TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Tarmo Lehtonen

# TIEDONTALLENNUS MOBIILILAITTEISSA

Työn ohjaaja: Tony Torp

Tampere 2007

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Tarmo Lehtonen
Tiedontallennus mobiililaitteissa
Tutkintotyö 48 sivua
Työn ohjaaja Tony Torp
Työn teettäjä TAMK
Maaliskuu 2007
Hakusanat Tiedontallennus, Symbian, Windows Mobile, Mobile Linux

## TIIVISTELMÄ

Tutkintotyössä tutkitaan tiedontallennuksen toteutusta erilaisissa mobiililaitejärjestelmissä.

Suomen kielellä on niukasti materiaalia, jota voidaan käyttää mobiiliohjelmoinnin opettamisessa. Tarkoituksena on ottaa selville miten tiedontallennus on toteutettu mobiililaitteissa ja saada aikaiseksi pohjamateriaalia, jota voidaan käyttää opetuksen kehittämiseen.

Tutkintotyötä tehtäessä on tutkittu kirjoja Symbianista ja Windows Mobilesta, haettu tietoja Internetin ohjelmointisivustoilta ja käytetty hyväksi omaa ohjelmointikokemusta.

Tutkintotyössä kerrotaan, kuinka tiedostoihin tallentaminen tapahtuu kussakin käsitellyssä järjestelmässä. Painopisteenä on Symbian-käyttöjärjestelmä.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering
Tarmo Lehtonen
Data Saving in Mobile Devices
Engineering Thesis 48 pages
Thesis Supervisor Tony Torp
Commissioner Tampere Polytechnic
March 2007
Keywords Data Saving, Symbian, Windows Mobile, Mobile Linux

## ABSTRACT

Mobile systems are a relatively new very quickly growing sector of the ITC industry. At the moment there is little material in Finnish to use in educating people. The goal of this study is to find out how data saving is done in three different mobile systems, Symbian OS, Windows Mobile and Mobile Linux, and to provide base material for developing the education of mobile programming.

## SISÄLLYSLUETTELO

# 1 JOHDANTO

Mobiililaitteiden eli matkapuhelimien, kämmentietokoneiden ja muiden mukana liikkuvien laitteiden käyttö lisääntyy jatkuvasti. Matkapuhelinten käyttäjien määrä maailmassa on nykyään miljardeja ja määrä kasvaa nopeasti. Uusia laitteita julkaistaan vuosittain suuri määrä. Matkapuhelinten tekniikka on vielä hyvin nuorta ja sitä kehitetään jatkuvasti. Uusi tekniikka dokumentoidaan usein pelkästään englannin kielellä ja suomeksi on vaikea löytää opintomateriaaleja.

Tämän tutkintotyön tarkoituksena on tuottaa opintomateriaalia suomen kielellä ja tutkia millä tavalla tiedontallennus tapahtuu mobiililaitteissa.

Tutkittaviksi mobiiliympäristöiksi valittiin Symbian OS, Mobile Linux ja Microsoft Windows Mobile. Nämä ympäristöt ovat kolme tunnetuinta. Suurin osa maailman älypuhelimista käyttää tällä hetkellä Symbian-käyttöjärjestelmää. Yhä useammat laitevalmistajat suuntaavat katseensa avoimen lähdekoodin maailmaan ja Linuxiin entistä parempien katteiden toivossa ja Linux-käyttöjärjestelmän osuus maailman mobiililaitemarkkinoilla on kasvamassa huomattavasti. Microsoft toisaalta on maailman suurimpia ja tunnetuimpia ohjelmistoyrityksiä ja sen tuotteilla on huomattava markkina-asema lähes kaikilla tietotekniikan osa-aluilla. Microsoft onkin panostanut huomattavia summia mobiilikehitykseen ja tavoittelee merkittävää asemaa myös tällä alalla.

Edellä mainituista ympäristöistä keskityn eniten Symbianiin ja sen tapaan tallentaa tietoa. Muita järjestelmiä käyn lävitse pintapuolisemmin. Aluksi esittelen kaikki järjestelmät ja tutkin, minkälaisia tekniikoita Symbian-älypuhelin käyttää. Myöhemmin tarkastelen Symbianin erilaisia tapoja tallentaa tietoa tiedostoihin ja tietovarastoihin. Sen jälkeen esittelen Linuxin tapaa hoitaa tiedontallennus ja käsittelen Windows Mobilen `system.IO` -luokat, jotka vastaavat tiedostojen ja tietovirtojen käsittelystä siinä ympäristössä. Analysoin järjestelmien yhtäläisyyksiä, eroja ja käytettävyyttä ja lopuksi teen yhteenvedon kaikesta.

# 2 KÄYTETYT LYHENTEET JA NIIDEN SELITYKSET

Taulukossa 1 on tutkintotyössä käytetyt lyhenteet ja niiden selitykset.

*Taulukko 1 Käytetyt lyhenteet*

| **Lyhenne** | **Selitys** |
|---|---|
| PDA | Personal Digital Assistant eli kämmentietokone |
| .NET | Microsoftin tarjoama kehysympäristö Windows käyttöjärjestelmään |
| SDK | Software Development Kit eli sovelluskehityspakkaus |
| BP | Baseband processor |
| AP | Aplication Processor |
| IPC | Inter Processor Communication |
| MMU | Memory Management Unitia |
| TLB | Translation Look-aside Buffer |
| ICache | Käskyvälimuisti |
| DCache | Datavälimuisti |
| LFFS | Log Flash File System |
| FTL | Flash Translation Layerin |
| ECC | Correction Code |
| DMA | Direct Memory Access |

# 3 TIEDONTALLENNUKSESTA YLEISESTI

## 3.1 Käsiteltävät mobiiliympäristöt

Tässä osiossa esittelen lyhyesti käsiteltävät käyttöjärjestelmät ja niiden tiedontallennukseen liittyvät toiminnallisuudet. Myöhemmissä osioissa käsittelen kutakin käyttöjärjestelmää vielä tarkemmin ja syvällisemmin.

### 3.1.1 Symbian OS

Symbian OS on pienitehoisille ja vähillä resursseilla varustetuille laitteille tarkoitettu käyttöjärjestelmä. Se perustuu alkujaan Psionin PDA-laitteita varten kehittämään EPOC-käyttöjärjestelmään. Symbianista löytyy useita ohjelmistoalustoja, mm. Series 60 (S60), Series 80 (S80), Series 90 (S90) ja UIQ.

Käsittelen myöhemmässä osiossa Symbianin tiedostojen ja datan hallintaa. Symbian sisältää abstraktiot tiedostojen, tietovirtojen ja säiliöiden käyttöön. Tietovirtoja eli streameja ei käytetä pelkästään tiedostoja käytettäessä vaan myös muissa yhteyksissä. Näin on siksi, että ne tarjoavat entistä helpommat syntaksit ja niitä voidaan käyttää uudelleen datan muokkauksessa olioiden yhteydessä. Säiliöitä käytetään täydentämään tietovirtoja säilömällä ja hakemalla ulkoistettua tietoa.

### 3.1.2 Mobile Linux

Linux on avoimeen lähdekoodiin perustuva käyttöjärjestelmä. Linuxista löytyy lukuisia erilaisia versioita. Mobile Linux -käsitteellä tarkoitetaan mobiililaitteisiin sulautettua Linuxia.

Linuxista löytyy myös mobiililaitteissa useita eri versioita eri laitteista. Monet valmistajat käyttävät sitä pohjanaan ja hiljattain onkin perustettu Linux Mobile Foundation. Tuon säätiön jäseninä ovat mm. Motorola, NEC, Samsung ja Vodafone ja muut suuret yritykset. Säätiön tarkoituksena on kehittää Foundation Platform eli Linuxiin pohjautuva avoin mobiililaitteisiin tarkoitettu ohjelmistoalusta.

Nokian 770 ja N800 Internet Tabletit pohjautuvat myös avoimeen lähdekoodiin ja Linuxiin. Niiden kehitysympäristönä on Maemo.

### 3.1.3 Microsoft Windows Mobile

Microsoft Mobile on Microsoftin kehittämä sovellusalusta Pocket PC:lle ja Smartphone-älypuhelimille. Sovellusalustalla tarkoitetaan tässä yhteydessä integroitua tietojenkäsittely-ympäristöä joka koostuu käyttöjärjestelmästä, .NET-ajoaikaympäristöstä, tietyistä sovelluksista ja näihin liittyvistä sovelluskehitystyökaluista.

Microsoft Windows Mobilen uusin versio on Windows Mobile 6.0, joka on juuri ilmestynyt. Versio 5.0, jota käsittelen, perustuu Windows CE .NET 5.0 -käyttöjärjestelmään. Microsoft tarjoaa itsenäisille sovelluskehittäjille työkaluksi Smartphone 2005 SDK:n (Software Development Kit eli sovelluskehityspakkaus). Tätä SDK:ta voidaan käyttää mm. Visual Studio 2005:n kanssa.

## 3.2 Tiedontallennukseen liittyvä laitteisto Symbian-puhelimessa

Jotta voitaisiin täysin ymmärtää tiedontallennukseen mobiililaitteissa liittyvät ohjelmistotekniset haasteet, on tunnettava myös asiaan liittyvä laitteisto periaatteellisella tasolla. Tässä osiossa käsittelen tyypillisen Symbian-puhelimen sisältä löytyvää laitteistoa tiedontallennuksen näkökulmasta.

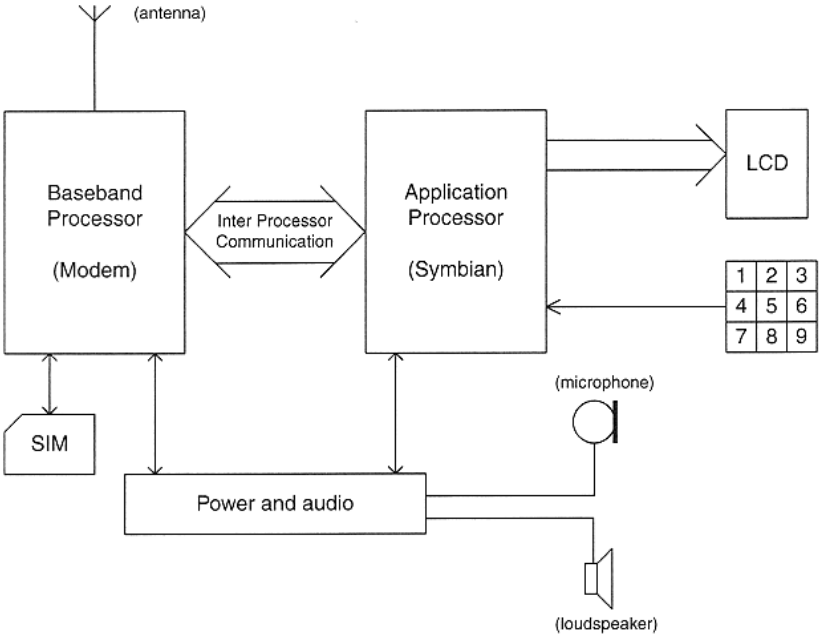
Symbian-matkapuhelimet on suunniteltu ensisijaisesti toimimaan puhelimina. Niiden päätehtävänä on soittaa puheluita ja niiden akun keston tulee olla pitkä. Vasta toissijaisena sovelluksena tulee niiden mahdollisuus toimia ohjelmistoalustana erilaisille sovelluksille. Tämä asettaa tiettyjä rajoituksia sekä laitesuunnittelijoille että ohjelmistosuunnittelijoille. /1/

Yksi EPOC32:n kantavista ajatuksista on myös se, ettei käyttäjän data saisi kadota missään tilanteessa, vaikka laite kaatuisi tai tapahtuisi muuta odottamatonta. Tiedostojärjestelmä on tarkoituksella tehty robustiksi eli kestäväksi.

Käytetyt tekniikat ovat pitkälti samankaltaisia kuin PC-maailmasta tutut tekniikat, sillä erotuksella, että ne ovat paljon kompaktimpia ja pienempitehoisia. Linux- ja Windows-pohjaiset puhelimet pohjautuvat pitkälti samaan tekniikkaan, niissä käytetään vain eri käyttöjärjestelmää.

### 3.2.1 Symbian-puhelimen rakenne

Symbian-puhelin koostuu kahdesta toisiaan tukevasta toimialueesta: Baseband Processor (BP) hoitaa radioliikenteen ja siihen liittyvän toiminnallisuuden ja Aplication Processor (AP) hoitaa sovelluksiin liittyvän laskennan ja korkeamman tason koodin eli käyttöliittymän käyttäjälle. Näiden ympärillä toimivat puhelimen muut osat, kuten akku, näyttö, kaiuttimet ja muut puhelimeen liitetyt laitteet. Kuvassa 1 nähdään tällainen rakenne.




*Kuva 1 Tyypillinen kahden sirun ratkaisu /1/*

Prosessorien välillä toimii nopea Inter Processor Communication -linkki (IPC). Tiedontallennuksen kannalta näistä kahdesta prosessorista AP on olennaisempi. Se toimii kahdessa eri moodissa: täydellä teholla, kun puhelinta käytetään ja toimettomana virransäästötilassa, kun puhelinta ei käytetä. Sitä voidaan sanoa Symbian-puhelimen sydämeksi. Se sisältää ARM CPU:n, muistia, näyttö- ja audioliitännät, multimediakiihdyttimiä ja muita toiminnallisuuksia ja oheislaitteita. Tarkastellaan näistä muutamia tiedontallennukseen olennaisesti liittyviä kokonaisuuksia. /1/

### 3.2.2 Fyysinen muistikartta

Symbian-puhelimet käyttävät 32-bittistä muistiavaruutta. Näin ollen puhelimella on periaatteessa käytössä 4 GB suoraan osoitettavaa muistia. Puhelin siis näkee 4 GB, mutta sitä ei ole niin paljoa käytössä. Symbian OS käyttää Memory Management Unitia (MMU) muuntaakseen fyysiset muistiosoitteet ohjelmistoille soveltuvaksi virtuaalimuistiavaruudeksi. Vaikka 32-bittisyys mahdollistaa 4Gt:n käytön, käytännössä muisti on jaettu pienempiin osiin. /1/

### 3.2.3 Memory Management Unit (MMU)

Symbian OS tarvitsee myös MMUn koordinoimaan ja valvomaan virtuaalimuistin käyttöä avoimen käyttöjärjestelmän puolella. MMU sijaitsee prosessorin ja systeemiväylän välissä ja kääntää virtuaalimuistiosoitteet prosessorin ymmärtämiksi fyysisiksi osoitteiksi. Tämä tehdään joka kerta, kun muistia käsitellään.

MMU järjestää muistin sivuihin. Yleensä nämä sivut ovat pieniä, 4 kt:n kokoisia, mutta suurempia 64 kt:n ja 1 Mt:n osioita esiintyy.

Joka kerta kun muistia luetaan, MMU hakee virtuaalimuistin kautta fyysisestä muistista oikean väyläosoitteen ja tarkistaa sivun käyttöoikeudet. Nopeuttaakseen tätä toimenpidettä MMU käyttää Translation Look-aside Bufferia (TLB) tallentaakseen tulokset välimuistiin. Jos välimuistissa ei ole haettua muistiosoitetta, MMU tekee uuden virtuaalisen haun ja tallentaa tuloksen TLB:hen. /1/

### 3.2.4 Välimuisti

Välimuisteja käytetään prosessoreissa, jotta ne voisivat saavuttaa optimaalisen käyttönopeuden. Näin on siksi, että prosessori toimii huomattavasti nopeammin kuin siihen liitetty muisti. Välimuisti pitää paikallisia kopioita viimeksi muistista haetuista tiedoista, joita prosessori voi käyttää välittömästi, sen sijaan, että hakisi hitaammasta muistista saman tiedon ja odottaisi sitä. Prosessorin kannalta sillä, missä tarvittava tieto sijaitsee kullakin ajanhetkellä, ei ole merkitystä. Prosessorin yhteydessä olevan välimuistin käyttö on useita kertaluokkia nopeampaa kuin keskusmuisti.

Välimuisti käyttää hyväkseen sitä, että ajettava ohjelmakoodi on usein hyvin pitkälti itseään toistavaa. Silmukassa oleva koodi käyttää samoja käskyjä ja osoitteita useita kertoja. Kun prosessori tarvitsee jonkin tiedon muistista, välimuisti tarkistaa, onko siinä tuota muistiosoitetta vastaavaa riviä. Jos on, tuo tieto palautetaan prosessorille välittömästi, jos ei, haetaan muistista tuo vaadittu tieto prosessorille ja lisätään

muistiosoite välimuistiin. Samalla poistetaan tarvittaessa jokin vanha tieto välimuistista. Laitteen käynnistyessä muisti on tyhjä eikä sieltä tarvitse poistaa mitään.

Symbian-puhelimien prosessoreissa on käytössä kaksi erillistä välimuistia, käskyvälimuisti (ICache) ja datavälimuisti (DCache). /1/

### 3.2.5 Keskusmuisti

Keskusmuistiin tallennetaan kaikki kulloinkin käytössä oleva data ja usein myös ajettava koodi. Keskusmuistin määrä ratkaisee, minkätyyppisiä ja miten isoja sovelluksia voidaan käyttää samanaikaisesti, ja keskusmuistin hakunopeus vaikuttaa niiden suorituskykyyn. Symbian-puhelimessa on yleensä 8 – 64 Mt keskusmuistia, mutta uusimmissa puhelimissa, kuten Nokian N95:ssä, on muistia jopa 160 Mt. Käyttöjärjestelmä ei itsessään vaadi paljoa muistia, joten tarvittava määrä määräytyy puhelimen käyttötarkoitusten mukaan. Esimerkiksi kamera ja muut multimediasovellukset tarvitsevat käyttöönsä suuria määriä muistia. Myös käytössä olevan flash-muistin tyyppi vaikuttaa muistin tarpeeseen, sillä NAND-tyyppistä flash-muistia käytettäessä joudutaan kopioimaan megatavuittain ohjelmakoodia keskusmuistiin ajettavaksi.

Keskusmuistin vaikutus puhelimen kokonaishintaan ja energian kulutukseen on merkittävä, sillä prosessorin lisäksi myös väylään liitetyt oheislaitteet käyttävät keskusmuistia. /1/

### 3.2.6 Flash-muisti

Symbian-puhelimissa tietoa säilytetään tavallisesti flash-muistissa. Flash-muisti on piipohjainen katoamaton muisti, jota voidaan ohjelmoida ja tyhjentää elektronisesti. Flash-muistin erityispiirteenä on se, että yksittäisiä bittejä voi muuttaa vain arvosta

yksi arvoon nolla. Bitin arvon palauttaminen takaisin arvoon yksi vaatii kokonaisen muistin osion, tyypillisesti 64 Kt:n, uudelleen kirjoittamisen.

Flash-muisteja on kahden tyyppisiä: NOR- ja NAND-tyyppisiä. Nimet viittaavat niiden sisäiseen rakenteeseen. Symbian-puhelimissa on käytössä molempia tyyppejä, tosin erilaisilla tiedostojärjestelmillä. NOR-flash käyttää LFFS (Log Flash File System) -järjestelmää ja NAND-tyypillä käytössä on standardi FAT-tiedostojärjestelmä Flash Translation Layerin (FTL) päällä. NORia voidaan ajaa suoraan paikaltaan, NANDi täytyy kopioida keskusmuistiin.

Tyypillisessä Symbian-puhelimessa on käytössä 32 - 64 Mt flash-muistia, joskin uusissa malleissa multimedia ominaisuuksien lisääntymisen ja 3G:n tulemisen myötä lisämuistien määrää lasketaan jo gigatavuissa. /1/

#### 3.2.6.1 NOR Flash

NOR-tyyppistä flash-muistia voidaan käyttää samoin kuin tavallista RAM-muistia. Siihen tallennettua tietoa voidaan lukea satunnaisessa järjestyksessä ja järjestelmä voi myös käynnistyä suoraan siltä. Käyttäjädatan tallennukseen Symbian käyttää NOR-tyypin päällä LFFS:ää. Tämä järjestelmä on optimoitu käyttämään hyväkseen NOR-tyypin erityispiirteitä.

NOR-tyyppi sallii rajoittamattoman määrän kirjoitusoperaatiota samaan data lohkoon, kun muutetaan bittejä yhdestä nollaan. Flash-muisteilla on yleensä kirjoituspuskuri, jonka ansiosta useita bittejä voidaan kirjoittaa samanaikaisesti. Puskuroituun kirjoitusoperaatioon kuluu aikaa 50 - 600µs kirjoitettavan datan mukaan. Data, jossa viereiset bitit ovat samoja, tallentuu nopeammin kuin data, jossa ne vaihtelevat. NOR-lohkojen pyyhkiminen on hidasta, 0,5 - 1 sekuntia, mutta se voidaan keskeyttää ja myöhemmin sitä voidaan jatkaa. LFFS käyttääkin tätä ominaisuutta hyväkseen ja siivoaa poistettuja tietoja pois taka-alalla, kun muita tärkeämpiä operaatioita ei ole käynnissä. /1/

#### 3.2.6.2 NAND Flash

NAND flashia käsitellään lohkopohjaisena levynä suorasaantisen muistin sijaan. Toisin kuin NOR-tyypillä sillä ei ole osoitelinjoja, joten se ei näy suoraan muistikartalla. Tämä ero tarkoittaa käytännössä sitä, että NAND-muistista ei voi suoraan ajaa ohjelmakoodia, vaan se on ensin kopioitava keskusmuistiin. Siksi NAND-tyyppiä käytettäessä tarvitaan enemmän keskusmuistia verrattuna vastaavaan NOR flashia käyttävään puhelimeen. NAND-muistin kirjoitusoperaatiot ovat noin kymmenen kertaa nopeampia kuin NOR-muistia käytettäessä.

NAND flash yhdistyy prosessoriin ja käyttää erityistä liitäntää 8- tai 16-bittisen väylän kautta. Tuo liitäntä käyttää DMA:ta datasiirtoihin ja sisältää ECC:n laskemiseen vaadittavat piirit. Liitäntä kirjoittaa ja lukee NAND flashia käyttäen datasivuja.

Puhelin ei voi suoraan käynnistyä NAND-muistista. Käytettäessä sitä käynnistysprosessi on monimutkaisempi. NAND on myös luonteeltaan vähemmän luotettava kuin NOR. Bittivirheet ovat mahdollisia eri muistioperaatioiden yhteydessä. Tätä ongelmaa voidaan helpottaa laskemalla Error Correction Code (ECC) eli virheenkorjauskoodi jokaiselle datasivulle. Jokaisen lukuoperaation yhteydessä lasketaan ECC uudelleen ja verrataan aikaisempaan. Tällä tekniikalla saadaan korjattua yksittäisiä bittivirheitä, mutta usean bitin virheitä ei. Jos usean bitin virheitä löytyy, pidetään tuota kyseistä datasivua korruptoituneena, viallisena. /1/

### 3.2.7 Direct Memory Access (DMA)

Symbian käyttää DMA:ta siirtämään oheislaitteiden vaatimat datasiirrot pois prosessorin vastuulta, jotta prosessori voi suorittaa muita tehtäviä. DMA voi vähentää prosessorille tulevien keskeytyksien määrää jopa sadasosaan jollain tietyllä aikavälillä. Tämä säästää virtaa ja parantaa puhelimen käyttöliittymän reaaliaikaisuutta.

DMA on väylän toimintaa säätelevä oheislaite. Se voi siirtää suuria määriä dataa eri oheislaitteiden välillä ilman prosessorin käyttöä. /1/

## 3.3 Tiedontallennuksen tarpeet

Mobiililaitteiden kehitys on kulkenut pitkään kohti entistä visuaalisempaa ilmettä. Multimediasovellusten määrä, koko ja laitteistolle asetetut vaatimukset ovat jatkuvasti lisääntyneet. Tämä tarkoittaa myös tiedontallennuksen tarpeen lisääntymistä. Laitteiden tarjoama muistikapasiteetti on sisäisen muistin ja fyysiseltä kooltaan pienien, mutta tallennuskapasiteetiltaan kasvavien muistikorttien ansiosta kasvanut merkittävästi ja tämä trendi näyttäisi jatkuvan vielä.

Toisaalta laitteet hakevat verkosta yhä enemmän sisältöä, mikä vähentää datan tallentamisen tarvetta, mutta toisaalta laitteisiin halutaan tallentaa yhä enemmän dataa. Itse tuotetun datan määrä kasvaa laitteiden lisäominaisuuksien kasvaessa ja hienostuessa. Yhä tarkemmalla resoluutiolla varustetut kamerat ja videokuva vaativat yhä enemmän muistikapasiteettia. Lisäksi valmis materiaali ohjelmistoissa, kuten GPS-navigaatio-ohjelman kartat, vievät yhä enemmän ja enemmän resursseja.

Tämä on käyttäjän ja ohjelmoijan kannalta hyvä asia. Vaikka ohjelmoijan on suunniteltava sovelluksensa yhtä huolellisesti kuin ennenkin, muistikapasiteetin lisääntyminen tarjoaa monia mahdollisuuksia rakentaa entistä suurempia, sisällöltään rikkaampia ja laadukkaampia sovelluksia. Sovelluksen kehittäjä hyötyy voidessaan kehittää uudenlaisia sovelluksia ja käyttäjä hyötyy saadessaan käyttöön entistä hyödyllisempiä sovelluksia.

# 4 SYMBIAN OS

Kuten aikaisemmin kerroin, käsittelen tässä osiossa Symbianin tiedostojen ja datan hallintaa ja keskityn tiedostojen, tietovirtojen ja storejen käyttöön. Aluksi tarkastelen kutakin tiedon tallennuksen tapaa erikseen ja lopuksi esittelen esimerkin tiedoston tallentamisesta streamia käyttäen yksinkertaisessa sovelluksessa.

## 4.1 Tiedostojen käsittely ja File Server

### 4.1.1 Sovelluksen käyttämien tiedostojen sijoittaminen

Käyttäjän näkökulmasta on vain kahdenlaisia tiedostoja: käyttäjän omaa dataa sisältäviä tiedostoja ja järjestelmätiedostoja. Symbianin versiossa 8.0 ja edeltävissä versioissa nuo järjestelmätiedostot sijaitsevat tyypillisesti hakemistossa `\System\` kullakin levyasemalla. Myös tiettyjen ohjelmien, kuten sähköpostin, käyttämien tiedostojen oli hyvä sijaita tuossa hakemistossa. Sellaiset tiedostot, joita käyttäjä voi avata niin halutessaan, oli hyvä sijoittaa muihin hakemistoihin. Sovellusten oli kuitenkin aina määriteltävä, missä niiden käyttämiä datatiedostoja säilytetään.

Symbian 3rd editioniin on tuotu Platform Security eli kehitysalustan turvallisuus malli. Se sisältää uusia turvallisuus ominaisuuksia, jotka ovat muuttaneet tiedostojen sijoittamista ja niiden käsittelyä. Symbian 3rd editionissa on käytössä Data caging ominaisuus, joka tarkoittaa sitä, että sovelluksilla ja käyttäjillä on pääsy vain tietyille aluille tiedostojärjestelmässä. Käytännössä sovelluksilla on pääsy vain omiin yksityisiin eli privaattikansioihin ja sellaisiin kansioihin, jotka on jätetty avoimiksi. Sovellus ei siis voi lukea toisen sovelluksen privaattikansioita.

Taulukossa 2 kerrotaan hakemistoihin ja dataan liittyvät rajoitukset.

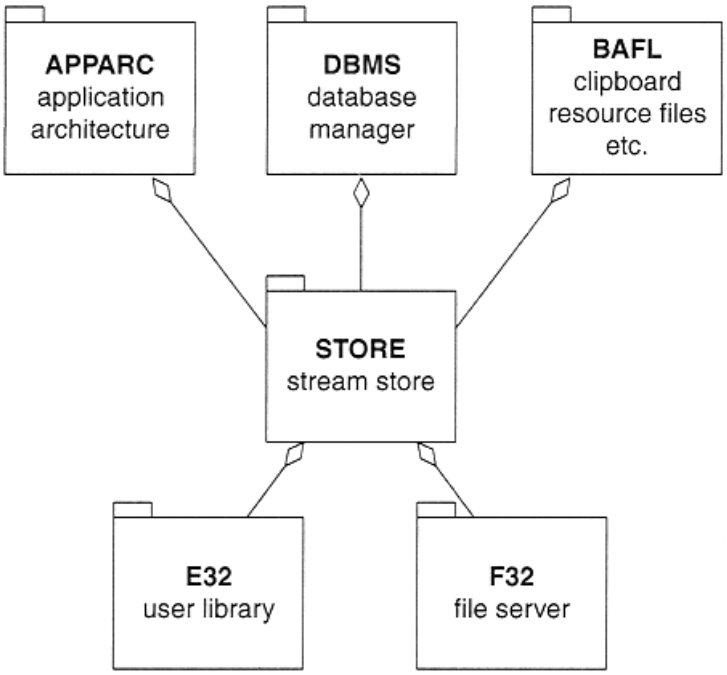
*Taulukko 2 Hakemistot ja niiden sisältämän datan luvun rajoitukset*

| **Kansio** | **Käyttö** |
|---|---|
| /sys/ | Rajoitettu järjestelmä alue, johon on pääsy ainoastaan Trusted Computing Base (TCB)-ominaisuuden omaavilla moduuleilla. Ajettavat tiedostot sijoitetaan kansioon /sys/bin/, joka on ainoa paikka, josta niitä voidaan ajaa. |
| /private/ | Sovelluksien privaattidata on sijoitettu kansioon /private/<SID>/. |
| /resource/ | Sovellusten julkinen data kuten ikonit ja bittikartat sijoitetaan tähän kansioon. Data on kuitenkin read-only tyyppistä, jos sovelluksella ei ole TCB –ominaisuutta. |

Muu data laitteessa, kuten kuvat, musiikki ja dokumentit, ovat käyttäjän ja sovellusten käytettävissä normaalisti. /7/

### 4.1.2 Tiedostoihin ja datansiirtoon liittyvät APIt eli sovellusliittymät

Kuvassa 2 on esitelty tiedostoihin ja datansiirtoon liittyvät APIt ja niiden suhteet toisiinsa.




*Kuva 2 Tiedostoihin ja datansiirtoon liittyvät APIt /1/*

*File server* eli tiedostopalvelin tarjoaa perustiedostopalvelut, kuten levyasemat, hakemistot, tiedostot ja asennettavat tiedostojärjestelmät käyttöön korkeamman tason komponenteille. Se onkin olennainen osa Symbian-käyttöjärjestelmän pohjaa.

*Stream store* on keskeinen osa datan käsittelyn osalta. Oliot käyttävät datan vaihtoon keskusmuistin ja tietovirtojen välillä <<- ja >>-operaattoreita sekä `ExternalizeL()`- ja `InternalizeL()`-funktioita. Storet ovat oikeastaan vain tietovirtojen kokoelmia. Symbianin natiivit tiedosto- ja dokumenttiformaatit käyttävät tiedosto-storeja.

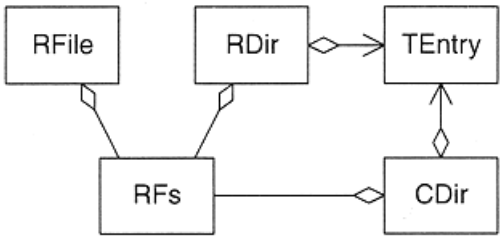
*Clipboard*- eli leikepöytä-API käyttää stream store-teknologiaa ja yhtä tiedostoa, `clipbrd.dat`, joka on jaettu kaikkien sovellusten kesken. Kopiointiohjelma voi näin ollen asettaa dataa leikepöydälle useassa eri formaatissa ja liittämisohjelma voi valita sen tarkoituksiin parhaiten soveltuvan formaatin.

## 4.2 File Server

File server (F32) eli tiedostopalvelin on tärkeä osa Symbianin tiedontallennusta ja käsittelyä. Se tarjoaa client APIn, joka sallii käytettävien ohjelmien käyttää levyasemia, hakemistoja ja tiedostoja sekä lukea ja kirjoittaa dataa tiedostoihin. F32 käyttää samankaltaista järjestelmää kuin DOS: se tarjoaa 26 eri mahdollista asemaa nimettyinä A: … Z:, täysin hierarkkisen hakemistorakenteen ja se sallii pitkien tiedostonimien käyttämisen, ja sallii myös lähes minkä tahansa kirjoitusmerkin käytön, lukuun ottamatta tietysti järjestelmän varaamia merkkejä. Hakemistojen nimet on erotettu käyttäen \-merkkiä, kuten Windowsissa. Pistettä voidaan käyttää merkitsemään tiedostopäätettä, vaikka Symbianin kannalta sillä ei ole mitään erikoismerkitystä. Tiedostonimi mukaan lukien sen täysi hakemistopolku voi olla 256 merkkiä pitkä.

Windowsin tapaan Symbianin tiedostopalvelin säilyttää tiedoston nimessä käytetyt isot kirjaimet, mutta ei ota niitä huomioon. Tämä tarkoittaa sitä, että samannimisiä, toisistaan vain isojen kirjaimien osalta eroavia tiedostoja ei voida käyttää.

Kuvassa 3 on esitetty tiedostopalvelimen pääluokat. Tarkastelen niitä tarkemmin myöhemmissä osioissa.




*Kuva 3 Tiedostopalvelimen pääluokat ja niiden suhteet /2/*

Käyttääkseen tiedostopalvelinta ohjelman on aloitettava istunto sen kanssa. Istuntoja käytetään kaikkiin tiedostoihin liittyviin operaatioihin. Kuvassa 3 esiintyvä `RFs` on juuri tällainen istunto.

### 4.2.1 File Server -istunnot

Kaikki palvelimet käyttävät istuntopohjaista kommunikaatiota ja niin myös tiedostopalvelin. Tämä tapahtuu niin, että asiakasohjelma lähettää tiedostopalvelimelle haluamansa funktion, kuten vaikkapa tiedostoon kirjoittamisen funktiolla `RFile::Write()`, muunnettuna viestiksi. Palvelin suorittaa halutun käskyn ja saadut tulokset palautetaan asiakkaalle. Ilman istuntoa ei voida tehdä mitään.

Tiedostopalvelimen käyttö koostuu seuraavanlaisista toiminnoista

- Yhdistetään `RFs` tiedostopalvelimeen
- Avataan tiedosto, samalla määritellään käytettävä `RFs`
- Tehdään mitä on tarkoitus
- Suljetaan tiedosto
- Suljetaan `RFs` käyttäen `Close()`-funktiota

`RFs` sisältää seuraavanlaisia tiedoston hallintaan liittyviä toimintoja:

- Senhetkisen hakemiston käyttö
- Hakemistojen teko, poisto ja uudelleen nimeäminen
- Tiedostojen poisto ja uudelleen nimeäminen
- Tiedostojen ja hakemistojen attribuuttien muuttaminen
- Muutoksista ilmoittaminen
- Levyasemien ja tallenteiden käyttö
- Tiedostojärjestelmien lisäys ja poisto

### 4.2.2 Tiedostojen käyttö

Jokaista avattua tiedostoa edustaa `RFile`-olio. Tiedosto voidaan avata käyttäen jotain neljästä `RFile`n tarjoamasta funktiosta. Nämä funktiot ovat:

- `Open()`, joka avaa jo olemassa olevan tiedoston lukemista tai kirjoittamista varten.

- `Create()`, joka luo uuden tiedoston kirjoittamista varten.
- `Replace()`, joka tuhoaa olemassa olevan tiedoston ja luo uuden kirjoittamista varten.
- `Temp()`, joka avaa väliaikaisen tiedoston ja varaa nimen sitä varten.

Tiedoston avaamisen jälkeen siitä voidaan lukea käyttämällä `Read()`-funktiota tai kirjoittaa siihen käyttämällä `Write()`-funktiota. Tiedostosta voi myös etsiä kohdan käyttämällä `Seek()`-funktiota tai siihen voidaan ajaa palvelimen puolelta kirjoituspuskuri `Flush()`-funktiolla.

Useita käyttötapoja tuetaan: jaettu luku, rajattu kirjoitus tai jaettu kirjoitus. Jos tiedosto ei tue käytettävää käyttötapaa, operaatio estetään. Normaalisti käyttötapa määritellään tiedostoa avattaessa, mutta on myös mahdollista muuttaa sitä tiedoston ollessa avoinna käyttämällä `ChangeMode()`-funktiota. Käytettäessä jaettua lukua voidaan tiedosto väliaikaisesti lukita rajattuun pääsyyn tietyille tiedoston osille funktiolla `Lock()`. Tämä lukitus voidaan poistaa käskyllä `UnLock()`. Tiedosto voidaan myös nimetä uudelleen käyttämällä `Rename()`-funktiota.

### 4.2.3 Hakemistot

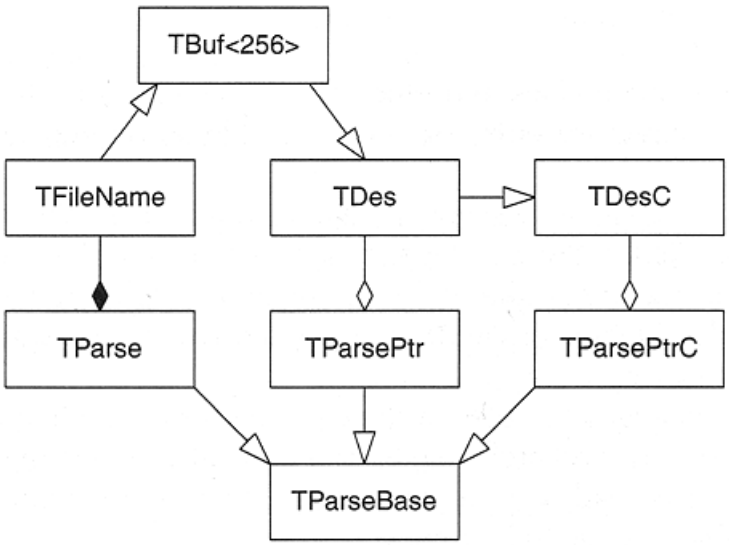
Hakemiston sisältämiä tiedostoja ja muita hakemistoja kuvataan hakemistomerkinnällä (**Directory Entry**) tai lyhyemmin vain merkinnällä (**Entry**). Kuvassa 3 esiintyvä `RDir`-luokka mahdollistaa hakemiston kaikkien merkintöjen läpi käymisen iteroimalla. `TEntry` taas sisältää yhden yksittäisen merkinnän.

Koska tiedostopalvelimen kutsuminen erikseen jokaisen hakemistomerkinnän kohdalla on aikaa vievää, RFs tarjoaa korkeamman tason funktion `GetDir()`:in, joka hakee useamman merkinnän `CDir`:iin.

### 4.2.4 Tiedostonimien käyttö

Tiedostonimiä ei tulisi muokata suoraan itse, vaan tähän tarkoitukseen olisi hyvä käyttää `TParseBase`-luokkaa. Tiedostonimi sisältää yleensä neljä osaa: Levyasematunnuksen, hakemistopolun, tiedostonimen ja tiedostopäätteen.

Kuvassa 4 on esitelty tiedostonimien hallintaan liittyvät luokat.




*Kuva 4 tiedostonimien hallinnan luokkahierarkia /1/*

Perusluokka tiedostonimien jäsentelyyn on `TParseBase`. Sillä on kolme toteutusta:

- `TParse`, joka sisältää kokonaisen tiedostonimen ja on siten iso olio.
- `TParsePtr`, joka toimii kuin `TPtr` siinä, että se viittaa ulkopuoliseen muokattavissa olevaan puskuriin.
- `TParsePtrC`, joka toimii kuin `TPtrC` siinä, että se viittaa ulkopuoliseen puskuriin jota ei voi muokata.

`TFileName` on vain yksinkertainen `TBuf<256>`-deskriptori, joka tarpeeksi pitkä sisältämään pisimmän mahdollisen tiedostonimen. Se on iso olio, eikä sitä sen vuoksi tulisi koskaan sijoittaa pinoon. Olisi hyvä käyttää mahdollisuuksien mukaan viittausta tai jotain muuta vähemmän tilaa vievää deskriptori-tyyppiä. Esimerkiksi, jos tiedetään varmasti, että tiedostonimi on vain 20 merkkiä pitkä, voidaan käyttää `TBuf<20>`:tä.

`TParseXxx`-luokat mahdollistavat kyselyjen suorittamisen tiedostoon tai tiedostonimen muokkaamisen.

- `TParsePtrC` käyttää olemassa olevaa tiedostonimeä viittaamalla siihen `const TDesC&` -deskriptorilla. Tämä mahdollistaa kaikkien `TParseBase`-luokan tukemien kyselytoimintojen käytön.
- `TParsePtr` käyttää olemassa olevaa tiedostonimeä viittaamalla siihen `TDes&`-deskriptorilla.
- `TParse` sisältää tiedostonimen `TFileName` muodossa. Kuten aiemmin mainittiin, tämä on iso olio, joten `TParse-luokkaa` ei kannata käyttää, jos jompikumpi kahdesta muusta luokista käy.

`TParseBase`n kyselytoiminnot mahdollistavat mm. tiedostonimen kunkin elementin hakemisen yksittäin tai tiedostonimen hakemisen kokonaisuudessaan. Voidaan myös kysyä, onko jokin tietty elementti olemassa. Jokerimerkkien käyttö on myös tuettu.

## 4.3 Tietovirrat eli Streamit

Tietovirtojen kaksi keskeisintä luokkaa ovat `RWriteStream`, joka ulkoistaa datan ja `RReadStream`, joka sisäistää datan. Tarkastellaan aluksi mitä nämä asiat tarkoittavat.

### 4.3.1 Ulkoinen ja sisäinen muoto

Data säilyy eri muisteissa erilaisessa formaatissa. Prosessorin sisällä ja ajettavassa muistissa olevan datan muoto määräytyy prosessorin tyypin, C++-kääntäjän ja ohjelman toiminnallisuuden mukaan. Tämän tyyppisen datan sanotaan olevan sisäisessä muodossa, *Internal format.* Tieto voi olla jakautuneena muistiin hajanaisesti.

Data, jota säilötään tiedostossa massamuistissa tai jota siirretään jotain siirtomediaa käyttäen, on ulkoisessa muodossa, *Exernal format.* Tässä muodossa bittien ja tavujen tarkalla järjestyksellä on väliä. Myöskään osoittimia ei voida käyttää, vaan sisäisessä muodossa oleva tieto on muunnettava sarjamuotoon. Vastaavasti ulkoisessa muodossa oleva tieto on palautettava takaisin alkuperäiseen muotoonsa. Ulkoisessa muodossa olevaa dataa voidaan myös pakata tai salata.

### 4.3.2 Tapoja sisäistää ja ulkoistaa dataa

On muutama tapa sisäistää ja ulkoistaa dataa.

- Voidaan käyttää sijoitus- ja hakuoperaattoreita: ulkoistus: `stream << olio` ja sisäistys: `stream >> olio`
- Voidaan ulkoistaa `olio.ExternalizeL(stream)` ja sisäistää `olio.InternalizeL(stream)`
- Voidaan yhdistää muistinvarausrakentaja ja sisäistys yksittäiseen funktioon: `olio=luokka::NewL(stream)`

On olemassa useita luku- ja kirjoitusvirtaluokkia, jotka periytyvät `RWriteStream`ista ja `RReadStream`ista ja jotka käyttävät eri olioihin tallennettuja virtoja. Näitä olioita ovat mm. tiedostot, muisti, tietovirtasäilöt eli *Stream storet* ja hakemistosäilöt eli *Dictionary storet.* Joitain tietovirtoja käytetään tiedon esikäsittelyyn ennen kuin se kirjoitetaan toiseen tietovirtaan.

Seuraavissa koodiesimerkissä `kirjoitin` on jonkun `RWriteStreamin` aliluokan olio ja `lukija` on jonkun `RReadStreamin` aliluokan olio.

#### 4.3.2.1 Sijoitus- (<<) ja hakuoperaattorien (>>) käyttö

Symbianin sisäänrakennettujen tyyppien ulkoistus onnistuu käyttämällä <<, kuten seuraavassa esimerkissä.

```
TInt32 x;
kirjoitin << x;
TBuf <20> text = KText;
kirjoitin << text;
```

Sisäistys onnistuu samalla tavalla käyttämällä vuorostaan >>.

```
TInt32 x;
Lukija >> x;
TBuf <20> text;
Lukija >> text;
```

Sijoitus- ja hakuoperaattoreita ei voida kuitenkaan käyttää aina kaikissa tapauksissa. Käytettäessä <<- ja >>-operaattoreita on aina käytettävä funktiota, joka määrittää käytetyn datan ulkoisessa muodossa viemän tilan. Esimerkiksi `TInt` määritellään olevan vähintään 32 bittiä pitkä, mutta se voi olla myös pidempi. Yritettäessä kirjoittaa tätä tyyppiä virtaan, tulee käännösvirhe. Sen sijaan on käytettävä jotain seuraavassa väliotsikossa määriteltyä tyyppiä.

#### 4.3.2.2 WriteXxxL()- ja ReadXxxL()-funktiot

WriteXxxL()- ja ReadXxxL()-funktioita voidaan käyttää, kun halutaan tarkasti määritellä kuinka data ulkoistetaan. Seuraavana pieni koodiesimerkki:

```
TInt i = 37;
kirjoitin.WriteInt8L();
...
TInt j = lukija.ReakInt8L();
```

Tässä tapauksessa on erittäin selvää, että halutaan käyttää 8-bittistä ulkoista formaattia. Seuraavassa taulukossa 1 on esitetty kaikki integer-tyyppisiin muuttujiin liittyvät WriteXxxL() ja ReadXxxL()-funktiot.

*Taulukko 3 WriteXxxL()- ja ReadXxxL()-funktiot*

| RWriteStream funktio | RReadStream funktio | <<-tyyppi | Ulkoinen formaatti |
|---|---|---|---|
| `WriteL()` | `ReadL()` | | Data sisäisessä formaatissa |
| `WriteL(RreadStream&)` | `ReadL(RwriteStream&` | | Siirto toisesta tietovirtatyypistä |
| `WriteInt8L()` | `ReadInt8L()` | `TInt8` | 8-bittinen etumerkillinen kokonaisluku |
| `WriteInt16L()` | `ReadInt16L()` | `TInt16` | 16-bittinen etumerkillinen kokonaisluku |
| `WriteIntL32()` | `ReadInt32L()` | `TInt32` | 32-bittinen etumerkillinen kokonaisluku |
| `WriteUint8L()` | `ReadUint8L()` | `TUint8` | 8-bittinen etumerkitön kokonaisluku |
| `WriteUint16L()` | `ReadUint16L()` | `TUint16` | 16-bittinen etumerkitön kokonaisluku |
| `WriteUint32L()` | `ReadUint32L()` | `TUint32` | 32-bittinen etumerkitön kokonaisluku |
| `WriteReal32L()` | `ReadReal32L()` | `TReal32` | 32-bittinen liukuluku |
| `WriteReal64L()` | `ReadReal64L()` | `TReal64` | 64-bittinen liukuluku |

Jos käytät <<- ja >>-operaattoreita sisäänrakennettujen tyyppien kanssa, ne kutsuvat taulukossa 3 olevia funktioita. Taulukon 3. sarake <<-tyyppi näyttää, mitä tyyppiä Symbian käyttää kutsuttaessa kyseessä olevaa funktiota.

#### 4.3.2.3 Raaka data

Tarkastellaan `WriteL()` ja `ReadL()`-funktioiden käyttöä lähemmin. Niiden määrittelyt löytyvät s32strm.h tiedostosta. WriteL()-funktion määrittelyistä löytyvät seuraavat 8 bittiset instanssit:

```
IMPORT_C void WriteL(const TDesC8& aDes );
IMPORT_C void WriteL(const TDesC8& aDes, TInt aLength );
IMPORT_C void WriteL(const TUint8* aPtr, TInt aLength );
```

Samat löytyvät myös 16 bittisinä:

```
IMPORT_C void WriteL(const TDesC16& aDes );
IMPORT_C void WriteL(const TDesC16& aDes, TInt aLength );
IMPORT_C void WriteL(const TUint16* aPtr, TInt aLength );
```

Nämä funktiot yksinkertaisesti kirjoittavat määritellyn datan. Ne noudattavat seuraavia sääntöjä:

- `aLength`-parametri määrittää kuinka monta tavua kirjoitetaan.
- Ilman `aLength`-parametriä koko deskriptori kirjoitetaan.
- `const TUint8* aPtr` -versio kirjoittaa määritellyn määrän tavuja osoittimen osoittamasta paikasta alkaen.
- `TDesC16` ja `TUint16` versiot kirjoittavat tavujen sijasta Unicode merkkejä.

S60 SDK Helpistä löytyy kunkin funktion tarkempi määrittely. SDK help jaetaan Symbian SDK:den mukana, ja on suuri apu varsinaista koodaustyötä tehdessä.

ReadL()-funktiolle löytyy samankaltaiset, joskaan ei symmetriset funktiot:

```
IMPORT_C void ReadL(TDes8 &aDes);
IMPORT_C void ReadL(TDes8 &aDes, TInt aLength);
IMPORT_C void ReadL(TDes8 &aDes, TChar aDelim);
IMPORT_C void ReadL(TUint8* aPtr, TInt aLength);
IMPORT_C void ReadL(TInt aLength);

IMPORT_C void ReadL(TDes16 &aDes);
IMPORT_C void ReadL(TDes16 &aDes, TInt aLength);
IMPORT_C void ReadL(TDes16 &aDes, TChar aDelim);
IMPORT_C void ReadL(TUint16* aPtr, TInt aLength);
```

Lukemisessa ongelma on se, että kuinka tiedetään milloin lopetetaan. Kirjoittaessa deskriptorin pituus määrää datan pituuden. Kirjoittaessa toimitaan seuraavasti:

- `TDes8& aDes` formaatissa luetaan deskriptorin `MaxLength()`-bitit, joiden mukainen pituus luetaan.

- `aLength`-parametri määrittelee, kuinka monta tavua luetaan.
- Kun määritellään erotinmerkki (`aDelim`), stream lopettaa datan lukemisen erotinmerkin jälkeen. On tärkeä huomata, että tämä tarkoittaa sitä, että myös erotinmerkki luetaan. Jos kyseessä olevan deskriptorin `MaxLength()` tulee vastaan ennen erotinmerkkiä, lukeminen lopetetaan `MaxLength()` merkkien jälkeen.

Muiden `ReadXxxL()`-funktioiden tavoin nämä funktiot voivat aiheuttaa leaven `KErrEof`-virheellä, jos tiedoston loppu tulee vastaan lukuoperaation yhteydessä.

Näitä funktioita tulisi käyttää suurella varovaisuudella. WriteL() periaatteessa vain kopioi datan sellaisenaan streamiin, joten on hyvä varmistautua, että data on valmiiksi ulkoisessa muodossa.

#### 4.3.2.4 ExternalizeL()- ja Internalize()-funktiot

Jos käytössä on tietyn tyyppinen olio, jonka luokka on itse kirjoitettu, sille on kirjoitettava myös omat `Externalize()`- ja `Internalize()`-funktiot. Näiden funktioiden tulisi olla määritelty seuraavalla:

```
class oma
{
public:
          ...
          void Externalize( RWriteStream& aStream ) const;
          void Internalize( RWriteStream& aStream );
          ...
};
```

Operaattoreita << ja >> varten on olemassa yleinen pohjatemplate, mutta käytännössä luokkaa varten olisi hyvä kirjoittaa niille tarkempi toteutus.

## 4.4 Storet eli tietosäilöt

Useat järjestelmät tarjoavat Symbianin tavoin stream API:n, mutta Symbian menee asiassa vieläkin pidemmälle. Symbian suunniteltiin alun perin tiedostopohjaiset storet eli tietosäilöt mielessä. Näitä tietosäilöjä on kahta eri tyyppiä: direct file store ja permanent file store. Tarkastellaan niitä lähemmin.

### 4.4.1 Direct file store

Tämän tyyppisiä tietosäilöjä käytetään usein erilaisissa lataa/tallenna -tyyppisissä sovelluksissa, joissa dataa säilytetään RAM-muistissa ja kokonainen dokumentti ladataan ja tallennetaan kerrallaan tarvittaessa.

Käytännössä tämän tyyppinen tietosäilö on vain keskitetty paikka, johon ohjelman streamit tallennetaan. Se muodostuu stream kirjastosta, joka sisältää osoitukset siihen tallennettuihin streameihin.

Tämän tyyppisiin tietosäilöihin liittyy kuitenkin tiettyjä rajoituksia. Nämä rajoitukset tekevät kuitenkin näiden säilöjen käytöstä yksinkertaista ja helppoa. Kun direct file store -tyyppisiä tietosäilöjä halutaan käyttää, tietosäilö ladataan kokonaisuudessaan keskusmuistiin, missä sitä säilytetään toimintojen ajan. Levylle kirjoitettaessa se tallennetaan kokonaisuudessaan vanhan tiedoston päälle.

### 4.4.2 Permanent file store

Dictionary file store -tyyppistä tietosäilöä käytetään, kun halutaan käyttää vain osaa tiedosta kerralla ja säilyttää loppuja kiinteässä muistissa tietokannan tavoin.

Tämän tyyppistä tietosäilöä käyttäessä voidaan tehdä seuraavia asioita:

- Tietosäilöön luotua streamiä voidaan käyttää vapaasti. Siihen voidaan lisätä asioita ja se voidaan tuhota.
- Tietosäilössä voidaan pitää haluttu määrä streameja auki lukemista ja kirjoittamista varten.
- Tietosäilö voidaan avata sulkemisen jälkeen ja sitä voidaan muokata vapaasti.

Tämän vapauden hintana on se, että tietosäilöä käytettäessä on oltava erittäin huolellinen. Koska tietosäilö säilyy samanlaisena, vaikka sitä käyttävä sovellus sammutetaan ja käynnistetään uudestaan, on oltava todella varovainen että säilön eheys säilyy.

## 4.5 Esimerkki tiedostoon kirjoittamisesta

Alla olevassa esimerkissä on luokka CKirjoittaja, joka kirjoittaa tiedostoon sille tuodun deskriptorin. Olio luodaan ConstructL()- ja NewL()-metodeisssa käyttäen kaksivaiheista rakennusta ja tiedostoon kirjoitetaan WriteToFile()-metodissa.

CKirjoittaja.cpp

```
CKirjoittaja* CKirjoittaja::NewL( const TDesC& aFileB )
{
    CKirjoittaja* self = new(ELeave) CKirjoittaja;
    CleanupStack::PushL(self);
    self->ConstructL( aFileB );
    CleanupStack::Pop(self);
    return self;
}

void CKirjoittaja::ConstructL( const TDesC& aFileA)
{
    iFileName = aFileA;
    User::LeaveIfError( iFs.Connect() );
    iFile.Replace(iFs, iFileName, EFileWrite);
    iFile.Close();
}

void CKirjoittaja::WriteToFile( const TDesC& aString)
{
//yhdistetään file serveriin

    User::LeaveIfError( iFs.Connect() );
    CleanupClosePushL( iFile );
```

```
// avataan tiedosto
    User::LeaveIfError( iFile.Open( iFs, iFileName, EFileWrite) );
    TFileText file;
    file.Set( iFile );

//kirjoitetaan tiedostoon käyttäen TFileText:iä
    TInt err=file.Write( aString );

//kirjoitetaan tiedostoon käyttäen TFile:iä
    iFile.Write( aString );
    CleanupStack::PopAndDestroy(&iFile);
}

CKirjoittaja::CKirjoittaja(){}

CKirjoittaja::~CKirjoittaja(){}
```

CKirjoittaja.h:

```
class CKirjoittaja : public CBase
{
public:
    static CKirjoittaja* NewL( const TDesC& aFileB );
    void WriteToFile( const TDesC& aString);
    void ConstructL( const TDesC& aFileA);
    CKirjoittaja();
   ~CKirjoittaja();
private: //data
    RFile iFile;
    RFs iFs;
    TBufC<20> iFileName;
}
```

# 5 MOBILE LINUX

Linuxia käyttöjärjestelmänään käyttäviä mobiililaitteita on useita. Tässä osuudessa keskityn Nokian Maemo -kehitysympäristöön.

Maemo on avoin Linux-pohjainen ohjelmistokehitysalusta mobiililaitteille. Sen vahvuuksia ovat muun muassa olemassa olevien ohjelmistojen helppo siirrettävyys mobiiliversioiksi ja aiemman Linux-tietämyksen hyödynnettävyys. Ensimmäinen Maemoa käyttävä laite oli Nokia 770 Internet Tablet ja toinen siihen pohjautuva laite on uusi Nokia N800 Internet Tablet.

## 5.1 File I/O funktiot

Maemo käyttää tiedostojen avaamiseen, lukemiseen ja kirjoittamiseen GnomeVFS –kirjastoa. Sen avulla voidaan käyttää useita eri tyyppisiä tiedoston käsittelyyn liittyviä tapoja yhden rajapinnan kautta. Se tukee paikallisen tiedostojärjestelmän lisäksi mm. ftp-protokollaa.

### 5.1.1 GnomeVFS

GnomeVFS on lyhenne Gnome Virtual File Systemistä. Sen ansiosta käyttäjä ja sovelluskehittäjä voivat käyttää erilaisia tiedostojärjestelmiä samalla tavalla.

GnomeVFS API on pidetty mahdollisimman lähellä Unix-maailman POSIX-standardia lukuun ottamatta muutamia muutoksia. Esimerkiksi tiedoston deskriptorien sijaan käytetään GnomeVFSHandlea ja kaikista I/O operaatioista palautetaan GnomeVFSResult, joka kertoo operaation tuloksen.

GnomeVFS tukee asunkronisia operaatiota ja tarjoaa metodit esimerkiksi tiedostojen hakemiseen taustalla.

GnomeVFS:ään voidaan myös kirjoittaa omia moduleita, joilla voidaan tukea jotain tiettyä tiedostojärjestelmää tai tietoliikenneprotokollaa. /4/

### 5.1.2 Tiedostosta lukeminen

Tiedoston luku tapahtuu käyttäen GnomeVFS:n funktioita. Aluksi haetaan tiedoston tiedot ja muodostetaan kahva tiedostoon. Tämän jälkeen tiedosto avataan ja luetaan teksti puskuriin. Kun data on luettu, tiedosto suljetaan.

Koodiesimerkissä yksinkertainen tekstinkäsittelyohjelma lukee tekstin tiedostosta ja tulostaa sen näytölle.

```
void LuetaanTiedostoPuskuriin( MainView* mainview )
{

// Yritetään saada tiedoston tiedot

     vfs_result = gnome_vfs_get_file_info(mainview->file_name,
&finfo, GNOME_VFS_FILE_INFO_DEFAULT);

    if ( vfs_result != GNOME_VFS_OK ) {interface_error(
MAEMOPAD_ERROR_OPEN_FAILED, mainview );
        return;
    }

// yritetään muodostaa kahva tiedostoon

    vfs_result = gnome_vfs_open
          (&handle, mainview->file_name, GNOME_VFS_OPEN_READ);
    if ( vfs_result != GNOME_VFS_OK ) {
        interface_error( MAEMOPAD_ERROR_OPEN_FAILED, mainview );
        return;
    }

// Varataan muistia temp_bufferia varten

    temp_buffer = g_malloc(finfo.size + 1);
    memset(temp_buffer, 0, finfo.size + 1);

// Luetaan tiedostosta puskuriin

    gnome_vfs_read(handle, temp_buffer, finfo.size, &in_bytes);
```

```
// Tulostetaan teksti ruudulle

    gtk_text_buffer_set_text
          ( GTK_TEXT_BUFFER (mainview->buffer), temp_buffer, -1);

// Vapautetaan temp_buffer, suljetaan tiedosto ja palataan

    g_free(temp_buffer);
    gnome_vfs_close(handle);
}
```
/3/

### 5.1.3 Tiedostoon kirjoittaminen

Tiedostoon kirjoittaminen tapahtuu myös GnomenVFS:n funktioita käyttäen. Kuten lukemisoperaatiossa, aluksi tarvitaan kahva tiedostoon. Tiedot kopioidaan puskuriin talteen ja puskuri kirjoitetaan tiedostoon. Lopuksi suljetaan kahva tiedostoon.

Koodiesimerkissä yksinkertaisen tekstikäsittelyohjelma kirjoittaa ruudulla olevan tekstin tiedostoon.

```
void write_buffer_to_file ( MainView* mainview )
{

// Yritetään luoda kahva tiedostoon

    vfs_result = gnome_vfs_create
          (&handle, mainview->file_name, GNOME_VFS_OPEN_WRITE, 0,
          0600);
    if ( vfs_result != GNOME_VFS_OK ) {
        interface_error( MAEMOPAD_ERROR_SAVE_FAILED, mainview );
        return;
    }

// Etsitään tekstin alku ja loppu

    gtk_text_buffer_get_bounds
          ( GTK_TEXT_BUFFER (mainview->buffer), &start, &end);

// Kopioidaan kaikki tekstit temp_bufferiin

    temp_buffer = gtk_text_buffer_get_slice
          ( GTK_TEXT_BUFFER (mainview->buffer), &start, &end, TRUE);

// Kirjoitetaan teksti tiedostoon
```

```
    gnome_vfs_write(handle, temp_buffer, strlen(temp_buffer),
&out_bytes);

// Vapautetaan väliaikainen puskuri, suljetaan tiedosto ja palataan

    g_free(temp_buffer);
    gnome_vfs_close(handle);
}
```

/3/

# 6 MICROSOFT WINDOWS MOBILE

Microsoftin Windows Mobile perustuu Windows käyttöjärjestelmään ja käyttää hyväkseen .NET sovelluskehystä. .NETin sisältämä `System.IO` -nimiavaruus sisältää tyypit, jotka mahdollistavat synkronisen ja asynkronisen kirjoittamisen ja lukemisen sekä streameihin että tiedostoihin.

.NET-ohjelmointia voidaan tehdä mm. C#- ja Visual Basic-ohjelmointikielillä. Koodiesimerkit esitän C#-kielisinä. Ne löytyvät myös Microsoftin MSDN–nettisivuilta.

## 6.1 File I/O -luokat

Esittelen muutamia tärkeitä luokkia.

Tiedostojen käsittelyyn liittyy hakemistojen luominen, siirto ja niiden selaaminen. Näitä toimintoja tarjoaa kaksi luokkaa: `Directory` ja `DirectoryInfo`. `DriveInfo`-luokka tarjoaa metodeja informaation hakemiseen asemasta.

`File`-luokka tarjoaa staattisia metodeita tiedostojen luomiseen, kopioimiseen, poistamiseen, siirtämiseen ja avaamiseen sekä auttaa `FileStreamin` luomisessa. `FileInfo`-luokka tarjoaa instanssi metodit samoille asioille. Näiden luokkien erona on se, että `File`-luokkaa voidaan käyttää ilman että siitä luodaan instanssia. `FileInfo`-luokkaa käytettäessä on aluksi luotava siitä instanssi. Vasta sen jälkeen voidaan käyttää luokan metodeja. /6/

`FileStream` tukee tiedostojen lukemista satunnaisesti käyttäen sen `Seek`- eli etsi-metodia. Se avaa tiedostot oletusarvoisesti synkronisesti, mutta tukee myös asynkronista operaatiota.

`FileSystemInfo` on `FileInfon` ja `DirectoryInfon` abstrakti kantaluokka. `Path` tarjoaa metodeita hakemistopolku merkkijonojen käsittelyyn eri alustoille sopivalla tavalla. /5/

## 6.2 Streamit

Streameihin kirjoittamiseen ja niistä lukemiseen käytetään useita erilaisia luokkia. Nämä kaikki luokat periytyvät `Stream`-luokasta. Stream-luokka on suuniteltu abstraktina luokkana ja muut luokat toteuttavat sen määrittelemät toiminnallisuudet.

`BinaryReader` ja `BinaryWriter` lukevat ja kirjoittavat koodattuja merkkijonoja ja binääritason data tyyppejä streameihin.

`StreamReader` lukee merkkejä streameistä käyttäen koodausta muuttaessaan merkit biteistä merkeiksi ja takaisin. Kyseisen luokan rakentaja yrittää varmistaa oikean koodauksen käytön. `StreamWriter` taas kirjoittaa merkkejä streamiin muuttaen merkit biteiksi.

`StringReader` lukee merkkejä stringeistä eli merkkijonoista ja `StringWriter` kirjoittaa merkkejä merkkijonoon. Nämä kaikki luokat käyttävät samaa API:a, joten on mahdollista valita käyttääkö streamia vai stringiä.

`TextReader` on abstrakti kantaluokka `StreamReader`ille ja `StringReader`ille. `TextWriter` on sama `StreamWriter`ille ja `StringWriter`ille. Siinä missä stream-luokat on suunniteltu toteuttamaan bitti tyyppisiä syötteitä TextXXX toteuttaa samat asiat Unicode merkkeinä. /5/

## 6.3 Esimerkkejä System-luokkien ja Streamien käytöstä

### 6.3.1 Tyypillinen `FileStream`-luokan käyttötapaus

Ensiksi luodaan `FileStream`-olio ja määritellään tiedostonimi ja `FileMode`. `FileMode` on enumeraatio parametri, jolla määritellään mitä tiedostolle tehdään: avataanko se vai luodaanko uusi tiedosto.

Yleisiä `FileMode`-arvoja on neljä:

- `FileMode.Open` – avaa olemassa olevan tiedoston. Jos tiedostoa ei löydy, heittää virheen.
- `FileMode.Create` – luo uuden tiedoston. Jos tiedosto on jo olemassa, sen päälle kirjoitetaan uusi.
- `FileMode.OpenOrCreate` – avaa tiedoston, jos se on olemassa, muuten luodaan uusi tiedosto.
- `FileMode.Append` – avaa tai luo tiedoston samalla tavalla kuin `FileMode.OpenOrCreate`, jonka jälkeen hakee tiedoston lopun kirjoitusta tai lukemista varten.

Toiseksi määritellään tiedoston luku- ja jako-oikeudet, jos se on tarpeen. `FileAccess` enumeraatio parametrilla on kolme arvoa: `Read`, `Write` ja `ReadWrite`. Näillä arvoilla määritellään mitä ohjelma saa tehdä tiedostolle. Ohjelma voi siis joko vain lukea, kirjoittaa tai tehdä molempia. `FileAccess`ia käyttäen saadaan ohjelmasta turvallisempi. Esimerkiksi vain luettavaksi tarkoitettuun tiedostoon ei pääse vahingossa kirjoittamaan.

`FileShare`-enumeraatio määrittelee kuinka muut ohjelmat voivat käyttää tiedostoa. Sille on tyypillisesti neljä arvoa: `FileShare.Read`, `FileShare.Write`, `FileShare.ReadWrite` ja `FileShare.None`. Muut

ohjelmat voivat siis joko vain lukea tiedostoa, kirjoittaa tiedostoon, tehdä molempia tai eivät mitään, jos on valittu `FileShare.None`.

Kolmanneksi luetaan tiedostosta käyttämällä `Read()`- tai `ReadByte()`-metodeja tai kirjoitetaan tiedostoon käyttämällä `Write()`- tai `WriteByte()`-metodeja.

Lopuksi suljetaan `FileStream` käyttämällä `Close()`-metodia.

Tiedostoja käytettäessä on muistettava riittävä virheen käsittely. On erittäin suositeltavaa että tiedoston käsittely koodi sijoitetaan aina `try...catch` -lauseen sisään. Vaikka itse kirjoitettu ohjelmakoodi olisi itsessään täysin virheetöntä, on silti otettava huomioon, että käytössä oleva muisti on rajallinen ja sitä käyttävien muiden ohjelmien muistin käsittely saattaa aiheuttaa virhetilanteita. Siksi onkin aina hyvä kirjoittaa ohjelmaan sopiva virheenkäsittely `catch`-osioon. /6/

### 6.3.2 Hakemistojen ja tiedostojen käsittely

Tämä koodiesimerkki käyttää I/O-luokkia luodakseen listauksen hakemistosta löytyvistä .exe-päätteisistä tiedostoista. Esimerkissä `DirectoryInfo` on nykyinen hakemisto, jota ilmaistaan ”.” -merkillä. Koodi listaa kaikki .exe-päätteiset tiedostot sekä niiden koon, luontiajan ja nimen.

Tarkempi koodin selitys on kommentoitu rivien väliin.

```
using System;
using System.IO;
class HakemistoListaaja
{

// HakemistoListaaja luokalle tuodaan String –muotoisena
// haluttu hakemistopolku.

    public static void Main(String[] args)
    {
```

```
//Tarkastetaan onko tuotu hakemistopolku olemassa, jos
// ei ole, käytetään hakemistoa jossa ohjelma sijaitsee.

        string polku = ".";
        if (args.Length > 0)
        {
            if (File.Exists(args[0])
            {
                  polku = args[0];
            }
            else
            {     Console.WriteLine("{0} ei loytynyt;
                  Kaytetaan nykyista hakemistoa:",
                  args[0]);
            }

// Luodaan DirectoryInfo luokka ja asetetaan sille
// hakemistopolku.

        DirectoryInfo hakemisto = new DirectoryInfo( polku );

// Haetaan .exe-päätteiset tiedostot.

        foreach (FileInfo tiedosto in hakemisto.GetFiles( "*.exe" ))
        {

// Otetaan talteen tiedoston nimi, koko ja luontiaika.

            String nimi = tiedosto.Name;
            long koko = tiedosto.Length;
            DateTime luontiAika = tiedosto.CreationTime;

// Tulostetaan konsolinäkymään tiedoston tiedot.

            Console.WriteLine("{0,-12:N0} {1,-20:g} {2}", koko,
                  luontiAika, nimi);
        }
    }
}
```

/5/

### 6.3.3 Tiedostoon kirjoittaminen

Tiedostoon kirjoittamisesta löytyy kaksi esimerkkiä. Ensimmäisessä esimerkissä kirjoitetaan tekstiä jo olemassa olevaan tiedostoon. Toisessa esimerkissä luodaan tekstitiedosto ja siihen kirjoitetaan merkkijono. Molemmissa esimerkeissä käytetään `system` ja `system.IO` -kirjastoja.

```
class KirjoitusValmiiseenTiedostoon
{
    public static void Main()
    {

// Luodaan StreamWriter-luokan instanssi tekstin
// tiedostoon kirjoittamista varten. Using-käsky sulkee
// loputtuaan myös StreamWriterin.

        using (StreamWriter sw = new StreamWriter("Tiedosto.txt"))
        {

// Lisätään tekstiä tiedostoon.

            sw.Write("Tassa on  ");
            sw.WriteLine("tiedoston otsikko.");
            sw.WriteLine("--------------------");

// Tiedostoon voidaan kirjoittaa myös sattumanvaraisia
// asioita.

            sw.Write("paivamaara on: ");
            sw.WriteLine(DateTime.Now);
        }
    }
}
```

Tässä jälkimmäisessä esimerkissä on luokka, joka luo tiedoston johon kirjoitetaan merkkijono.

```
public class TekstiaTiedostoon
{
// Määritellään käytettävä tiedoston nimi.

    private const string TIEDOSTO_NIMI = "tekstitiedosto.txt";
    public static void Main(String[] args)
    {

// Tarkistetaan onko kyseinen tiedosto jo olemassa.

        if (File.Exists(TIEDOSTO_NIMI))
        {
            Console.WriteLine("{0} on jo olemassa.", TIEDOSTO_NIMI);
            return;
        }

// Luodaan tiedosto

        using (StreamWriter sw = File.CreateText( TIEDOSTO_NIMI ))
        {

// Tiedostoon voidaan nyt kirjoittaa asioita.
```

```
            sw.WriteLine ("Tama on tiedostoni");
            sw.WriteLine ("Voin kirjoittaa kokonaislukuja {0} tai
            liukulukuja {1}, ja niin edelleen.",
            1, 4.2);
            sw.Close();
        }
    }
}
```

Näistä esimerkeistä nähdään, että tiedoston luominen ja tekstin kirjoittaminen on varsin yksinkertaista ja helppoa C#-kielellä. /5/

### 6.3.4 Tiedostosta lukeminen

Seuraavat koodiesimerkit näyttävät miten tiedostosta voidaan lukea tekstiä. Ensimmäisessä esimerkissä käytetään virheen käsittelyä ja jälkimmäisessä ilmoitetaan käyttäjälle tiedoston loppuun pääsystä.

```
class Luku
{
    public static void Main()
    {

// Tehdään tiedoston luku operaatiot try lohkon sisällä
// jotta saadaan mahdolliset virhetilanteet kiinni.

        try
        {

// Luodaan StreamReaderin instanssi tiedostosta lukemista
// varten.
// Using-käsky sulkee loputtuaan myös StreamReaderin.

          using (StreamReader sr = new StreamReader("tiedosto.txt"))
            {
                String rivi;

// Luetaan ja tulostetaan rivit näytölle kunnes tiedosto
// loppuu.

                while ((rivi = sr.ReadLine()) != null)
                {
                    Console.WriteLine(rivi);
                }
            }
        }
```

```
// käsitellään mahdollinen virhetilanne

        catch (Exception e)
        {
            // Kerrotaan käyttäjälle mikä meni pieleen.
            Console.WriteLine("Tiedostoa ei voitu lukea:");
            Console.WriteLine(e.Message);
        }
    }
}
```

Jälkimmäinen esimerkki koodi luo `StreamReader`-olion, joka osoittaa tiedostoon käyttäen `File.OpenText` -kutsua. `StreamReader.ReadLine` palauttaa jokaisen rivin string-tyyppisenä merkkijonona. Kun tiedosto on käyty loppuun, eikä luettavaa enää ole, ilmoitetaan siitä käyttäjällä ja suljetaan stream.

```
public class TextFromFile
{
    private const string TIEDOSTON_NIMI = "Tiedosto.txt";
    public static void Main(String[] args)
    {

// Tarkistetaan onko tiedostoa olemassa.

        if (!File.Exists(TIEDOSTON_NIMI))
        {
            Console.WriteLine("{0} ei loydy.", TIEDOSTON_NIMI);
            return;
        }

// Avataan tiedosto ja luetaan tekstiä.

        using (StreamReader sr = File.OpenText(TIEDOSTON_NIMI))
        {
            String input;
            while ((input=sr.ReadLine())!=null)
            {
                Console.WriteLine(input);
            }

// Ilmoitetaan että tiedosto on luettu loppuun asti.

            Console.WriteLine ("Tiedoston loppu.");

// Suljetaan StreamReader.

            sr.Close();
        }
    }
```

/5/

# 7 YHTEENVETO

Tässä osiossa vertailen aikaisemmissa osioissa tutkimiani järjestelmiä. Tarkastelen mitä yhtäläisyyksiä niillä on, minkälaisia eroja niissä on ja arvioin niiden käytettävyyttä ohjelmoijan kannalta. Käytettävyyttä arvioin sillä kuinka korkea on ohjelmoinnin aloittamisen kynnys, kuinka helppoa on tehdä virheetöntä koodia ja kuinka monimutkaista koodista tulee.

## 7.1 Järjestelmien väliset yhtäläisyydet

Tiedostojen käsittely hoidetaan eri järjestelmissä varsin samankaltaisella tavalla. Pääpiirteissään tiedostoon kirjoittamiseen kuuluu tiedostojärjestelmään yhteyden otto, kahvan hakeminen tiedostoon, tiedoston avaus ja kirjoitus tai luku ja lopuksi tiedoston ja yhteyksien sulkeminen.

Tiedostoon kirjoittamiseen tarjotaan useita tapoja. Tiedostoon voidaan kirjoittaa tekstiä erilaisilla kirjoitus-komennoilla tai tiedostoon voidaan kirjoittaa tietovirta. Kaikissa käsitellyissä järjestelmissä tekstin kirjoitus tiedostoon oli olemassa ja siihen löytyi selkeät ohjeet. Ainakin Symbian ja Windows Mobile tarjoavat lisäksi myös tietovirtojen kirjoittamisen tiedostoon. Maemosta ei löytynyt tähän liittyvää dokumentaatiota, mutta myös Linux käyttöjärjestelmänä tukee tietovirtojen käyttöä.

Tiedostojen käsittelyn samankaltaisuus johtuu siitä, että sen pohjalla oleva syötteiden hallintatekniikka on pitkälti sama ja jo pitkään valmiina ollut teknologia. Laitteet käyttävät paljon samoja muistiteknologioita ja niiden käyttö on muodostunut lähes standardinomaiseksi. Käytössä olevien kehitysympäristöjen käyttämät valmiit kirjastot toteuttavat tarvittavat muistioperaatiot ja nämä kirjastot tarjoavat helppokäyttöisen käyttöliittymän tiedostojen käsittelyyn.

## 7.2 Järjestelmien väliset erot

Suurimmat erot järjestelmien välillä muodostavat jo edellä mainittujen tiedoston käsittelyn toteuttavien kirjastojen käyttöliittymät. Laitteen loppukäyttäjä ei käytännössä huomaa mitään eroa eri järjestelmien tiedon tallennuksen toteutuksessa, sillä käyttöjärjestelmien muut erot ovat huomattavasti suuremmat. Erot huomaa ainoastaan ohjelmoija.

Symbian yleensä assosioi tiedostot aina applikaation mukaan (MIME-tekniikkaa käyttäen), esimerkiksi jos tekstieditorilla tehdään html-tiedosto, se avautuu silti tekstieditoriin eikä selaimeen.

Symbian tarjoaa ohjelmoijalle eniten valinnan varaa siihen, kuinka tallentaa tietoa. Tämä on toisaalta hyvä ja toisaalta huono asia. Huono puoli asiassa on se, että tämä tekee tiedon tallentamisesta entistä monimutkaisempaa ja tällöin virheitä tulee entistä helpommin. Hyvä puoli asiassa on se, että ohjelmoijalla on mahdollisuus optimoida tekemäänsä koodia entistä pitemmälle. Symbian tarjoaa muista eroten lisäksi myös mahdollisuuden käyttää tietovarastoja.

Windows Mobile tarjoaa useita luokkia, joita käyttäen tieto voidaan tallentaa eri muodoissa. Windows ei ole yhtä monimutkainen kuin Symbian, mutta luokat tarjoavat silti runsaasti mahdollisuuksia. Symbiania käyttävällä ohjelmoijalla on mahdollisuus päästä syvemmälle käyttöjärjestelmän sisään ja muokata tarkemmin, minkä tyyppisenä data tallennetaan ja miten dataa muokataan, kun sitä siirretään laitteen käyttömuistista pysyvään muistiin. Windows ei tarjoa näin syvälle meneviä toimintoja.

## 7.3 Käytettävyys

Symbian on näistä kolmesta järjestelmästä vaikein hallita. Kyseessä on monimutkaisin järjestelmä ja sen aloituskynnys on korkea. Vaikkakin se tarjoaa

mahdollisuuden päästä käsiksi hyvinkin syvälle järjestelmään, tilanne, jossa tällaisia mahdollisuuksia oikeasti tarvittaisiin, on harvinainen. Monet asiat on jätetty turhaan monimutkaisiksi. Se, moniko bittistä data on ja miten se ulkoistetaan tai sisäistetään, on yleensä ohjelman toimintalogiikan kannalta epäoleellista. Tämä järjestelmän turha monimutkaisuus hidastaa ohjelmiston kehitystyötä ja aiheuttaa kustannuksia.

Järkevintä olisi, jos järjestelmä pystyisi itse hoitamaan sen miten tieto tallennetaan, ohjelmoijan määritellessä ainoastaan mitä tallennetaan. Vaikka järjestelmän käyttämisen oppiikin, kun sen kanssa tekee riittävästi töitä, aiheutuu tämänkaltaisesta turhasta monimutkaisuudesta tarpeetonta lisävaivaa.

Windows-ohjelmoijan on paljon helpompi tallentaa tietonsa tiedostoihin. Ohjelmoijalle jätetään valittavaksi riittävä määrä tapoja tallentaa tietoa, eikä ohjelmoijan tarvitse välittää siitä miten tieto tallentuu. Windows Mobile -ohjelmointi on myös helpompaa parempien työkalujen ansiosta. Microsoftin Visual Studio ja .Net on valmiimpi ja kehitystyötä tukevampi kehitysympäristö kuin mitä Symbianille on saatavilla.

## LÄHTEET

1. Jane Sales with Andrew Rogers [… et al.], Symbian OS internals : real-time kernel programming, WILEY

2. Richard Harrison with Alan Robinson [… et al.], Symbian OS C++ for Mobile Phones, WILEY

3. Maemo.org, [WWW-sivu], saatavissa: http://www.maemo.org

4. Introduction to Gconf and GnomeVFS | Imendio Developer Pages, [WWW-sivu], saatavissa: http://developer.imendio.com/publications/introduction_gconf_gnomevfs

5. Microsoft Developer Network, [WWW-sivu], saatavissa: www.msdn.com

- Basic File I/O, [WWW-sivu], saatavissa: http://msdn2.microsoft.com/en-us/library/336wast5.aspx
- How to: Create a Directory Listing, [WWW-sivu], saatavissa: http://msdn2.microsoft.com/en-us/library/5cf8zcfh.aspx
- How to: Read and Write to a Newly Created Data File, [WWW-sivu], saatavissa: http://msdn2.microsoft.com/en-us/library/36b93480.aspx

6. Baijian Yang, Pei Zheng, Lionel M. Ni, Professional Microsoft Smartphone Programming, WILEY, 494 s.

7. S60 Platform: Application Framework Handbook, versio 2.0, [PDF-dokumentti], saatavissa: http://www.forum.nokia.com, Nokia Corporation