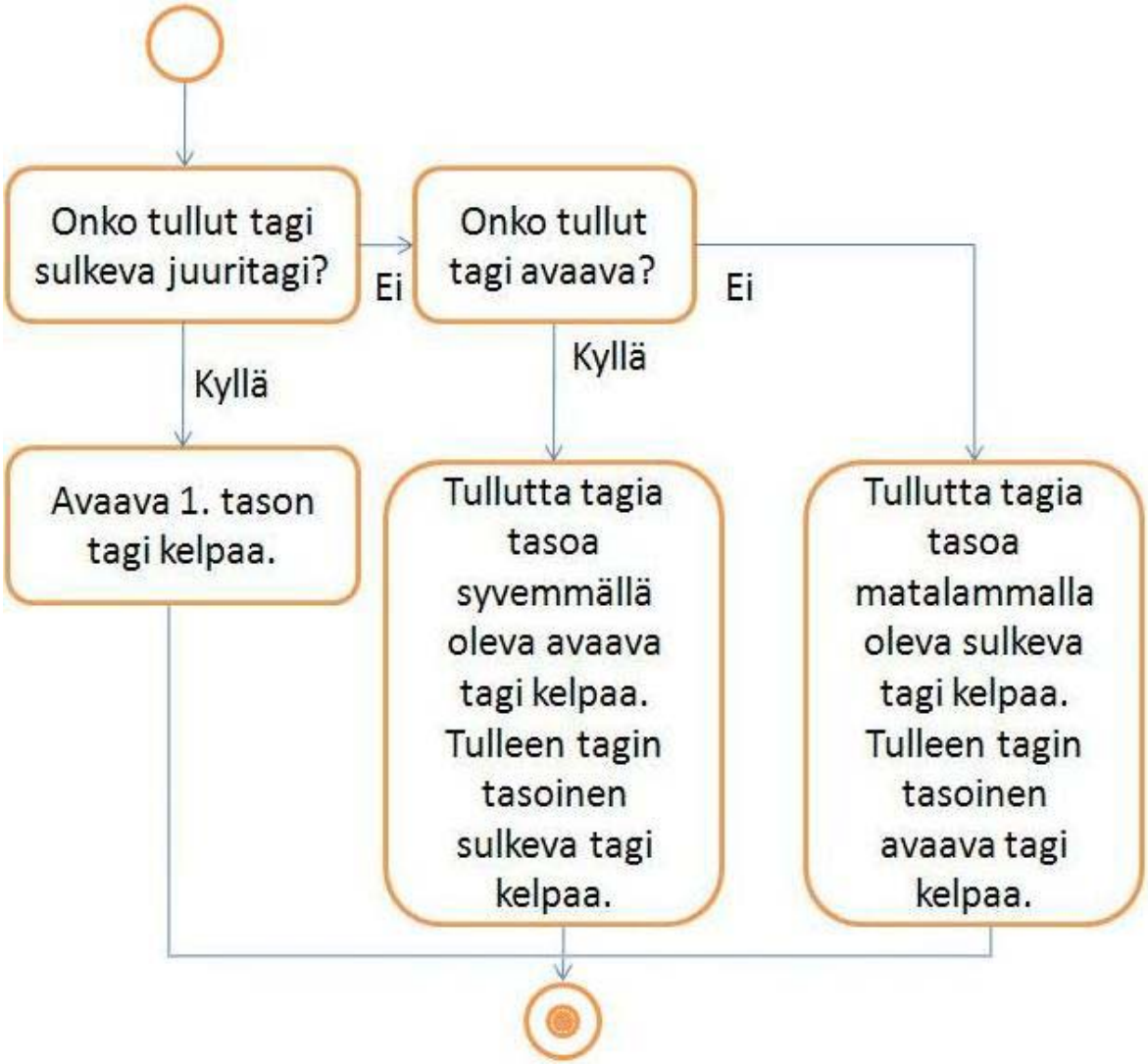
Kuva 10. Tiedoston jäsentäjän toiminta kaaviona

Seuraavassa esimerkissä kuvan 10 kaavio on esitettynä ohjelmakoodina. Ohjelmakoodi sijaitsee silmukassa, jossa käydään läpi jokainen tilanteeseen sopiva tagi. Kyseinen silmukka puolestaan sijaitsee silmukassa, jossa käydään läpi jokainen lähdetiedoston merkki.

```
If tulleet & uusin = Left(haettava, Len(tulleet) + 1) Then ’Osuiko?
    tulleet = tulleet & uusin
    If haettava = tulleet Then
        kokomerkkijono = True
    End If
Else ‘Ei osunut, nollataan koko jono
    tulleet = vbNullString
    kokomerkkijono = vbNullString
End If
```

4.3. Dynaaminen tagilaturi

Lähdetiedoston tulkitsemista varten on tärkeää pystyä vaihtamaan haluttuja tageja dynaamisesti. Dynaamisen tagilaturin tehtävä on tarkistaa tiedoston jäsentäjän löytämän tagin sijainti tagipuussa ja määrittää seuraavat sopivat tagit. Tätä varten dynaamisen tagilaturin tulee käydä läpi kaikki tagipuussa olevat tagit ja verrata tagin ominaisuuksia, kuten tasoa ja tyyppiä viimeksi löydettyyn tagiin. Mikäli yhtään tagia ei ole vielä tullut, toimitaan, kuten tullut tagi olisi sulkeva juuritagi, eli kuten kaikki tagit olisivat sulkeutuneet. Kuvassa 11 on esitetty yksinkertaistettuna kaaviona dynaamisen tagilaturin toiminta. Kaaviossa ei ole huomioitu kommentti- ja merkkijonotageja, joiden käyttöön liittyy muistitoimintoja.



Kuva 11. **Yksinkertaistettu kaavio dynaamisen tagilaturin toiminnasta**

4.4. Tulosten esittäminen

Toteutetun ohjelman ydin on yksinkertaistettuna vain paketoitu funktio. Tämä funktio palauttaa taulukkomuuttujan, joka sisältää tietoa löydetyistä tageista ja niiden väliin jäävistä sisällöistä. Tällöin ohjelman tulostiedon esitysmuotoa voi muuttaa joustavasti pienillä muutoksilla. Käyttöliittymän kautta käyttäjä voi valita halutuksi tulosten esitysmuodoksi joko Excel-työkirjan tai vaihtoehtoisesti HTML-tiedoston.

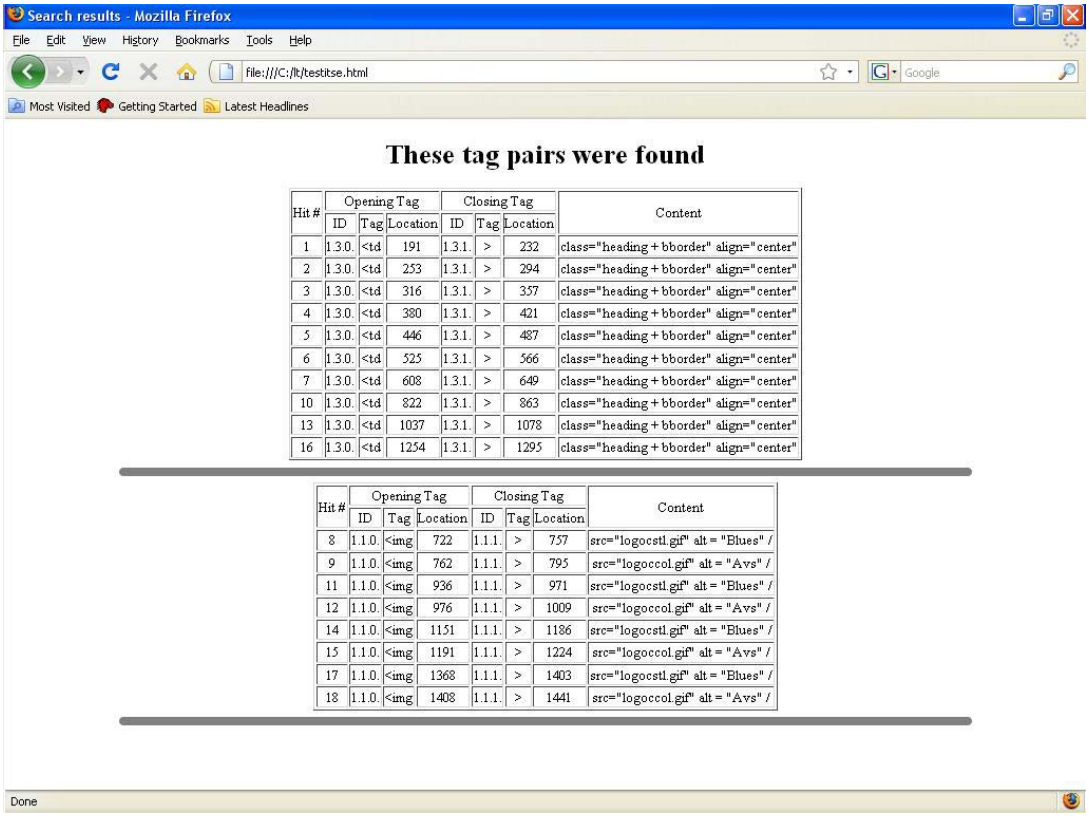
Mikäli käyttäjän valinta on Excel-työkirja, ohjelma luo uuden työkirjan. Työkirjaan luodaan uusi taulukko jokaista käyttäjän INI-tiedostossa määrittelemää avaavaa tagia kohti. Taulukkoa ei kuitenkaan tehdä, jos lähdetiedostosta ei löydy yhtään kyseistä avaavaa tagia. Taulukot nimetään avaavan tagin tagitunnisteen mukaan, millä vältetään myös erikoismerkkien tuomat mahdolliset ongelmat. Tulokset järjestetään taulukkoon osumanumeron mukaan. Osumanumero puolestaan määräytyy avaavan tagin löytymiskohdan mukaan. Osumanumerointi on yhteinen kaikille tagitunnisteille, eli se ei kerro tietyn tagitunnisteen osumamäärää. Kuvassa 12 on esimerkki löytyneiden tagien esitystavasta Excel-työkirjassa.

	A	B	C	D	E	F	G	
1	Osuma #	Aloittavan tagin tunniste	Aloittava tagi	Sisältö alkaa kohdasta	Päättävän tagin tunniste	Päättävä tagi	Sisältö päättyy kohtaan	Tagiparin sisältö
2	1	1.2.0.	<IMG	1700	1.2.1.	>	1772	style="FLOAT: left" alt=Avalanche src=
3	2	1.2.0.	<IMG	1797	1.2.1.	>	1876	style="FLOAT: right" alt=Avalanche src
4	3	1.2.0.	<IMG	4459	1.2.1.	>	4544	alt="Loading. Please Wait..." src="styl
5	4	1.2.0.	<IMG	4846	1.2.1.	>	4922	class=ipd title="Close Window" alt=X
6	5	1.2.0.	<IMG	5456	1.2.1.	>	5462	alt="
7	6	1.2.0.	<IMG	7131	1.2.1.	>	7250	title="Open Menu" alt=V src="http://f
8	7	1.2.0.	<IMG	9646	1.2.1.	>	9722	alt="Reply to this topic" src="style_im
9	8	1.2.0.	<IMG	9846	1.2.1.	>	9916	alt="Start new topic" src="style_image
10	9	1.2.0.	<IMG	10326	1.2.1.	>	10341	height=8 alt="
11	10	1.2.0.	<IMG	10761	1.2.1.	>	10880	title="Open Menu" alt=V src="http://f
12	11	1.2.0.	<IMG	11057	1.2.1.	>	11182	style="VERTICAL-ALIGN: middle" alt=V
13	12	1.2.0.	<IMG	11361	1.2.1.	>	11486	style="VERTICAL-ALIGN: middle" alt=V
14	13	1.2.0.	<IMG	11635	1.2.1.	>	11760	style="VERTICAL-ALIGN: middle" alt=V
15	14	1.2.0.	<IMG	11926	1.2.1.	>	12051	style="VERTICAL-ALIGN: middle" alt=V
16	15	1.2.0.	<IMG	12219	1.2.1.	>	12344	style="VERTICAL-ALIGN: middle" alt=V
17	16	1.2.0.	<IMG	12590	1.2.1.	>	12715	style="VERTICAL-ALIGN: middle" alt=V
18	17	1.2.0.	<IMG	12868	1.2.1.	>	12993	style="VERTICAL-ALIGN: middle" alt=V
19	18	1.2.0.	<IMG	13165	1.2.1.	>	13290	style="VERTICAL-ALIGN: middle" alt=V
20	19	1.2.0.	<IMG	15900	1.2.1.	>	15984	alt="" src="style_images/avalanche/f
21	20	1.2.0.	<IMG	16345	1.2.1.	>	16441	height=50 alt="" src="style_images/av
22	21	1.2.0.	<IMG	16502	1.2.1.	>	16605	style="VERTICAL-ALIGN: middle" alt="F
23	22	1.2.0.	<IMG	16651	1.2.1.	>	16791	id=pp-entry-gender-img-20049 style="
24	23	1.2.0.	<IMG	17290	1.2.1.	>	17381	style="PADDING-BOTTOM: 2px" alt=po
25	24	1.2.0.	<IMG	17886	1.2.1.	>	17984	height=90 alt="" src="http://nhluploa
26	25	1.2.0.	<IMG	18134	1.2.1.	>	18200	height=1 alt="" src="style_images/ava
27	26	1.2.0.	<IMG	18469	1.2.1.	>	18542	class=linked-sig-image src="http://i3
28	27	1.2.0.	<IMG	18747	1.2.1.	>	18826	alt="Go to the top of the page" src="s
29	28	1.2.0.	<IMG	19086	1.2.1.	>	19151	alt=+ src="style_images/avalanche/p
30	29	1.2.0.	<IMG	19310	1.2.1.	>	19377	alt="Quote Post" src="style_images/a
31	30	1.2.0.	<IMG	19851	1.2.1.	>	19935	alt="" src="style_images/avalanche/f
32	31	1.2.0.	<IMG	20297	1.2.1.	>	20403	height=50 alt="" src="http://nhluploa

1.2.0.

Kuva 12. Löytyneiden tagien esittäminen Excel-työkirjassa

Käyttäjän halutessa tulokset HTML-tiedostoon tehdään taulukoiden sijaan avaaville tageille HTML-tauluja. Kuvassa 13 on esimerkki HTML-tulostiedostosta.



Hit #	Opening Tag			Closing Tag			Content
	ID	Tag	Location	ID	Tag	Location	
1	1.3.0.	<td	191	1.3.1.	>	232	class="heading + bborder" align="center"
2	1.3.0.	<td	253	1.3.1.	>	294	class="heading + bborder" align="center"
3	1.3.0.	<td	316	1.3.1.	>	357	class="heading + bborder" align="center"
4	1.3.0.	<td	380	1.3.1.	>	421	class="heading + bborder" align="center"
5	1.3.0.	<td	446	1.3.1.	>	487	class="heading + bborder" align="center"
6	1.3.0.	<td	525	1.3.1.	>	566	class="heading + bborder" align="center"
7	1.3.0.	<td	608	1.3.1.	>	649	class="heading + bborder" align="center"
10	1.3.0.	<td	822	1.3.1.	>	863	class="heading + bborder" align="center"
13	1.3.0.	<td	1037	1.3.1.	>	1078	class="heading + bborder" align="center"
16	1.3.0.	<td	1254	1.3.1.	>	1295	class="heading + bborder" align="center"

Hit #	Opening Tag			Closing Tag			Content
	ID	Tag	Location	ID	Tag	Location	
8	1.1.0.	<img	722	1.1.1.	>	757	src="logocstl.gif" alt = "Blues" /
9	1.1.0.	<img	762	1.1.1.	>	795	src="logoccol.gif" alt = "Avs" /
11	1.1.0.	<img	936	1.1.1.	>	971	src="logocstl.gif" alt = "Blues" /
12	1.1.0.	<img	976	1.1.1.	>	1009	src="logoccol.gif" alt = "Avs" /
14	1.1.0.	<img	1151	1.1.1.	>	1186	src="logocstl.gif" alt = "Blues" /
15	1.1.0.	<img	1191	1.1.1.	>	1224	src="logoccol.gif" alt = "Avs" /
17	1.1.0.	<img	1368	1.1.1.	>	1403	src="logocstl.gif" alt = "Blues" /
18	1.1.0.	<img	1408	1.1.1.	>	1441	src="logoccol.gif" alt = "Avs" /

Kuva 13. Tuloksien esittäminen HTML-sivulla

4.5. Unicode-muuntajat

Ohjelman ydinosat on suunniteltu toimimaan myös erikoismerkkien kanssa. Tällaisia erikoismerkkejä ovat esimerkiksi rivinvaihdot ja sarkaimet. Jotta näitä pystyttäisiin luotettavasti käyttämään, niitä pitää jollakin tavalla muuntaa VBA:n helposti ymmärtämään muotoon. Tämä on toteutettu käyttämällä Unicodea, jonka avulla käyttöön saadaan kymmeniä tuhansia merkkejä. /10/

VBA:n käyttämät Unicode-funktiot palauttavat merkin normaalina kymmenjärjestelmän numerona. Tällöin ongelmaksi muodostuu merkin pituus, joka voi olla merkistä riippuen mitä vain yhden ja viiden välillä. Tämä on ohjelmassa korjattu muuttamalla numero heksadesimaaliluvuksi, jolloin yksikään merkki ei ole neljää heksadesimaalinumeroa pidempi. Lisäksi kaikki numerot muunnetaan neljän mittaiseksi lisäämällä sopiva määrä nollia tuloksen eteen. Lopuksi tuloksena tulevan heksadesimaalinumeron perään lisätään alaviiva, jolloin voidaan helpommin seurata yksittäisen merkin tilannetta. Näin esimerkiksi kirjain ”T” muuttuu muotoon ”0054_”.

Ohjelman käyttämästä Unicode-muodosta merkit muunnetaan vastaavalla prosessilla takaisin. Tällöin tulostiedostoihin saadaan sekä selkokieliset että Unicode-muodossa olevat tulokset.

4.6. Lähdetiedoston lukeminen

VBA:lla tekstitiedoston voi lukea yksinkertaisesti, mutta halutessa huomioida erikoismerkit suositellaan käyttämään FileSystemObject-objektia. FileSystemObjectilla päästään käsittelemään tietokoneen tiedostojärjestelmää. Avaamalla tekstitiedosto OpenTextFile -funktiota käyttäen voidaan tiedosto lukea esimerkiksi ReadAll-funktion avulla.

Tiedoston sisältö muutetaan Unicode-muotoon käymällä ReadAll-funktiolta saatu merkkijono merkki kerrallaan läpi. Tehokkuuden kannalta ei ole järkevää kirjoittaa merkkejä suoraan edellisen merkin perään, sillä aiempi merkkijono joudutaan kirjoittamaan aina uudestaan. Tästä johtuen kannattaa merkkijono halkaista pienempiin osiin ja lopuksi yhdistää nämä osat. Seuraavassa esimerkissä merkkijono ”OPINNÄYTETYÖ” käsitellään oletustavalla, lisäämällä seuraava merkki edellisen.

O; 1 merkki
OP; +2 merkkiä, yhteensä 3 merkkiä
OPI; +3 merkkiä, yhteensä 6 merkkiä
...
OPINNÄYTETYÖ; +12 merkkiä, yhteensä 78 merkkiä

Seuraavassa esimerkissä puretaan merkkijono ”OPINNÄYTETYÖ” kahden merkin pituisiin osiin.

Halkaistaan merkkipareiksi: OP-IN-NÄ-YT-ET-YÖ; 12 merkkiä
Yhdistetään merkkiparit:
OPIN; +4 merkkiä, yhteensä 16 merkkiä
OPINNÄ; +6 merkkiä, yhteensä 22 merkkiä
OPINNÄYT; +8 merkkiä, yhteensä 30 merkkiä
OPINNÄYTET; +10 merkkiä, yhteensä 40 merkkiä
OPINNÄYTETYÖ; +12 merkkiä, yhteensä 52 merkkiä

Halkaisemalla kahdentoista merkin pituinen merkkijono kahden merkin osiin säästetään siis kolmasosa merkkienkirjoituskerroista. Seuraavassa esimerkissä jälkimmäinen tapa on esitetty ohjelmakoodina:

```
If (l Mod 2 = 0) Or (l = Len(merkkijono)) Then
    vPalat(k) = uusijono
    k = k + 1
    uusijono = vbNullString
End If
```

Ohjelma on toteutettu pääasiallisesti tekstimuotoista tietoa ajatellen, mutta periaatteessa binäärimuotoista tietoa voi käsitellä aivan vastaavasti. Ohjelma toimii minkä tahansa lähteen kanssa, kunhan lähde voidaan lukea merkki kerrallaan.

4.7. Tietojen noutaminen Internetistä

Paikallisten tekstitiedostojen lukemisen lisäksi voidaan lukea myös normaaleita Internet-sivuja eli HTML-tiedostoja. Tarkalleen ottaen voidaan lukea mitä tahansa Internet-sivua, jonka lähdetietoon päästään käsiksi käyttämällä Internet Explorerin document-objektin body.outerHTML -metodia. Kyseinen metodi palauttaa sivun lähdetiedon merkkijonona, jota voidaan käsitellä vastaavasti kuin mitä tahansa tekstitiedostosta luettua tietoa. Seuraavassa esimerkissä Internet-sivun sisältö tallennetaan merkkijonoon.

```
'Varmistetaan, että IE on tyhjä ja sitten tehdään uusi IE objekti
Set ie = Nothing
Set ie = CreateObject("internetexplorer.application")

'Odotetaan sivun latautumista
Do
   If ie.ReadyState = 4 Then
      ie.Visible = False
      Exit Do
   Else
      DoEvents
   End If
Loop

Set objDoc = ie.Document

'Otetaan sivun sisältö talteen
strContent = objDoc.body.outerHTML

'Siivotaan jälkiä
Set objDoc = Nothing
ie.Quit
Set ie = Nothing
```

4.8. Testaaminen

Ohjelman testaaminen on suoritettu kahdessa vaiheessa. Ensimmäisessä vaiheessa jokainen ohjelman osa on testattu yksittäin ohjelman kehittämisen aikana. Tämän jälkeen testausta on suoritettu iteratiivisesti lisättäessä ohjelmaan uusia osia. Toisessa vaiheessa ohjelma on testattu käyttämällä mahdollisimman monimutkaisia, pelkästään testaamista varten rakennettuja lähdetiedostoja. Lisäksi ohjelmaa testattiin normaaleilla Internet-sivuilla. Ohjelma suoriutui testeistä hyvin.

Ohjelman testauksessa käytettiin paljon taulukkomuuttujien ja solualueiden välillä tapahtuvaa tarkastelua. Ohjelman tuloslaskentaa seurattiin kirjoittamalla taulukkomuuttujien

tiedot solualueisiin. Lopullisesta versiosta poistettiin kaikki viittaukset solualueisiin, joten tuloksena on hyvin pitkälti Excelistä riippumaton ohjelma.

4.9. Ohjelman käyttöohje

Ohjelman käyttäminen on varsin yksinkertaista. Ohjelman sisältävän työkirjan avaamisen jälkeen ohjelma käynnistyy automaattisesti. Ohjelman käynnistyttyä käyttäjän tulee vain syöttää lähde-, kohde- ja INI-tiedoston sijainnit. Lisäksi käyttäjä voi valita kielen. Käyttöliittymässä on myös tekstilaatikko ohjelman etenemisestä ilmoittamisen tiheyden säätämistä varten. Mikäli ohjelma suljetaan, sen saa uudelleen päälle painamalla Excel-työkirjassa olevaa painonäppäintä.

Mikäli halutaan kirjoittaa uusi INI-tiedosto, tämä voidaan tehdä ohjelman ulkopuolella kirjoittamalla normaali tekstitiedosto. Vaihtoehtoisesti voidaan avata Windowsin Muistio painamalla käyttöliittymässä painonäppäintä ”Avaa Muistio”. INI-tiedoston kirjoittaminen on tarkasti määritelty. Tätä käsitellään seuraavassa kappaleessa.

4.9.1. INI-tiedoston kirjoittaminen

Käyttäjän haluamat tagit kirjoitetaan INI-tiedostoon, jossa jokainen rivi vastaa yhtä tagia. Jokaisen rivin tulee olla yksilöllinen. Rivi jakautuu seuraavasti:

1. Tagityyppi. Tämä määritetään ensimmäisellä numerosarjaparilla, joka päättyy toiseen pisteeseen. Mikäli numerosarja on ”0.0.”, kyseessä on kommenttitagi. Mikäli numerosarja on ”0.1.”, kyseessä on merkkijonotagi. Kaikki muut numerosarjaparit ovat normaalitageja. Kommentti- ja merkkijonotageja ei voida sisällyttää itseensä moneen kertaan. Kommenttitagi voi kuitenkin olla merkkijonotagin sisällä ja päinvastoin. Nollalla ei saa aloittaa muita kuin kommentti- ja merkkijonotageja.

2. Tagin taso. Normaalitagin sijainti määritellään jonkun toisen tagin alle, mikäli kyseessä ei ole juuritagi. Juuritageissa pisteitä on yhteensä kolme. Kaikissa juuritagin alla olevissa tageissa on pisteitä kolme plus tagin taso, eli esimerkiksi kolmostasolla pisteitä on kuusi. Juuritagin alle tulevilla tagilla on sama alku kuin juuritageilla, eli esimerkiksi juuritagin ”1.1.0y” alle tagi voidaan sijoittaa antamalla sen tunnisteeksi ”1.1.1.0y”.

3. Tagin avaaminen ja sulkeminen. Tagitunnisteen viimeinen numerosarja määrittää tagin avaamisen ja sulkemisen. Jos numerosarja on ”0”, tagi on avaava. Jos taas numerosarja on jotakin muuta kuin ”0”, se on sulkeva.

4. Tagin tallentaminen. Riville numerosarjan jälkeen tulee kirjain ”y” tai ”n” määrittää, halutaanko tagin sisältö tallentaa vai ei. Jokaista haluttua avaavaa tagia kohti tulee olla myös vastaava haluttu sulkeva tagi.

5. Tagin tallentamisen määräävän kirjaimen jälkeen tulee tagierotin eli pystyviiva.

6. Lopuksi riville kirjoitetaan tagi itsessään. Tagi saa toistua INI-tiedostossa niin monta kertaa kuin halutaan. Tagi saa olla mikä tahansa kirjaimien, numeroiden ja merkkien yhdistelmä. Ohjelma tukee myös joitakin erikoismerkkitageja. Nämä kirjoitetaan lisäämällä tildet, eli ns. aaltoviivat merkin ympärille. Esimerkiksi jos halutaan kirjoittaa tagiksi
"a | b",
voidaan se kirjoittaa muodossa "a~horizontaltab~~verticalbar~~horizontaltab~b". Taulukossa 5 on listattu ohjelman tukemat erikoismerkit.

Taulukko 5. INI-tiedoston tukemat erikoismerkit

INI-tiedoston muoto	**Merkitys**
~horizontaltab~	Vaakasisennys
~linefeed~	Rivinsiirto
~carriagereturn~	Rivinpalautus
~verticaltab~	Pystysisennys
~backspace~	Askelpalautin
~verticalbar~	Pystyviiva

5. YHTEENVETO

Opinnäytetyön tulos on ohjelma, jonka avulla lähdetiedostosta saadaan luettua halutut tiedot. Ohjelma on helppo irrottaa Microsoft Excelistä, ja ohjelmasta on saatu nopeasti toimiva sekä luotettava.

Opinnäytetyötä tehdessä karttui kokemusta VBA:n mahdollisimman riippumattomasta käytöstä sekä Internet Explorerin ohjelmoinnista. Lisäksi esiselvityksen aikana VSTO tuli tutuksi.

Erityisesti mieleen jäi vaativa dynaamisen tagilaturin suunnittelu, sillä sen parissa kului useampi kuukausi. Myös syksyllä 2008 kirjoitettu opas Visual Basic Editorista ja lähdetiedon hankkimisesta sitä varten oli mieleenpainuva elämys.

Opinnäytetyön tekeminen itselle ilman minkäänlaisia ulkopuolisia paineita on todellinen kaksiteräinen miekka. Toisaalta se tuo rajattomasti vapautta, toisaalta se ajaa tekijän jahkailemaan opinnäytetyön tekemisen kanssa, koska kukaan ei ole asettamassa minkäänlaisia rajoja.

Lähdemateriaali oli käytännössä kokonaan englanninkielistä, mikä teki kirjoittamisesta paikoitellen haastavaa. Alalla käytetään toisinaan suomennettuja termejä ja toisia ei käännetä koskaan. Myös lyhenteiden kanssa työskentelyssä oli omat erikoisuutensa.

6. LÄHDELUETTELO

/1/ Cooper, Rob, Tucker, Michael, Expert Access 2007 Programming, 1. painos, Wiley, 2008.

/2/ Function X, The Types of Values of a Spreadsheet, [WWW-dokumentti], <http://www.functionx.com/vbaexcel/Lesson02.htm> 12.5.2009.

/3/ Green, John, Bullen, Stephen, Bovey, Rob, Alexander, Michael, Excel 2007 VBA Programmer's Reference, 1. painos, Wiley, 2007.

/4/ Groh, Michael R., Stockman, Joseph C., Powell, Gavin (Author), Prague, Cary N., Irwin, Michael R., Reardon, Jennifer, Access 2007 Bible, 1. painos, Wiley, 2007.

/5/ Gruener, Wolfgang, First Look CorelDraw X4: Still the one to beat, [WWW-dokumentti], <http://www.tgdaily.com/index2.php?option=com_content&do_pdf=1&id=35905> 7.5.2009.

/6/ Heaton, Jeff, Visual Basic Compared to Java, [WWW-dokumentti], <http://www.jeffheaton.com/java/vbjava.shtml> 8.5.2009.

/7/ Idehen, Kingsley Uyi, Microsoft Data Access API Backgrounder: ODBC, [WWW-dokumentti], <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com's%20BLOG%20%5B127%5D/1106> 3.12.2008.

/8/ Martin, Robert, Puls, Ken, Hennig, Teresa, RibbonX: Customizing The Office 2007 Ribbon, 1. painos, Wiley, 2008.

/9/ Microsoft Developer Network, Ribbon Extensibility Overview, [WWW-dokumentti], <http://msdn.microsoft.com/en-us/library/aa942866(VS.80).aspx> 8.5.2009.

/10/ Salminen, Riku, Hietaniemi, Jarkko: Mikä Unicode on?, [WWW-dokumentti], <http://www.unicode.org/standard/translations/finnish.html)> 29.4.2008.

/11/ Tyson, Herb, Word 2007 Bible, 1. painos, Wiley, 2007.

/12/ Walkenbach, John, Excel 2007 Formulas, 1. painos, Wiley, 2007.

/13/ Walkenbach, John, Excel 2007 Power Programming with VBA, 1. painos, Wiley, 2007.

/14/ Whitney, Justin, Nine Compelling Reasons to Move From VBA to VSTO 2005, [WWW-dokumentti], <http://www.devx.com/OfficeProDev/Article/28088> 7.5.2009.