Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Kirsi-Marja Eskola

Opinnäytetyö

QtCreator Hands-On lab plug-in

Työn ohjaaja Lehtori Tony Torp
Työn teettäjä Tampereen ammattikorkeakoulu
Tampere 2/2010

Tekijä	Kirsi-Marja Eskola
Työn nimi	QtCreator Nokia Hands-on lab plug-in
Sivumäärä	33
Valmistumisaika	12.04.10
Työn ohjaaja	Lehtori Tony Torp
Työn teettäjä	Tampereen ammattikorkeakoulu

TIIVISTELMÄ

Tämä työ käsittelee plug-inin eli liitännäisen kehittämistä QtCreator-sovellukseen. Työn tarkoituksena oli perehtyä QtCreator-kehitystyökaluun, Qt-ohjelmointikieleen sekä QtCreator-liitännäisiin.

Työssä esitellään Qt-sovelluskehitysympäristö sekä sen historiaa, QtCreator-kehitystyökalu sekä tutustutaan liitännäiseen käsitteenä.

Työn toivotaan antavan selkeämmän käsityksen Qt-ohjelmoinnista sitä tarkemmin tuntemattomalle lukijalle. Työn lukijan oletetaan olevan kuitenkin tutustunut C++-ohjelmointikieleen.

Työssä esitellään taustatutkimuksena tehty liitännäinen, joka tuo Nokian tuottaman Hands-on lab -e-oppimateriaalin ohjelmoijan saataville QtCreator-kehitystyökaluun. Liitännäinen esitellään tarkemmin työn lopuksi. Tässä työssä ei käsitellä Qt-ohjelmointikielen tuottamiseen tarkoitettuja muita kehitystyökaluja vaan keskitytään QtCreator-kehitystyökaluun. Tässä työssä ei myöskään käsitellä tarkkaa toimintamenetelmää liitännäisen tuottamiseen, vaan sen työvaiheet käydään läpi pääpiirteittäin.

Avainsanat Qt-ohjelmointi, QtCreator, plug-in, Linux, Nokia, Hands-On lab

Author	Kirsi-Marja Eskola
Name of the report	QtCreator Hands-on lab plug-in
Number of pages	33
Graduation time	April 2010
Thesis supervisor	Senior lecturer Tony Torp
Commissioned by	Tampere University of Applied Sciences

ABSTRACT

This thesis covers the basic development of plug-ins to QtCreator-development tool. The purpose of this thesis was to get acquainted to QtCreator software, Qt-programming language and to QtCreator plug-ins.

This thesis introduces Qt-programming language and it's history, QtCreator-development tool and the basic concept of plug-in.

This thesis is hoped to encourage a fuller picture of Qt-programming to the reader. The reader is assumed however to be familiar with the C++-programming language.

The plug-in made as background research, represents Hands-on lab e-learning material made by Nokia to the programmer through the QtCreator-development tool. The plug-in will be introduced more detailed in the end of this thesis. This thesis does not concern other Qt-programming development tools, it is only focused on the QtCreator development tool. This thesis also does not address the precise method of creating a plug-in for QtCreator but covers the basics of compiling a plug-in.

Keywords Qt programming, QtCreator, plug-in, Linux, Nokia, Hands-On lab

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Alkusanat

Tämä opinnäytetyö on tehty tammikuun 2010 ja huhtikuun 2010 välisenä aikana. Aiheen valintaan vaikutti lehtori Jari Mikkolaisen ehdottama aihe ja oma kiinnostukseni Qt-sovelluskehitysympäristöä kohtaan. Kun kuulin lehtori Mikkolaiselta mahdollisuudesta toteuttaa liitännäinen jo käytössäni olevaan QtCreator-kehitystyökaluun innostuin aiheesta jo pelkästään senkin takia, että Qt-ohjelmointikielen opiskelun kynnys muille ohjelmoijille madaltuisi.

Työ antoi minulle itselleni selkeämmän käsityksen siitä kuinka QtCreator-kehitystyökalu pohjimmiltaan toimii ja kuinka siihen on mahdollista lisätä itselleen tarpeellisia osia ja oppia samalla Qt-sovelluskehitysympäristöä uudesta näkökulmasta.

Tampereella 26. maaliskuuta 2010

Kirsi-Marja Eskola

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Sisällysluettelo

Termit ja lyhenteet

Alustariippumaton	Sovellus tai koodi joka ei ole sidottu tiettyyn laitteeseen.
BSD	Berkeley Software Distribution, käytetyin avoimen lähdekoodin lisenssi.
Emacs	Tekstieditori, joka tunnetaan suuresta määrästä ominaisuuksia.
Free Software -säätiö	Säätiö on perustettu valistamaan vapaan ohjelmiston eduista.
GIMP	Avoimen lähdekoodin kuvankäsittelyohjelma.
Gitorious	Web-pohjainen avoimen lähdekoodin projektiyhteisö.
GNOME	Avoimen lähdekoodin työpöytäympäristö.
GNU GPL	General Public License on vapaa ohjelmistolisenssi.
GTK+	Graafinen käyttöliittymäkirjasto.
Intrinsics	Tunnetaan myös nimellä Xt, X-ikkunajärjestelmässä käytetty kirjasto.
KDE	Avoimen lähdekoodin työpöytäympäristö Linuxissa.
LGPL	Rajoitettu GNU GPL -lisenssi.
Linux	Avointa lähdekoodia oleva käyttöjärjestelmän ydin.
Signaalit ja lokerot	Qt:n käyttämä olioiden välinen viestinvälitystekniikka.
Mac OS X	Applen kehittämä ja markkinoima käyttöjärjestelmä.
QPL	Trolltechin avoimen lähdekoodin lisenssi.
X11	GNU/Linux-käyttöjärjestelmissä käytetty ikkunointijärjestelmä.

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

1 Johdanto

Tässä työssä käsitellään QtCreator-kehitystyökalun liitännäisen kehittämistä Linux-käyttöjärjestelmällä sekä Linux-käyttöjärjestelmälle että Windows-käyttöjärjestelmälle. Työn tarkoituksena on perehdyttää lukija Qt-sovelluskehitysympäristöön ja siitä vielä tarkemmin sen liitännäisiin.

Työssä käydään ensin luvussa yksi läpi Qt-sovelluskehitysympäristö, sen historiaa ja perusteita. Qt on norjalaisen Trolltech yhtiön lanseeraama sovelluskehitysympäristö, jonka tarkoituksena on suoraviivaistaa ohjelmointia sekä mahdollistaa alustariippumaton ohjelmointi. Luvun tarkoituksena on perehdyttää lukija siihen, minkä takia ohjelmoijat tarvitsevat Qt-sovelluskehitysympäristöä.

Perusasioihin tutustumisen jälkeen käsitellään luvussa kaksi QtCreator-kehitystyökalu. QtCreator-kehitystyökalu on toteutettu sovelluskehittäjien toiveesta tuottamaan toivottua helpotusta Qt-ohjelmointiin. Se on myös Trolltechin tuottama kokonaisvaltainen ohjelmointityökalu. QtCreator perustuu olennaisesti liitännäisiin, joten luvun tarkoitus on selventää, miksi tämä työ käsittelee liitännäiset erillisenä osiona, sekä ilmentää QtCreatorin perusrakennetta ohjelmoijan kannalta.

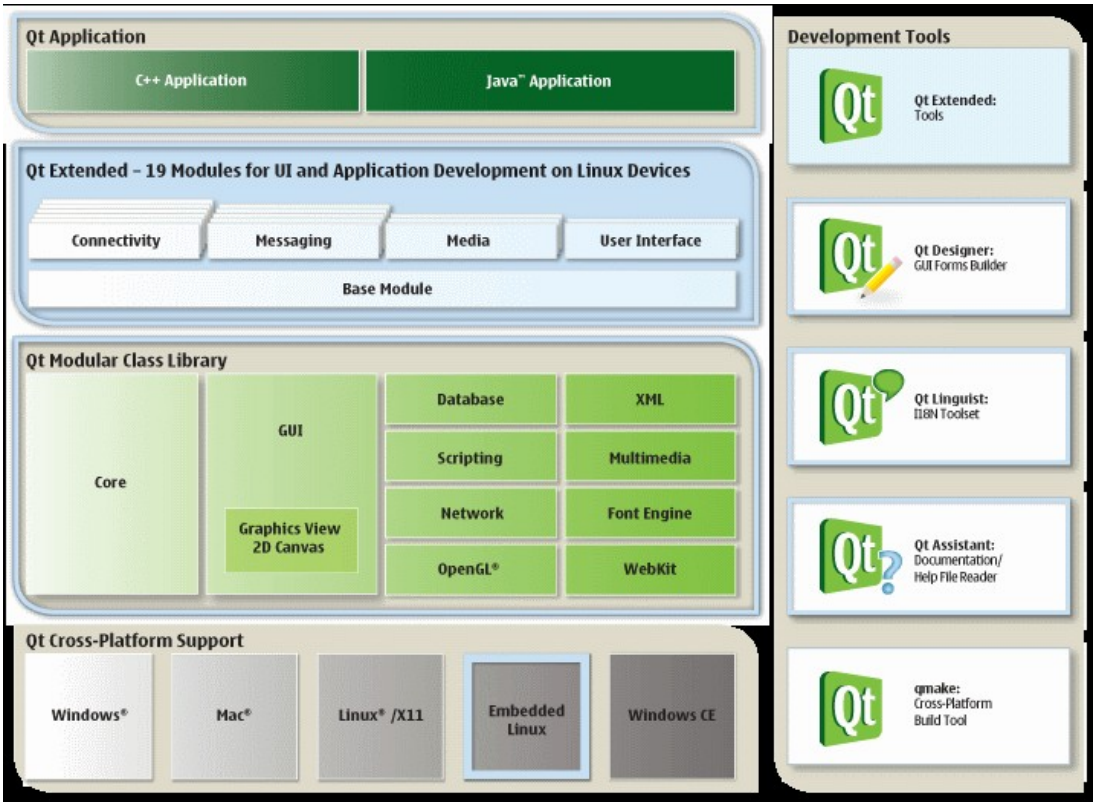
Luvussa kolme käydään läpi plug-in eli ohjelman liitännäinen käsitteenä. QtCreator-sovellus on toteutettu kokonaisuudessaan ytimen ympärille luoduista liitännäisistä, joten on hyvin luontevaa tuottaa siihen uusia tarpeellisia liitännäisiä. Luku käsittelee ensin liitännäisen yleisesti käsittäen muissakin ohjelmointiympäristöissä tuotettuja liitännäisiä ja luvun lopuksi tarkennamme liitännäisen tuottamista Qt-sovelluskehitysympäristössä.

Luvussa neljä kerrotaan mitä vaiheita liitännäisen toteuttaminen vaatii Qt-sovelluskehitysympäristössä ja miten se liitetään QtCreatoriin. Työn taustatutkimukseksi toteutettu Hands-on lab plug-in tuo Qt-ohjelmointia aloittelevan ohjelmoijan ulottuville liitännäisenä Nokian tuottaman e-oppimismateriaalin, jota on helppo seurata ohjelmointia opetellessa kehitystyökalun liitännäisenä. Työn toivotaan tuovan lukijalle selkeän käsityksen siitä, mitä Qt on ja minkälaisilla työkaluilla sitä voidaan ohjelmoida optimaalisesti.

Tämän työn tavoitteena on vastata selkeästi ja yksinkertaisesti kysymyksiin: Mikä Qt on? Mihin sitä käytetään? Mikä on liitännäinen ja miten se liittyy Qt sovelluskehitysympäristöön?

2 Qt-sovelluskehitysympäristö

Qt on alustariippumaton sovelluskehitysympäristö, joka on laajasti käytössä graafisten käyttöliittymien tuottamisessa. Alustariippumattomuus tarkoittaa yleensä sitä, että kohdelaitteena voi olla Linux-, MacOS X- tai Windows-käyttöjärjestelmän sisältävä PC-tietokone tai jokin mobiililaite. Qt:ta käytetään myös tekstipohjaisten käyttöliittymien kuten konsolityökalujen sekä palvelinsovellusten kehittämiseen. Kaikkein huomattavimmista kohteista, jotka on toteutettu Qt:lla, voisi mainita muun muassa Google Earth-palvelu, Opera-selaimen sekä internetissä toimivan Skype-puhelinsovelluksen. /1/



Kuva 1. Qt-sovelluskehitysympäristön arkkitehtuuri
(http://qt.nokia.com/images/template/product-architecture-diagram-collapsed/view)

Qt:tä kehittää nykyään Nokian Qt-kehitysosasto, joka sai alkunsa kun Nokia hankki omistukseensa norjalaisen Trolltech-yrityksen, Qt:n alkuperäisen tuottajan, 17. kesäkuuta 2008.

Qt hyödyntää C++-ohjelmointikieltä mutta käyttää laaja-alaisesti hyödyksi myös C-kieltä. Qt:tä voidaan hyödyntää myös muissa ohjelmointikielissä Qt:n kielisidosten avulla. Qt toimii kaikilla laajimmin käytössä olevilla käyttöjärjestelmillä ja sillä on myös laaja kansainvälisyystuki.

Kuva 2. Qt:llä toteutettu käyttöliittymä Fluken EtherScope-mittalaitteeseen (http://www.ferret.com.au/odin/images/165750/New-version-Fluke-Etherscope-from-TR-Corporation-165750.jpg)

2.1 Qt:n historia

Qt:n alkuperäiset kehittäjät Haavard Nord sekä Eirik Chambe-Eng Trolltech-yhtiöstä aloittivat Qt:n kehittämisen vuonna 1991. Qt sai nimensä Nordin Emacs-tekstieditorin fonttikirjastossa olevasta Q-kirjaimesta, sen näyttäessä miellyttävältä miehen silmään. T-kirjain lisättiin nimeen myös sen näyttäessä hyvältä Xt:n eli Linuxin X Window-työkalupaketin nimessä.

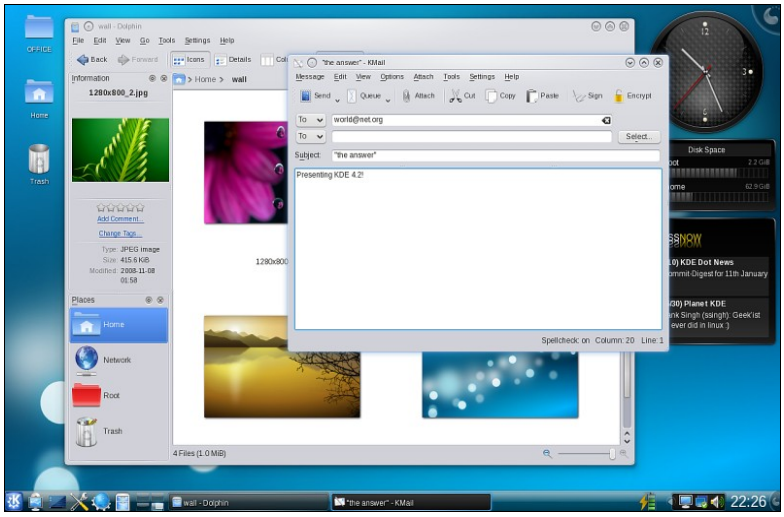
Qt:sta tehtiin ensin kaksi eri versiota, versio Linuxille ja Windowsille. Windows-versio oli saatavilla vain lisenssin alaisuudessa, mikä tarkoitti sitä, että avoimen lähdekoodin sovelluksia ei voitu siirtää Windowsille ostamatta lisenssillä varustettua Qt-versiota. Vuoden 2001 lopulla Trolltech julkaisi Qt:n version 3.0, jossa oli lisänä Mac OS X -tuki. Myös Mac OS X:n Qt-versio oli maksullisen lisenssin alainen kesäkuuhun 2003 asti, jolloin Trolltech julkaisi Qt:n version 3.2, jossa oli MAC OS X tuki.

Nokia hankki Trolltech ASA:n itselleen vuonna 2008 ja muutti sen nimen ensin Qt Softwareksi ja myöhemmin Qt Development Frameworks - nimiseksi. Sittemmin Nokia on keskittynyt kehittämään Qt:tä mobiililaitteidensa ensisijaiseksi kehitysalustaksi. Ensisijaisen kehitysalustan kehitys pitää sisällään myös Qt-tuen Symbian S60 laitteille. Nokia antoi lähdekoodin myös yleisön kehitettäväksi Gitorious-yhteisön kautta, ei vain siksi että sitä käytettäisiin, vaan myös siksi että yritys hyötyisi innokkaiden ohjelmoijien tuotoksista, jolloin myös sekä kieltä että laitetukea saataisiin parannettua.

2.2 Qt lisenssit

Jo Qt-ohjelmointikielen alusta asti Qt on ollut saatavilla kaupallisen lisenssin alla mikä on mahdollistanut sovellusten kehittämisen yksinoikeudella ilman rajoittavia tekijöitä. Kaupallisen lisenssin haltijat ovat voineet kehittää ja myydä omia sovelluksiaan ilman kielen kehittäjien vaatimuksia lopputuotteesta. Kaupallisen lisenssin rinnalla Qt:stä on vähitellen avautunut väylä kohti ilmaisia lisenssejä. Tällä hetkellä Qt onkin jo saatavilla GNU -lisenssillä, joka mahdollistaa sekä kaupallisten että ilmaisten ohjelmien tuottamisen.

Versioon 1.45 asti Qt:n lähdekoodin sai FreeQt lisenssin alla, joka ei kuitenkaan sopinut Open Source Initiativen periaatteeseen. Ristiriitaa asiasta syntyi vuonna 1998, kun selvisi, että KDE:stä, Linux-käyttöjärjestelmän Qt:llä tuotetusta K Desktop Environmentista, olisi tulossa yksi johtavista työpöytäympäristöistä ja että olennainen osa yhdestä heidän suurimmista ilmaisen lähdekoodin käyttöjärjestelmistään olisi yksinoikeudella tuotettu.



Kuva 3. KDE työpöytäympäristö, toteutettu Qt:lla
(http://qt.nokia.com/images/qt-in-use/opensource-showcase/desktop.png*)*

Version 2.0 myötä lisenssi muutettiin Q Public -lisenssiksi (QPL), joka oli edelleen Free Software -säätiön mukaan ristiriitainen ja johtikin loppujen lopuksi KDE Free Qt -säätiön syntyyn. Se takasi, että Qt lisensioitaisiin Berkeley Software License -vapaan ohjelmistolisenssin ohjelmistoksi. BSD-lisensiointi johti kahteen eri projektiin: Harmony-työkaluun, joka etsi mahdollisuutta tehdä Qt:stä ilmaisen lisenssin version kaupallisen rinnalle, sekä GNOME-työpöytäsovellukseen, jonka tarkoituksena oli korvata KDE kokonaan.

Trolltech julkaisi Qt:n version 4.0 Windowsille GPL lisenssin alaisuuteen vuonna 2005. Qt 4 tukee samoja alustoja kuin kaupallinen versio, joten ohjelmoijien on mahdollista tuottaa avoimen lähdekoodin sovelluksia jokaisella tuetulla alustalla. Tammikuussa 2009 julkaistu Qt versio 4.5 lisäsi taas uuden lisenssivaihtoehdon; LGPL Lesser General Public-lisenssin, tarkoituksenaan tehdä Qt:stä entistä houkuttelevampi ei-GPL -lähdekoodin projekteille ja suljetuille sovelluksille.

2.3 Qt ohjelmointikielenä

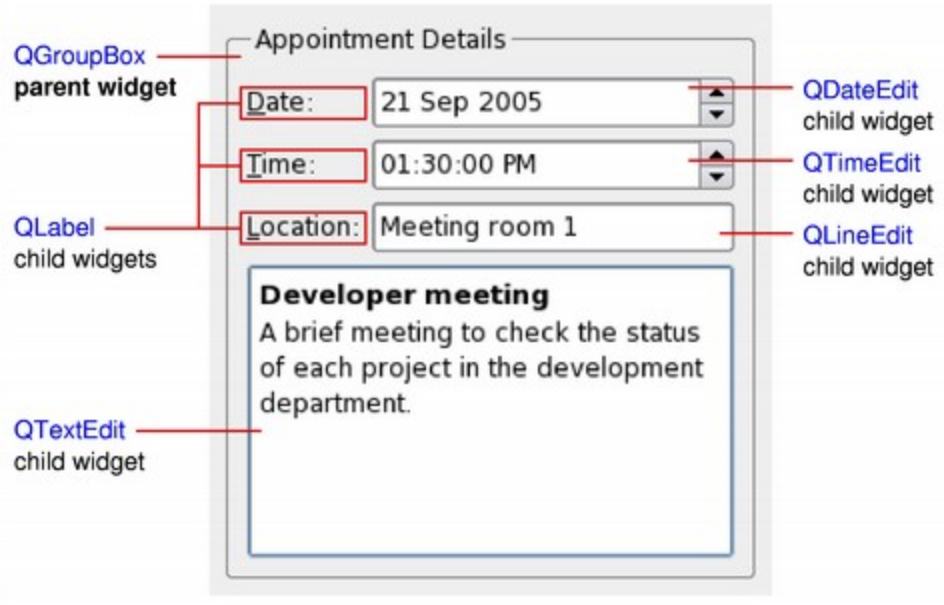
Qt on alustariippumaton sovelluskehitysympäristö, mikä tarkoittaa sitä että, kun ohjelmoija kirjoittaa yhden koodin, niin sen voi kääntää suoraan jokaiselle Qt:tä tukevalle laitteelle. Qt on luonut maineensa alustariippumattomana kehyksenä, mutta sen tehokkaan ohjelmointirajapinnan ansiosta monet yritykset käyttävät sitä yhden kohdelaitteen sovelluskehitykseen, esimerkiksi Adobe Photoshop Album on yksi massatuotantosovellus, joka on luotu käyttämällä Qt:tä ja on tarjolla vain Windows-käyttöjärjestelmän käyttäjille. /2/

Qt sisältää sarjan ”widgetejä” eli kontrolleja, jotka tarjoavat sovelluksen perusgraafisen käyttöliittymän toiminnallisuuden, muun muassa painikkeet ja tekstilaatikot. Qt tarjoaa myös innovatiivisen vaihtoehdon objektien väliselle kommunikoinnille: signaalit ja lokerot. Signaalit ja lokerot korvaavat aiemman turvattoman Takaisin kutsu -menetelmän. Qt sisältää myös paljon moderneja sovellusosia kuten valikot, Raahaa ja pudota-toiminnon, sekä lukittavat työkalupalkit. /2/

2.3.1 Qt GUI

Widgetit ovat Qt:n visuaalisia elementtejä, joita yhdistettäessä muodostuu graafisia käyttöliittymiä. Ne voivat olla siis painikkeita, valikkoja, viestilaatikoita, sovellusikkunoita ynnä muita tarvittavia objekteja. Widgeteihin kuuluu Qt-ohjelmointikielessä myös sovelluksen pohjapiirustukset eli layoutit.

Layoutit järjestelevät ”lapsi”-widgetit ”vanhempi”-widgetin sisällä. Layoutit suorittavat automaattisesti objektien järjestämisen paikoilleen ja niiden koon muuttamisen. Layoutit mahdollistavat myös ylemmän tason widgetien järkevän koon muuttamisen, esimerkiksi sovellusikkunan kokoa muuttaessa. ”Vanhempi”-widgetit ovat yleensä dialogeja eli sovellusikkunoita, jotka sisältävät n-määrän ”lapsi”-widgetejä.



Kuva 4. Vanhempi-widget ja lapsi-widgetit sijoitettuna layout-järjestykseen. (http://doc.trolltech.com/4.6/images/parent-child-widgets.png)

Esimerkki dialogin käytöstä Nokian Qt Reference Documentation www-sivuilta:

```
class RotateDialog : public QDialog
{
    Q_OBJECT
public:
    RotateDialog( QWidget *parent=0,
                const char *name=0 );
    void resizeEvent( QResizeEvent * );
private slots:
    void updateCaption();
private:
    QPushButton   *quit;
    QPushButton   *load;
    QPushButton   *print;
    QScrollBar    *scrollBar;
    QFrame        *frame;
    PixmapRotator *rotator;
};
```

RotateDialog-luokka perii QDialog-luokan ja sisältää 3 painiketta, vierityspalkin, kehyksen ja räätälöidyn pixmap-objekti -kiertäjän. Dialogilla on vain kolme jäsenfunktiota. Rakentaja alustaa eri widgetit dialogissa, resizeEvent asettaa widgetien sijainnin ja koon ja lokero päivittää dialogin otsikkotekstin.

Esimerkki painikkeen alustuksesta rakentajassa:

```
quit = new QPushButton("Quit", this);
quit->setFont(QFont("Times", 14,QFont::Bold));
connect( quit, SIGNAL(clicked()), qApp,SLOT(quit()) );
```

Ensin alustetaan painike quit parametreilla: ”Quit” määrittelee painikkeen tekstin ja this-komento kertoo napin vanhemman eli sen mihin painike sijoitetaan, tässä tapauksessa tähän samaiseen QDialogiin. Sitten quit painike saa fonttimäärittelyt ja viimeiseksi quit-painikkeen painettu-signaalille asetetaan lokero, joka lopettaa sovelluksen suorittamisen.

2.3.2 Signaalit ja lokerot

Signals and slots eli signaalit ja lokerot yhdistävät sovelluksen eri komponentit yhteen niin, että ne voivat kommunikoida helpolla ja luotettavalla tavalla. Signaalit ja lokerot vastaavat niin sanottua tarkkailijamallia ja toteuttavat tietyn palvelun ilman saman koodinpätkän toistamista useaan otteeseen monessa eri paikassa.

Observer- eli tarkkailijamalli on ohjelmistosuunnittelussa käytössä oleva malli, jossa objektit hallitsevat riippuvuuksia. Riippuvuuksia hallitsevia objekteja kutsutaan observereiksi eli tarkkailijoiksi. Tarkkailija ilmoittaa objekteille automaattisesti tilamuutoksesta yleensä kutsumalla jotakin metodia.

Signaalien ja lokeroiden perusajatus on, että widgetit voivat lähettää signaaleita, jotka sisältävät jonkun tietyn tapahtuman tietoa ja tämä tieto vastaanotetaan toiminnoissa, joita kutsutaan lokeroiksi.

Signaalit ja lokerot sopivat hyvin graafisten käyttöliittymien toimintoihin kuin myös I/O-prosesseihin, jotka vaativat että yksi toiminto on suoritettu ennen kuin toinen aloitetaan. Signaaleita ja lokeroita voidaan käyttää myös ajastimien kanssa. Kun ajastin laukeaa, lähetetään signaali tietylle lokerolle, joka toteuttaa tietyn toiminnon.

Kaikki QObject-luokasta perivät widgetit voivat sisältää signaaleita ja lokeroita. Signaali lähetetään kun sen omistavan objektin tila muuttuu. Tämä signaali saattaa olla jonkun lokeron mielestä on kiinnostava. Lokerot eivät välttämättä tiedä, mistä objektista signaali on tuotettu mutta juuri tämä ominaisuus takaakin turvallisen ja luotettavan viestijärjestelmän toimivuuden.

Lokerot voivat vastaanottaa signaaleita mutta ovat myös täysin toimivia funktioita. Signaaleita voidaan kytkeä niin moneen paikkaan kuin tarve vaatii ja lokerot voivat vastaanottaa niin monta signaalia kuin tarve vaatii, jopa signaalin kytkeminen signaaliin on mahdollista. Signaaleita ja lokeroita käytettäessä ei tarvitse kirjoittaa minkään näköistä rekisteröintikoodia, koska Qt:n metaobjektikääntäjä MOC tuottaa automaattisesti rekisteröinnin rakenteen.

Esimerkki signaalien ja lokeroiden käytöstä Nokian Qt Reference Documentation www -sivuilta:

Tiedostossa .h esitellään signal- ja slot-funktiot:

```
public slots:
  void setValue(int value);

signals:
  void valueChanged(int newValue);
```

Tiedostossa .cpp luodaan slot funktion sisältö:

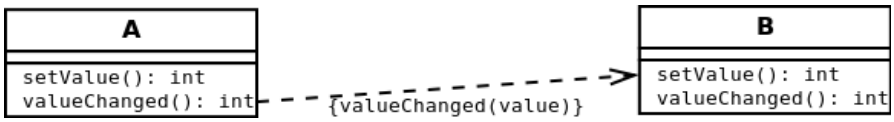
```
void Counter::setValue(int value)
{
   if (value != m_value) {
   m_value = value;
   emit valueChanged(value);
   }
}
```

Signaali kytketään lokeroon komennolla:

```
Qobject::connect(&a,
                 SIGNAL(valueChanged(int)),
                 &b,
                 SLOT(setValue(int)));
```

Signaali lähetetään siihen kytketyn muuttujan a arvon muuttuessa:

```
a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```



Kuva 5. Signaali kytkettynä lokeroon

2.3.3 Meta Object Compiler

MOC eli Meta Object Compiler on työkalu jota käytetään jokaisen Qt -sovelluksen taustalla. Se tulkitsee tiettyjä makroja C++-koodista huomautuksina tai selityksinä ja käyttää niitä luodakseen vaihtoehtoista C++-koodia, joka sisältää metatietoa sovelluksen käyttämistä luokista. MOC-työkalu lukee C++-otsikkotiedoston. Jos se löytää yhden tai useamman luokkamäärittelyn, joka sisältää Q_OBJECT makron, se tuottaa C++-lähdetiedoston, johon kuuluvat metaobjektit koodista löydetyille luokille. /5/

MOC:sta saatua metatietoa käytetään Qt:ssä tuottamaan sellaisia ohjelmointipiirteitä, jotka eivät alun perin ole saatavilla C++-ohjelmointikielessä, kuten signaalit ja lokerot, itsehavainnointi sekä asynkroniset funktiokutsut. /1/

Esimerkki Q_OBJECT-makron käytöstä ja siitä, miten MOC sen löytää:

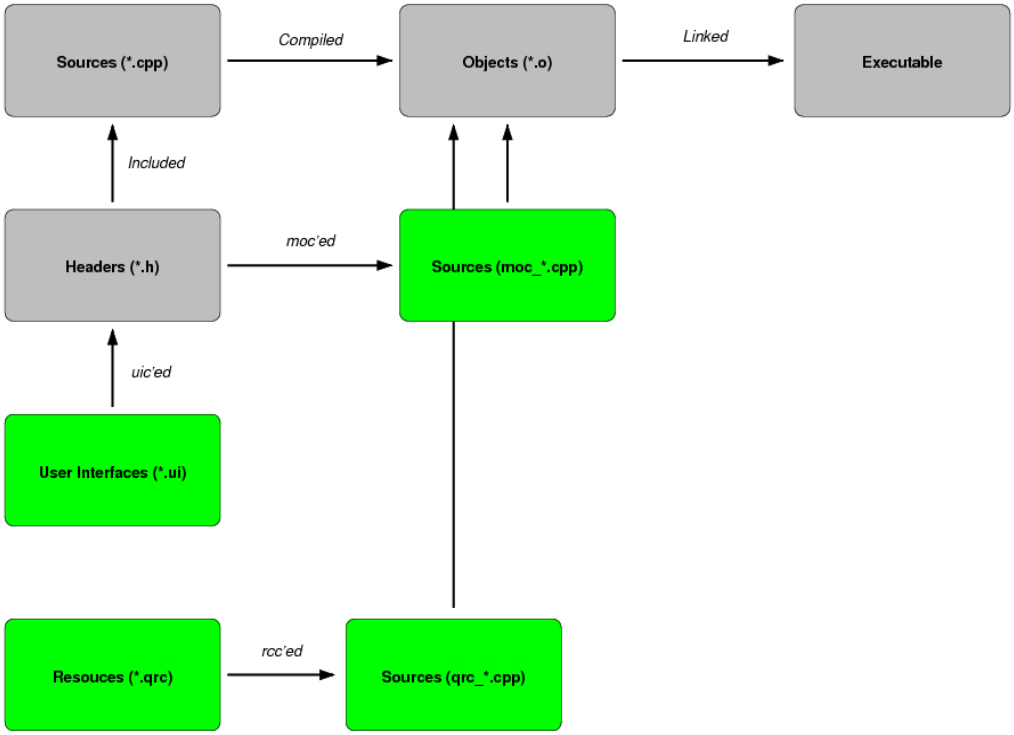
```
class MyClass : public QObject
    {
        Q_OBJECT

    public:
        MyClass(QObject *parent = 0);
        ~MyClass();

    signals:
        void mySignal();

    public slots:
        void mySlot();
    };
```

MOC lukee tämänkaltaisen otsikkotiedoston ja löytää Q_OBJECT-määrittelyn, minkä seurauksena se luo .moc-tiedoston, jossa on tämän luokan metamäärittelyitä.



Kuva 6. Kaavio MOC-toiminnoista verrattuna normaaliin C++-kääntämiseen. Normaalit C++-toiminnot on kuvattuna harmaalla ja Qt:n lisätoiminnot vihreällä. (http://thelins.se/learnqt/wp-content/uploads/qt-buildsystem.png)

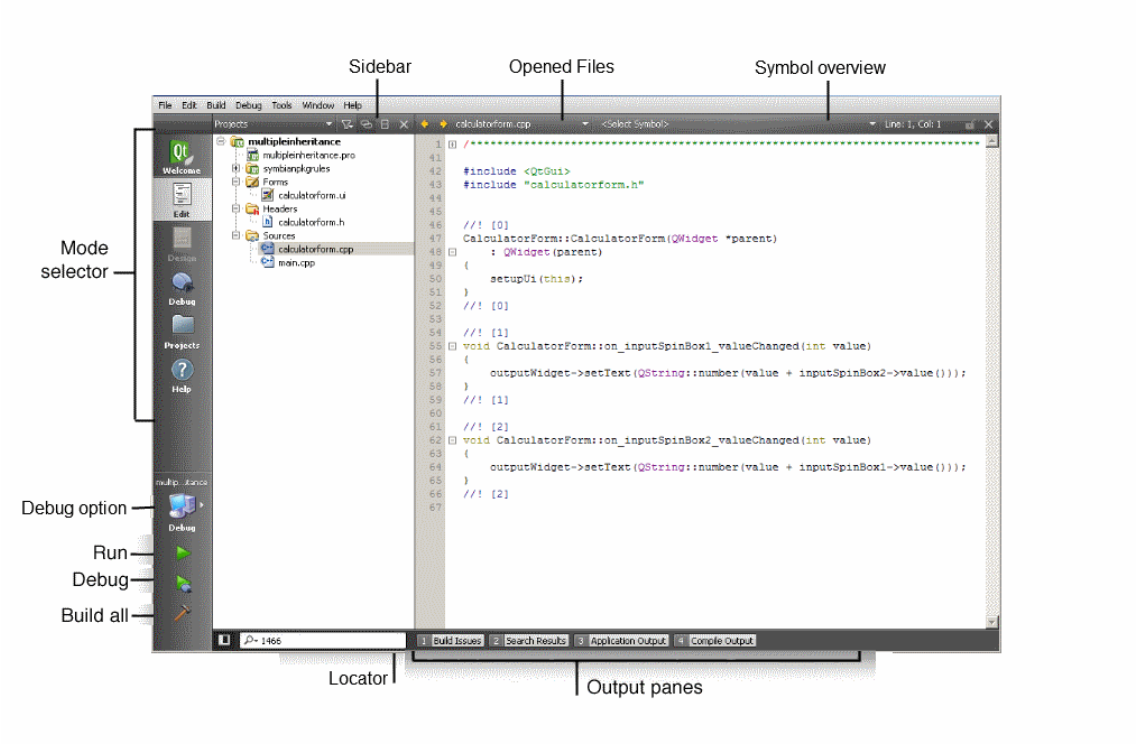
3 QtCreator-kehitystyökalu

Jokaisella isolla järjestelmällä on hyvin määritelty järjestelmäarkkitehtuuri joka oikein ymmärrettynä selventää järjestelmän sisäistä rakennetta. QtCreator ei eroa muista järjestelmistä tässä suhteessa mitenkään. Tässä kappaleessa käymme läpi QtCreatorin perusarkkitehtuurin, joka auttaa ymmärtämään lisää siihen ohjelmoitavista liitännäisistä. /2/

QtCreator on alustariippumaton kehitysympäristö, johon on integroituna C++-ohjelmointikieli. QtCreator pitää sisällään visuaalisen debuggerin eli työkalun jota käytetään testaamaan ohjelmakoodia ja sen toimivuutta. QtCreatoriin on myös integroituna graafisia käyttöliittymäkomponentteja. QtCreator käyttää C++-kääntäjää Linuxin GNU kääntäjä -kokoelmasta, Windowsissa käytössä on MinGW:n oletusasennus.

QtCreator tarjoaa integroituja työkaluja sovellusten suunnittelijoille ja kehittäjille. Sillä on mahdollista tuottaa sovelluksia usealle käyttöjärjestelmälle ja kohdelaitteelle riippumatta siitä millä alustalla itse QtCreator suoritetaan.

Ohjelmistosuunnittelijoille QtCreator tarjoaa kaksi visuaalista elementtiä, QtDesigner ja Qt Quick Designer, joita suunnittelija voi käyttää graafisten käyttöliittymien luomiseen. QtCreatorista on saatavilla kokonaisvaltainen integroitu kehitysympäristö IDE, joka sisältää kaiken tarvittavan Qt-sovellusten luomiseen. IDE on saatavilla Linux-, Mac OS X- ja Windows-käyttöjärjestelmille./5/



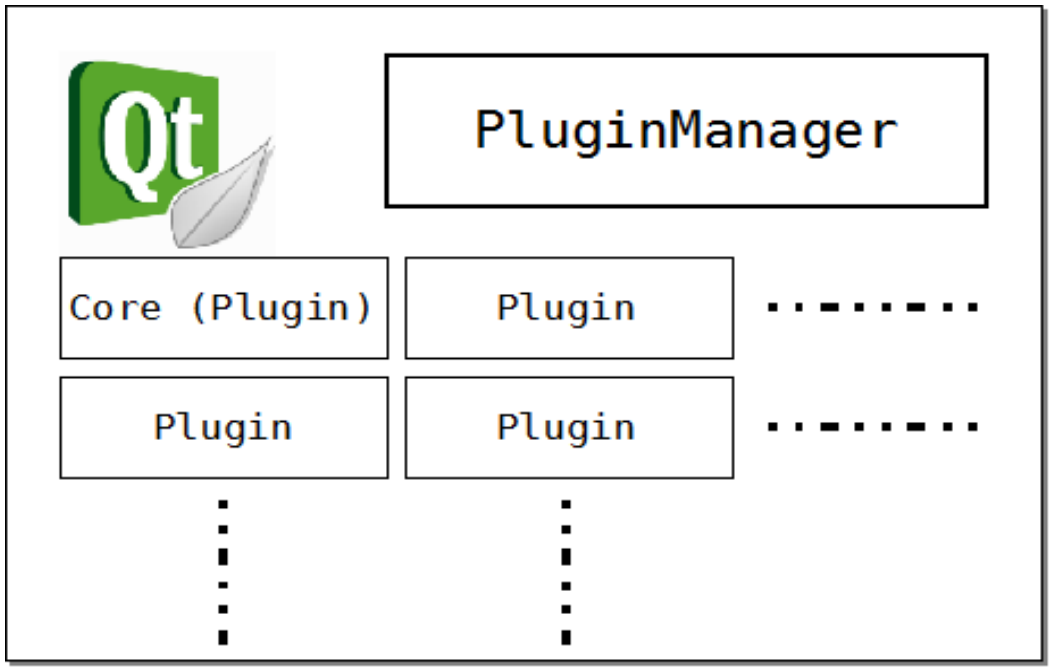
Kuva 7. QtCreator-kehitystyökalun komponentit ja ulkonäkö (http://qt.nokia.com/doc/qtcreator-snapshot/images/qtcreator-context-sensitive-help.png)

3.2 QtCreatorin arkkitehtuuri

QtCreatorin ydin on oikeastaan vain liitännäisten lataaja. Sovelluksen kaikki toiminnot suoritetaan erillisissä liitännäisissä. QtCreatorin toiminnan ”ydinajatus” on tuotettu ydinliitännäisessä: Core::ICore. QtCore-moduuli sisältää sovelluksen kaiken sisäisen toiminnallisuuden. Kaikki muut moduulit/liitännäiset perustuvat QtCore-moduuliin. Liitännäisen linkittäminen ydinprosessiin määritellään .pro-määrittelytiedostossa seuraavasti:

```
QT -= gui
```

Liitännäisten toimintaa valvova ExtensionSystem::PluginManager hallitsee liitännäisten lataamiset. PluginManager tuottaa yksinkertaisen keinon liitännäisten yhteiselolle, joissa toiset liitännäiset tuottavat palveluita toisille liitännäisille ja liitännäisten jatkeille. PluginManager perii luokan QObject, joten se on näin ollen myös widget ja siihen on mahdollista liittää signaali ja lokero. /6/

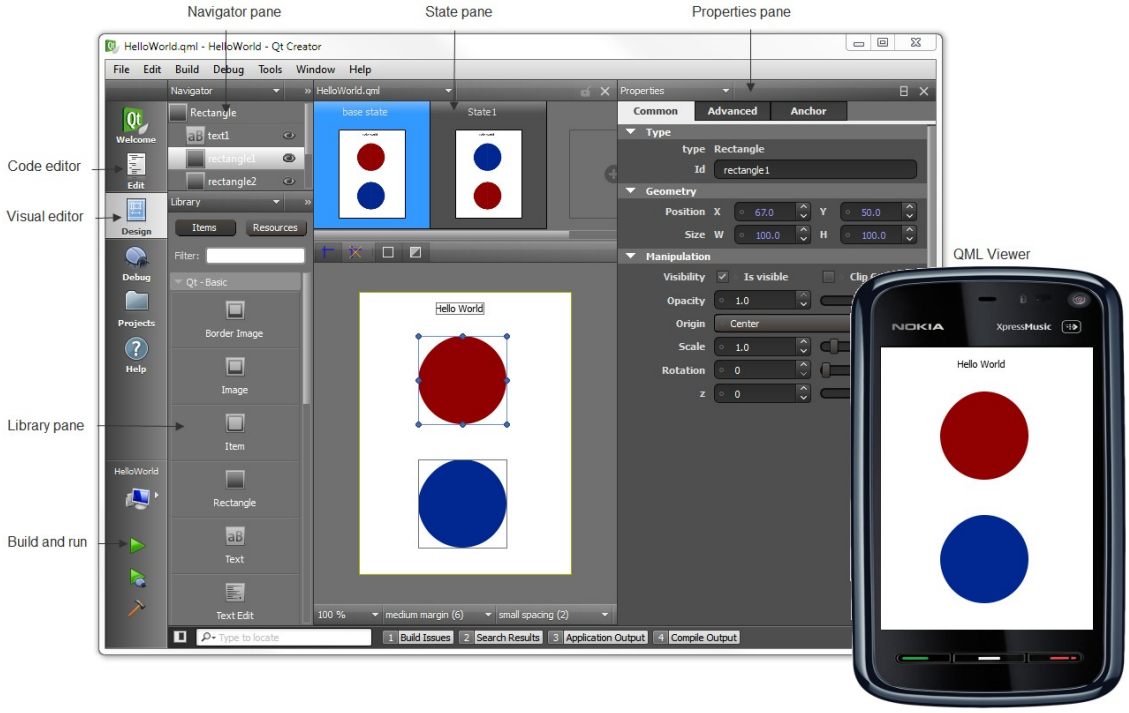


Kuva 8. QtCreatorin arkkitehtuurin perusajatus
(/6/ Writing QtCreator Plugins)

3.2 QtDesigner

QtDesigner on työkalu graafisten käyttöliittymien luomiseen Qt:n widgeteistä. QtDesigner on osa QtCreator kokonaisuutta ja sisältää visuaalisen Raahaa-pudota tyylisen käyttöliittymäkomponenttien luomistoiminnon. QtDesignerillä luodut widgetit voidaan saumattomasti liittää ohjelmoituun koodiin käyttämällä signaaleita ja lokeroita. /5/

Kaikkia QtDesignerissä luotuja ominaisuuksia voidaan dynaamisesti muuttaa ohjelmakoodista käsin. QtDesigner on, kuten muutkin QtCreatorin ominaisuudet, liitännäinen johon on helppo lisätä omia tarvittavia toimintoja. QtDesigner luo käyttöliittymän tarvitseman koodipohjan automaattisesti./5/



Kuva 9. QtDesigner QtCreatorissa, esimerkkinä näkymä Nokian mobiililaitteeseen. (http://qt.nokia.com/doc/qtcreator-snapshot/images/qmldesigner-visual-editor.png)

QtDesignerissä luodut komponentit eroavat koodissa luoduista komponenteista ui.-etuliitteensä ansiosta. Esimerkiksi koodissa luotu tekstialue:

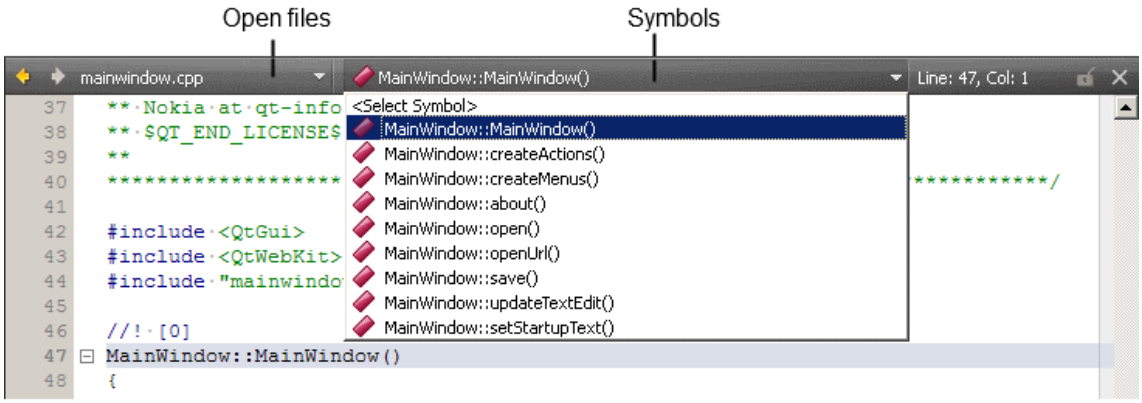
```
QTextBox textbox  = new QTextBox();
textbox->setText(”esimerkki koodin tekstialueesta”);
```

QtDesignerissä luotu tekstialue ei vaadi koodissa luomista vaan siihen voidaan liittää toiminto koodin sopivassa kohtaa näin:

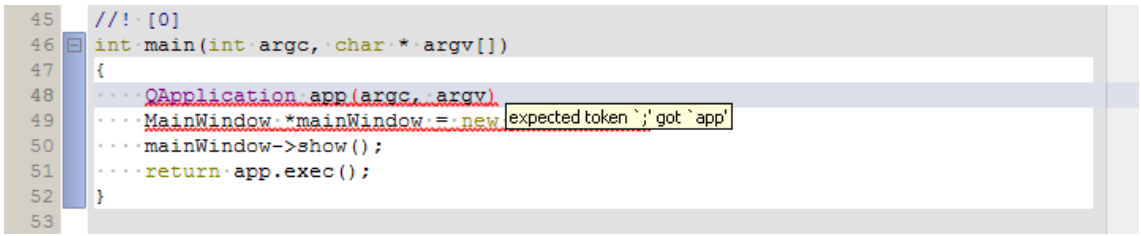
```
ui.textbox->setText(”esimerkki designerin tekstialueesta”);
```

3.3 QtCreator editori

QtCreatorin editori on suunniteltu helpottamaan ohjelmoijaa koodin luomisessa, etsimisessä ja muokkaamisessa. Editori sisältää syntaksin tarkistus -toiminnon, sisältökohtaisen avustustoiminnon ja rivi kohtaisen virheentarkistustoiminnon, joka tarkistaa virheitä myös kirjottaessa. QtCreatorin huomatessa virhe koodissa, se näyttää virheen tiedot, kun hiiren kursori on sijoitettu virheellistä koodia sisältävän rivin päälle. /5/

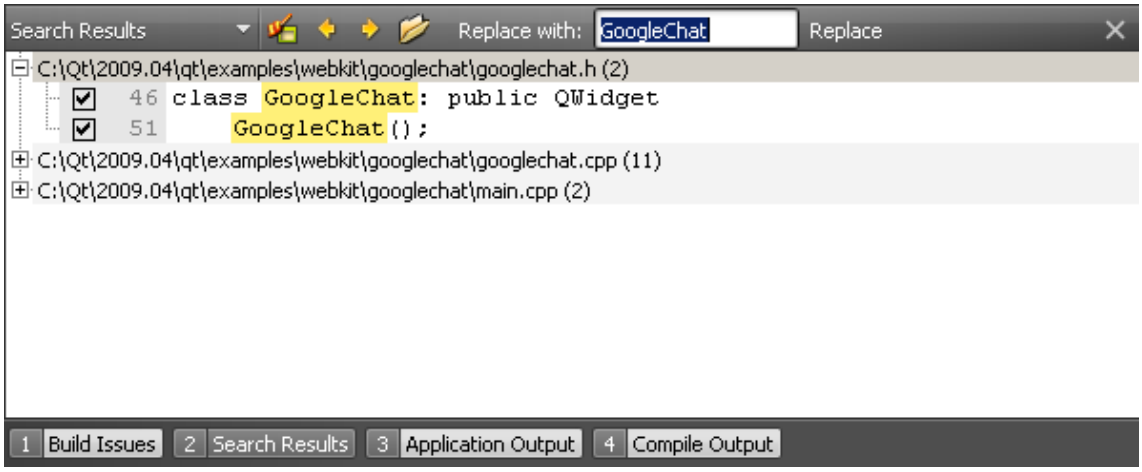


Kuva 10. QtCreatorin editor osa.
(http://qt.nokia.com/doc/qtcreator-snapshot/images/qtcreator-editortoolbar-symbols.png)



Kuva 11. Editorin virheenkorjaustoiminto, ko. kohdassa puuttuu puolipiste rivin lopusta.
(http://qt.nokia.com/doc/qtcreator-snapshot/images/qtcreator-syntaxerror.png)

Editorin ominaisuuksiin kuuluu myös koodin refaktorointi. Refaktoroinnissa muutokset koodiin on mahdollista tehdä ilman olemassa olevien toiminnallisuuksien muutoksia. Sillä voidaan saavuttaa koodin sisäisen rakenteen paranemista, suorituksen paranemista, koodin luettavuuden paranemista sekä koodirakenteen yksinkertaistumista./5/



Kuva 12. Refaktorointi editorissa.
(http://qt.nokia.com/doc/qtcreator-snapshot/images/qtcreator-refactoring-replace.png)

QtDesignerissä luodut graafiset komponentit ovat täysin muokattavissa editorissa kuten aikaisemmassa esimerkissä käsitelty tekstialue.

4 Liitännäinen

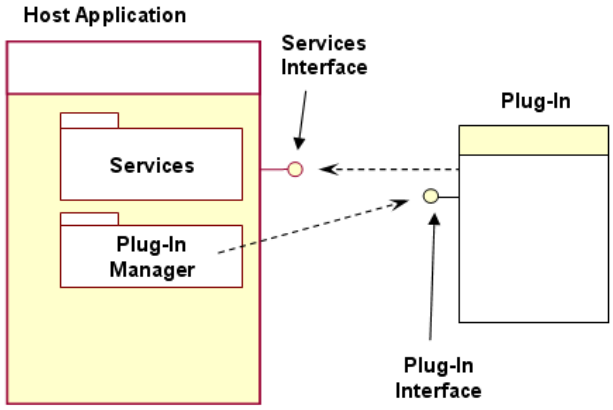
Liitännäinen on sovellukseen asennettava lisäosa, joka tuottaa käyttäjälle uusia ominaisuuksia sovellukseen. Se on laajennus tietokoneohjelmaan, jonka avulla alkuperäisen ohjelman toiminnallisuus paranee tai laajenee. Yleensä ohjelmassa on valmis rajapinta, jonka avulla liitännäisten toteuttaminen onnistuu. Tavallisesti liitännäisiä saa mediasoittimiin, Internet-selaimiin ja kuvankäsittelyohjelmiin. Niiden avulla saadaan mm. tuettua useampia tiedostoformaatteja, näytettyä visualisointeja ja muokattua toistettavaa informaatiota. /4/

Tämä kappale käsittelee liitännäisen käsitteenä, miten liitännäinen rakentuu ja liittyy isäntäsovellukseen sekä liitännäisen Qt:n ja QtCreatorin näkökulmasta.

4.1 Liitännäinen käsitteenä

Ensimmäiset liitännäiset alkoivat ilmestyä tietokonemaailmaan 1970-luvun puolessa välissä, kun Unisys VS/9 -käyttöjärjestelmän tekstieditorista EDT:stä oli mahdollista käsitellä sovellusta ja antaa sovelluksen käyttää puskurimuistiaan.

Perustasolla ajateltuna liitännäiset ovat joukko jaettuja kirjastoja, jotka lisäävät ylimääräisiä toimintoja tiettyyn käyttöliittymään. Liitännäiset keskustelevat isäntäsovelluksen kanssa ja tuottavat tiettyjä toimintoja tarvittaessa. Isäntäsovellus tarjoaa palveluja, joita liitännäiset voivat käyttää kuten menetelmän, jonka avulla liitännäinen voi rekisteröidä itsensä isäntäsovelluksen käyttöön, sekä protokollan, jolla tietoja siirretään isäntäsovelluksen ja liitännäisen välillä. /4/



Kuva 13. Liitännäisen liittyminen isäntäsovellukseen.
(http://wpcontent.answers.com/wikipedia/en/5/46/Plug-InExample.png)

”Liitännäiset ovat riippuvaisia palveluista, joita isäntäsovellus tarjoaa, eivätkä yleensä toimi itsenäisinä ohjelmina. Kääntäen isäntäsovellus on riippumaton liitännäisistä, joten liitännäisiä voidaan lisätä ja päivittää ilman muutoksia isäntäsovellukseen. Joissakin tapauksissa isäntäsovellusta ei tarvitse edes käynnistää uudelleen liitännäisen asentamisen jälkeen. ”/4/

Tavallisesti liitännäiset ovat toimintoja, jotka lukevat tai muokkaavat tietyn tyyppisiä tiedostoja (esimerkiksi purkavat ääni- tai videotiedostoja), salaavat tai purkavat sähköpostiviestejä (esimerkiksi PGP), käsittelevät kuvia grafiikkaohjelmassa tavoilla, joihin isäntäohjelma ei ilman liitännäistä pysty ja toistavat multimediaesityksiä WWW-selaimessa (esimerkiksi Adobe Flash). /4/

Mozilla Corporationin www-sivuilla liitännäinen www-selaimille on määritelty näin: *”Liitännäisten avulla selaimet kykenevät näyttämään tiettyjä kuva- tai mediatiedostoja, joita ne eivät muuten voisi näyttää. Liitännäiset ovat kuitenkin eri asia kuin laajennukset, jotka nekin lisäävät tai muuttavat selaimen toimintaa.”* /3/

Liitännäiset ovat hyvin todennäköisesti levinneet lähes jokaiseen käyttämäämme sovellukseen. Onko tämä sovelluskehittäjien valinta helpottaa omaa työtaakkaansa vai onko kuluttajista tullut entistä vaativampia käyttämiensä sovellusten suhteen?

4.2 Liitännäinen Qt-kehitysympäristössä

Qt:n toimintoja itsessään on mahdollista laajentaa liitännäisten avulla, mutta niin myös Qt:llä toteutettuja sovelluksia. Qt:n yleisimmin käytettävät liitännäiset ovat muun muassa tietokanta-ajureita, kuvaformaatteja, tekstikoodauksia sekä tyylejä.

Sovelluksille tarkoitetut liitännäiset toteutetaan QPluginLoader luokan avulla. Luodessa liitännäistä omaan sovellukseen täytyy projektiin liittää mukaan luokka QPluginLoader, joka hoitaa liitännäisen latauksen ohjelman suorituksen alussa. Se tarkistaa, että liitännäinen on linkitetty samaan Qt versioon kuin sovellus, johon liitännäinen on luotu. /5/

Liitännäisen kirjoittaminen vaatii sopivan liitännäispohjan määrittelyn, muutaman toiminnallisuuden toteuttamisen ja makron lisäämisen. Qt:ssä on useita liitännäisten perusluokkia. Perittävät liitännäiset sijaitsevat oletusarvoisesti plugins-kansiossa.

Kun sovellus suoritetaan, Qt käsittelee sovelluksen kantakansiota liitännäisen etsimiseen. Qt etsii liitännäisiä myös QlibraryInfo::location(QLibraryInfo::PluginsPath) komennon sisältämästä sijainnista, joka tyypillisesti sijaitsee Qt:n asennuskansiossa: `QTDIR/plugins`. Jos tarve vaatii, voidaan määritellä niin monta sijaintia kuin halutaan kutsumalla funktiota QCoreApplication::addLibraryPath(). Jos sijainti tarvitsee määritellä tarkasti, kutsutaan funktiota QCoreApplication::setLibraryPaths(). /5/

Qt sovellukset tietävät automaattisesti mitä eri liitännäisiä on saatavilla, jos liitännäiset sijaitsevat oletushakemistossa. Tätä menetelmää käyttämällä sovelluksiin ei tarvitse määritellä millään tavalla missä liitännäiset sijaitsevat Qt:n löytäessä ja suorittaessa ne automaattisesti./5/

Liitännäisen kirjoittamiseen kuuluu seuraavat vaiheet:

- Liitännäiseen on määriteltävä perittävä QObject-luokka ja käytettävät rajapinnat.
- MOC:lle on kerrottava rajapinnoista Q_INTERFACES-makrolla.
- Liitännäinen toteutetaan käyttämällä Q_EXPORT_PLUGIN-makroa.
- Liitännäisen kääntämiseen vaaditaan sopivan .pro-määrittelytiedoston käyttöä.

4.3 QtCreator ja liitännäinen

QtCreatorin liitännäiset ovat täysin vastaavia Qt:n muihin liitännäisiin. Liitännäinen suoritetaan liitännäiselle kuuluvasta kirjastosta QtCreatorin käynnistyessä vastaavalla tavalla, kuin jos liitännäinen olisi luotu johonkin ”tavalliseen” Qt-sovellukseen.

QtCreatorin ydinprosessi luo liitännäisen kirjastotiedostoista liitännäis-widgetin käyttämät perusosat QPluginLoaderin avulla. MOC luo tavallisen widgetin tavoin liitännäis-widgetistä metatietoa, jota QtCreator käyttää hyödyksi objektien ominaisuuksia listattaessa.

Liitännäisiä ohjelmoitaessa QtCreatoriin, on tärkeää varmistaa, että liitännäiset ovat määritelty samoin kuin sovellus itsessään. Jos sovellus on käännetty release-muodossa, vain release-muotoisen liitännäiset ovat yhteensopivia kyseisen sovelluksen kanssa. Liitännäisen määrittelytiedoston yksi ainoa rivi määrittelee muodon mihin liitännäinen käännetään:

```
CONFIG += release
```

Kun määrittelytiedostoon on sisällytetty tämä rivi, liitännäisen pitäisi olla täysin yhteensopiva sen QtCreator version kanssa mihin se on ohjelmoitu, millä alustalla tahansa.

Kun QtCreator käynnistyessään luo liitännäiset, se tarkistaa jokaisen liitännäisen yhteensopivuusavaimen ja vain yhteensopivat liitännäiset käynnistetään, kaikki liitännäiset joiden yhteensopivuusavain on erilainen kuin QtCreatorin oma, jätetään suorittamatta./5/

Yhteensopivuusavain sisältää muun muassa seuraavia asioita:

- käyttöjärjestelmän sekä kääntäjän arkkitehtuurin
  * siltä varalta, että kääntäjän eri versiot eivät ole yhteensopivia toistensa kanssa
- Qt-kirjaston määrittelyn, lista puuttuvista ominaisuuksista jotka saattavat vaikuttaa toimintaan
  * kaksi eri kokoonpanoa samasta Qt-versiosta eivät ole binäärisesti yhteensopivia.
- Tarvittaessa erillinen merkkijono voidaan käsitellä komentorivillä
  * tarjoaa sovelluskehittäjille mahdollisuuden luoda liitännäisiä, jotka voidaan suorittaa vain siihen linkitetyn kirjaston kanssa.

Debuggauksen yhteydessä nämä tarkistukset on mahdollista ohittaa määrittelemällä Qt:n esikäsittelymakro `QT_NO_PLUGIN_CHECK`. /5/

5 Nokia Hands-on lab QtCreator plug-in

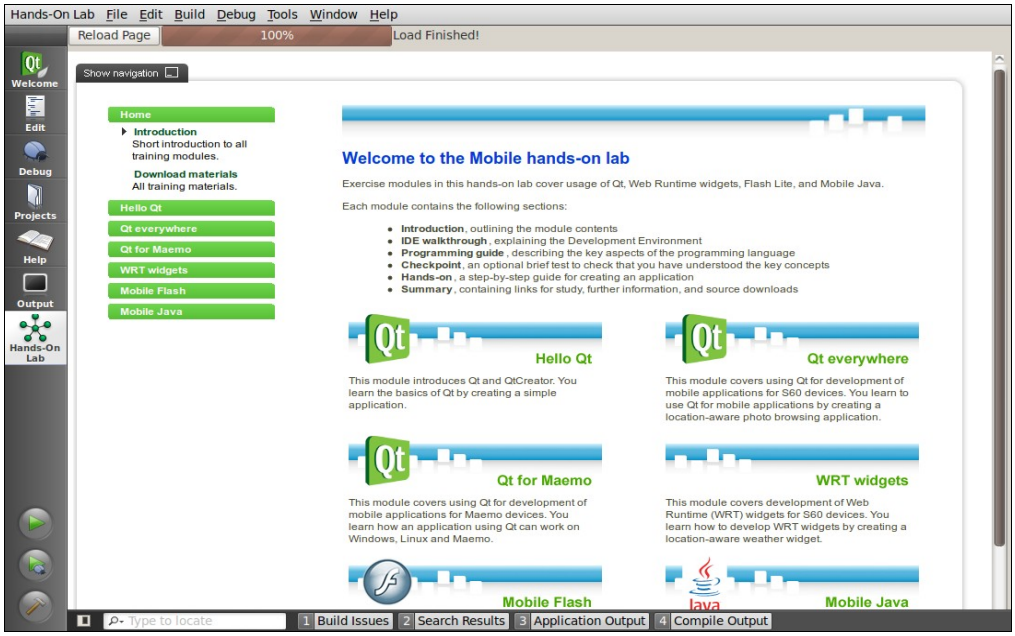
Hands-OnPlugin-liitännäisen perusajatuksena on tuottaa Nokian Qt e-oppimateriaali Qt:ta aloittelevan ohjelmoijan saataville, helposti käytettävällä tavalla.

QtCreatorin rakenteen ansiosta, ulkopuolisen tuottajan on helppo lisätä siihen omia osia liitännäisten muodossa. Liitännäisten rakenteesta löytyy todella vähän tietoa, joten niiden oppiminen tapahtuu parhaiten tekemällä. Qt:n Reference Documentation www-sivulla on maininta liitännäisten liittämisestä omiin sovelluksiin sekä QtDesigneriin. QtCreator liitännäisistä ei siis löydy Qt:n dokumenteista mainintaa, mutta QtCreatorin rakentuessa täysin samalla tavalla kuin muutkin Qt-sovellukset, voidaan näitä ohjeita hyödyntää myös tämän liitännäisen luomisessa.

Liitännäisten tekemisessä tarvitsee huomioida muutama eroavaisuus suhteessa tavalliseen Qt-sovellukseen. Liitännäiselle määritellään ohjausmäärittelyt ja liitännäisen sijainti .pro- sekä .pluginspec-määrittelytiedostoissa.

QtCreator suorittaa …/plugins kansiossa olevat liitännäiset tietyssä järjestyksessä käynnistyksen yhteydessä. Tämä järjestys määräytyy .pro- ja .pluginscpec-tiedostojen perusteella. Muut projektin tiedostot sisältävät käytettävät objektit ja toiminnot kuten tavallisissakin Qt-sovelluksissa.

Tässä luvussa käydään läpi taustatutkimuksena toteutettu Nokia Hands-on lab e-oppimateriaali -liitännäinen QtCreator-sovellukselle. Luvussa on mukana ohjelmakoodia liitännäisestä, sekä määrittelytiedostoja, jotka on selitetty esimerkkeineen toteutus järjestyksessä.



Kuva 14. Liitännäinen auki QtCreator-kehitystyökalussa

5.1 HandsOnPlugin.pro

HansOnPlugin.pro-tiedosto pitää sisällään projektin ohjausmäärittelyt.
Jos projektissa käytetään joitakin Qt:n omia kirjastoja hyödyksi lisätään ne .pro-tiedoston alkuun:

```
QT += webkit \ gui
```

Webkit kirjastoliitäntä tuo pluginiin QWebkit-moduulin toiminnot, gui viittaa QtCore-moduuliin eli Qt-ydintoimintoon.

HandsOnPlugin-liitännäinen on määriteltävä myös kirjastoksi. Kirjaston nimi riippuu käyttöjärjestelmästä. Kirjaston nimen alkuosa määräytyy .pro tiedostossa TARGET määrityksellä ja pääte on alustakohtainen: HandsOnPlugin.dll Windows-ympäristössä ja libHandsOnPlugin.so Linux-ympäristössä:

```
TEMPLATE = lib
TARGET = HandsOnPlugin
```

Seuraavaksi määritellään asetukset joita liitännäinen käyttää hyödykseen. Asetukset on määritelty qtcreatorplugin.pri ja coreplugin.pri-tiedostossa. Nämä tiedostot ovat QtCreatorin ytimen suorittamia liitännäisten ohjaustiedostoja.

```
include(../../qtcreatorplugin.pri)
include(../../plugins/coreplugin/coreplugin.pri)
```

Seuraavat rivit .pro-tiedostossa määrittelevät projektiin kuuluvat tiedostot, ne tiedostot jotka pitävät sisällään liitännäisen muotoilumääritykset ja toiminnot. Vastaavat tiedot löytyvät jokaisesta Qt-projektista, oli se sitten liitännäinen tai sovellus:

```
HEADERS += HandsOnPlugin.h \ HandsOnWindow.h
SOURCES += HandsOnPlugin.cpp \ HandsOnWindow.cpp
```

Muut liitännäisen vaatimat tiedostot määritellään viimeiseksi .pro-tiedostossa, näitä tietoja ei löydy tavallisista Qt-sovelluksista:

```
OTHER_FILES += HandsOnPlugin.pluginspec
```

5.2 HandsOnPlugin.pluginspec

Jokaisella liitännäisellä kuuluu olla pluginspec-tiedosto, joka pitää sisällään jotakin metadataa liitännäisestä. Tiedosto pitää sisällään mm. liitännäisen nimen, jota käytetään myös liitännäisen kirjaston nimenä (HandsOnPlugin.dll Windowsissa ja libHandsOnPlugin.so Linuxissa).

Muita tietoja pluginspec-tiedostossa on versionumero, vaadittava QtCreator-versio, palveluntarjoajan nimi, tekijänoikeudet, lisenssiteksti, kuvaus, tekijän URL ja lista riippuvuuksista. Listassa on lueteltuna kaikki liitännäiset, joista tämä liitännäinen on riippuvainen.

QtCreatorin ydinprosessi ja PluginManager pitävät huolen, että .pluginspec-tiedostossa määritellyt riippuvuudet luodaan ennen liitännäisen luomista. Tässä tapauksessa ainoa riippuvuus on QtCreatorin ydinprosessi, joten liitännäinen luodaan ytimen käynnistyksen jälkeen tärkeysjärjestyksessä suhteessa muihin QtCreatorin liitännäisiin.

```
<plugin name="HandsOnplug-in" version="0.0.1" compatVersion="1.2.1">
      <vendor>Kirsi-Marja Eskola</vendor>
      <copyright>(C) 2010 Kirsi-Marja Eskola</copyright>
      <license>Do anything you want</license>
      <description>Plug-in to integrate Forum Nokia Hands-on lab e-
      learning material to QtCreator.</description>
      <url>thaa</url>
      <dependencyList>
      <dependency name="Core" version="1.2.1"/>
      </dependencyList>
</plugin>
```

pluginspec-tiedoston tulisi sijaita samassa kansiossa kuin liitännäisen .pro-tiedosto.

5.3 HandsOnplug-in.h

HandsOnPlugin-luokka toteuttaa IPlugin-rajapinnan ja perii Q_Object-luokan:

```
class HandsOnPlugin : public ExtensionSystem::IPlugin
{
    Q_OBJECT

public:
    HandsOnPlugin();
    ~HandsOnPlugin();
    void extensionsInitialized();
    bool initialize(const QStringList &arg, QString *errMsg);

private:
    HandsOnWindow *m_view;

private slots:
    void about();

};

} // namespace Internal
} // namespace HandsOn

#endif // HANDSONPLUG_IN_H
```

5.4 HandsOnPlugin.cpp

Paikallisten muuttujien (muu kuin widget tai funktio) luomista lukuun ottamatta rakentaja ja tuhoajat eivät tee mitään.

```
HandsOnPlugin::HandsOnPlugin()
{ }

HandsOnPlugin::~HandsOnPlugin()
{ }
```

Initialize()-metodia kutsutaan, kun QtCreatorin ydinprosessi on valmis liitännäisen alustukseen. Tämän toiminnon ideaalisin käyttötapa olisi, että siinä alustettaisiin liitännäisen sisäinen tila, sekä rekisteröitäisiin toiminnot ja objektit QtCreatorille.

Metodia kutsutaan kun ydinprosessi on luonut kaikki liitännäisen riippuvuudet. HandsOnPluginin ainoa riippuvuus on ydinprosessi, joten kutsuminen tapahtuu sovelluksen käynnistämisen alkuvaiheilla, heti tärkeämpien toimintojen suorittamisen jälkeen.

ErrMsg pitää sisällään virhetiedon, jos jokin meni pieleen liitännäisen luonnissa, ja sitä pitäisi käsitellä niin, että sen on käyttäjän luettavissa joko liitännäisen käyttöliittymästä tai debuggerin terminaalista käännettäessä/testattaessa.

```
bool HandsOnplugin::initialize(const QStringList &args,
QString *errMsg)
{
    Q_UNUSED(args);
    Q_UNUSED(errMsg);

    // Add a mode to BaseMode.
    Core::BaseMode *baseMode = new Core::BaseMode;
    baseMode->setUniqueModeName("HandsOn.HandsOnMode");

    //set mode name aka side bar button name
    baseMode->setName(tr("Hands-On Lab"));

    //set sidebar icon image
    baseMode->setIcon(QIcon("star.png"));

    //set priority on list 0=least important(on bottom)
    baseMode->setPriority(0);

    //set base mode window widget
    //for WebView get new HandsOnWindow-widget
    m_view = new HandsOnWindow();
    //set widget to base
    baseMode->setWidget(m_view);

    //set objects to workbench, destroyed automatically
    addAutoReleasedObject(baseMode);

    // Fetch the action manager
    Core::ActionManager* am = Core::ICore::instance()-
>actionManager();
```

```
// Create a HandsOn Lab menu
    Core::ActionContainer* ac = am->createMenu("HandsOnplug-
in.HandsOnMenu");
    ac->menu()->setTitle("Hands-On Lab");

    // Create a command for "HandsOnLab".
    Core::Command* cmd = am->registerAction(new
              QAction(this),
                  "HandsOnLab.AboutHandsOn",
                  QList<int>() << 0);
    cmd->action()->setText("Start Hands-On Lab on
                                windowedmode");

    // Add HandsOnLabmenu to the menubar
    am->actionContainer(Core::Constants::MENU_BAR)-
>addMenu(ac);

    // Add the "HandsOnLab" action to the HandsOn menu
    ac->addAction(cmd);
    // Connect the action
    connect(cmd->action(), SIGNAL(triggered(bool)), this,
SLOT(about()));

    return true;
}
```

ExtensionsInitialized()-toimintoa kutsutaan Initialize()-toiminnon jälkeen, kun kaikki liitännäisen riippuvuudet on luotu, tilanteessa jossa jonkin toisen liitännäisen toiminto on riippuvainen tästä liitännäisestä. Ydinprosessi kutsuu extensionsInitialized()-toimintoa ennen riippuvaisen liitännäisen suorittamista.

```
void HandsOnPlugin::extensionsInitialized()
{

}
```

Viimeisenä liitännäinen rekisteröidään .cpp-tiedoston lopuksi käyttämällä makroa Q_EXPORT_plugin:

```
Q_EXPORT_plugin(HandsOnPlugin)
```

6 Yhteenveto

Qt on joustava ja tehokas sovelluskehitysympäristö, jolla ohjelmoitaessa on mahdollista tuottaa laajamittaisia ohjelmistoprojekteja hyvin lyhyessä ajassa ja mikä parasta, sovellusmarkkinoita ajatellen, yhden koodin kirjoittaminen kaikille kohdelaitteille yhteensä on tehokkuuden huipentuma. Qt perustuu C++-ohjelmointikieleen, ja on kehitetty ohjelmoijien tarpeeseen helpottamaan ohjelmointia sekä sovellusten levitystä.

Qt:tä tukevia alustoja ovat mm. Linux-, Windows- sekä Mac OS X -käyttöjärjestelmän sisältävät tietokoneet, Nokian Symbian- ja maemo-mobiililaitteet, sekä sulautettujen järjestelmien laitteita, kuten muutamat Fluke-mittalaitteet.

Qt on käytössä todella monessa suositussa sovelluksessa meidän siitä mitään tietämättä, vaikka olemme tuttuja ohjelmointimaailman kanssa. Qt on alustariippumaton ohjelmointikieli, mutta monella yrityksellä se on käytössä vain yhdelle kohdelaitteelle tarkoitetussa kaupallisessa sovelluksessa. Qt:n suosiosta on voidaan päätellä sen olevan helposti omaksuttava kieli.

Liitännäiset ovat sovelluksen lisäosia, jotka on tuotettu helpottamaan sovelluksen toimintaa tai lisäämään siihen ominaisuuksia joita se ei normaalisti sisällä. Liitännäiset eivät toimi ilman sovellusta johon se on kehitetty.

Qt:n toiminnot perustuvat pääsääntöisesti ytimen ympärille liitettyihin liitännäisiin. Qt sovelluskehityksessä käytettävän QtCreator-työkalun jokainen komponentti on oma liitännäisensä. Liitännäiset luodaan sovelluksen käynnistyessä ja niille tehdään yhteensopivuustarkastus. Jos liitännäinen on yhteensopiva, se käynnistetään ja on käytettävissä sovelluksessa kuten muutkin sen alkuperäiset toiminnot.

QtCreator on hyvä sovellus Qt-aloittelijalle esimerkkien ja laajennettavuutensa ansiosta. Kieleen sisälle päästyään Eclipse-integraatiot ja muut kehitystyökalut tarjoavat tehokkaamman tavan ohjelmoida. QtCreator-liitännäisten ohjelmoinnin helppouden ansiosta jokainen voi kuitenkin muodostaa QtCreatorista juuri itselleen sopivan kehitystyökalun.

Qt:n tarjoamat kirjastot on helppo oppia. Qt:n objektien avulla voidaan luoda näyttäviä ja toimivia käyttöliittymiä lyhyessä ajassa. QtCreatoriin sisältyvä QtDesigner on hyvä lähtökohta graafisten käyttöliittymien luomiselle. Kieltä jo oppineet kirjoittavat kuitenkin käyttöliittymäkomponentit toiminnallisuuksineen nopeasti suoraan koodiin, joka jättää QtDesignerin käytön lähes olemattomaksi.

Qt on monipuolinen, helposti omaksuttava ja kiinnostava sovelluskehitysympäristö. Qt:llä saa nopeasti aikaan näyttäviä ja toimivia sovelluksia. Qt:n paras puoli on sen alustariippumattomuus, joka helpottaa sovellusten tuottamista eri alustoille. Linux-käyttöjärjestelmä kasvattaa jatkuvasti suosiotaan ja markkinoille on tuotu yhä enemmän Qt:tä tukevia älypuhelimia, nämä ja monet muut syyt varmasti lisäävät Qt:n suosiota myös harrastajien parissa, uusien innovaatioiden luomisessa.

Lähdeluettelo

Kirjalliset lähteet

/1/ Blanchette, Jasmin & Summerfield, Mark 2008. C++ GUI Programming with Qt4. Prentice Hall.

/2/ Molketin, Daniel 2007. The Book of Qt 4, the Art of Building Qt Applications. No Starch Press.

Sähköiset lähteet

/3/ Add-ons for Mozilla Firefox [www-sivu][viitattu 19.4.2010] https://addons.mozilla.org/fi/firefox/browse/type:7

/4/ Plugin – Tekniikkatermien sanakirja [www-sivu][viitattu 11.4..2010]. http://fin.afterdawn.com/sanasto/selitys.cfm/plugin

/5/ Qt Reference Documentation. [www-sivu] [viitattu 29.3.2010]. http://doc.trolltech.com/4.6/index.html

/6/ Writing QtCreator Plugins. [www-sivu] [viitattu 29.3.2010]. http://www.vcreatelogic.com/downloads/?file=Writing-Qt-Creator-Plugins&type=pdf

Liitteet

LIITE 1

HansOnPlugin-liitännäisen lähdekoodit kokonaisuudessaan.

```
HandsOnPlugin.pro
*********************************************
QT += webkit \ gui
CONFIG += release
TEMPLATE = lib
TARGET = HandsOnPlugin
include(../../qtcreatorplugin.pri)
include(../../plugins/coreplugin/coreplugin.pri)
HEADERS += HandsOnPlugin.h \
    HandsOnWindow.h
SOURCES += HandsOnPlugin.cpp \
    HandsOnWindow.cpp
OTHER_FILES += HandsOnPlugin.pluginspec
```

```
HandsOnPlugin.pluginspec
*********************************************
<plugin name="HandsOnPlugin" version="0.0.1" compatVersion="1.3.1">
    <vendor>Kirsi-Marja Eskola</vendor>
    <copyright>(C) 2010 Kirsi-Marja Eskola</copyright>
    <license>Do anything you want</license>
    <description>Plugin to integrate Forum Nokia Hands-on lab e-
learning material to QtCreator.</description>
    <url>thaa</url>
    <dependencyList>
        <dependency name="Core" version="1.3.1"/>
    </dependencyList>
</plugin>
```

```
/**************************************************************************
  *
  * HandsOnPlugin.h
  * Author: Kirsi-Marja Eskola
  * Date: 14.02.2010
  *
  *************************************************************************/
#ifndef HANDSONPLUGIN_H
#define HANDSONPLUGIN_H

#include <extensionsystem/iplugin.h>
#include <QtWebKit/QWebView>
#include "HandsOnWindow.h"


namespace HandsOn {
namespace Internal {

class HandsOnPlugin : public ExtensionSystem::IPlugin
{
    Q_OBJECT


public:
    HandsOnPlugin();
    ~HandsOnPlugin();
    void extensionsInitialized();
    bool initialize(const QStringList &arg, QString *errMsg);

private:
    HandsOnWindow *m_view;

private slots:
    void about();

};

} // namespace Internal
} // namespace HandsOn

#endif // HANDSONPLUGIN_H
```

```
/**************************************************************************
  *
  * HandsOnPlugin.cpp
  * Author: Kirsi-Marja Eskola
  * Date: 14.02.2010
  * Description: HandsOnPlugin functionality
  *
  **************************************************************************/
#include "HandsOnPlugin.h"

#include <coreplugin/actionmanager/actionmanager.h>
#include <coreplugin/basemode.h>
#include <coreplugin/coreconstants.h>
#include <coreplugin/icore.h>
#include <coreplugin/modemanager.h>
#include <coreplugin/uniqueidmanager.h>
#include <QtCore/QtPlugin>
#include <QtGui/QAction>
#include <QtGui/QMenu>

using namespace HandsOn::Internal;

//constructor
HandsOnPlugin::HandsOnPlugin()
{
}
//destructor
HandsOnPlugin::~HandsOnPlugin()
{
}
//function when plug-in inizialised
bool HandsOnPlugin::initialize(const QStringList &args, QString
*errMsg)
{
    Q_UNUSED(args);
    Q_UNUSED(errMsg);

    // Add a mode to BaseMode.
    Core::BaseMode *baseMode = new Core::BaseMode;
    baseMode->setUniqueModeName("HandsOn.HandsOnMode");

    //set mode name aka side bar button name
    baseMode->setName(tr("Hands-On Lab"));

    //set sidebar icon image
    baseMode->setIcon(QIcon("star.png"));

    //set priority on list 0=least important(on bottom)
    baseMode->setPriority(0);

    //set base mode window widget
    //for WebView get new HandsOnWindow-widget
    m_view = new HandsOnWindow();
    //set widget to base
    baseMode->setWidget(m_view);

    //set objects to workbench, destroyed automatically
```

```
    addAutoReleasedObject(baseMode);

    // Fetch the action manager
    Core::ActionManager* am = Core::ICore::instance()-
      >actionManager();

    // Create a HandsOn Lab menu
    Core::ActionContainer* ac = am-
      >createMenu("HandsOnPlugin.HandsOnMenu");
    ac->menu()->setTitle("Hands-On Lab");

    // Create a command for "HandsOnLab".
    Core::Command* cmd = am->registerAction(new QAction(this),
                                            "HandsOnLab.AboutHandsOn",
                                            QList<int>() << 0);
    cmd->action()->setText("Start Hands-On Lab on windowed mode");

    // Add HandsOnLabmenu to the menubar
    am->actionContainer(Core::Constants::MENU_BAR)->addMenu(ac);

    // Add the "HandsOnLab" action to the HandsOn menu
    ac->addAction(cmd);
    // Connect the action
    connect(cmd->action(), SIGNAL(triggered(bool)), this,
      SLOT(about()));

    return true;
}
void HandsOnPlugin::extensionsInitialized()
{

}
void HandsOnPlugin::about()
{
    //WINDOWED MODE FOR HANDS-ON LAB

    //for WebView get new HandsOnWindow-widget
    m_view = new HandsOnWindow();

    //set widget visible
    m_view->setVisible(true);

    //set objects to workbench
    addAutoReleasedObject(m_view);

}


Q_EXPORT_PLUGIN(HandsOnPlugin)
```

```
#ifndef HANDSONWINDOW_H
#define HANDSONWINDOW_H
/*************************************************************************
  *
  * HandsOnWindow.h
  * Author: Kirsi-Marja Eskola
  * Date: 14.02.2010
  *
*************************************************************************/
#include <QtGui/QWidget>
#include <QtWebKit/QWebView>
#include <QtCore/QUrl>
#include <QtGui/QLabel>
#include <QtCore/QUrl>

QT_FORWARD_DECLARE_CLASS(QLabel);

namespace HandsOn {
namespace Internal {

class HandsOnWindow : public QWidget
{
    Q_OBJECT

public:
    HandsOnWindow(QWidget *parent = 0);

private slots:
    void loadFin(bool);
    void loadStart();

private:
    QWebView *view;
    QLabel *label;
    QUrl *url;

};

} // namespace Internal
} // namespace HandsOn

#endif // HANDSONWINDOW_H
```

```
/*****************************************************************
  *
  * HandsOnWindow.cpp
  * Author: Kirsi-Marja Eskola
  * Date: 14.02.2010
  * Description: Creates a webview widget to use on HandsOnPlugin.
  *
  *****************************************************************/
#include "HandsOnWindow.h"

#include <QtCore/QUrl>
#include <QtGui/QProgressBar>
#include <QtGui/QPushButton>
#include <QtWebKit/QWebSettings>

using namespace HandsOn::Internal;

HandsOnWindow::HandsOnWindow(QWidget *parent)
   : QWidget(parent)
{
    view = new QWebView(this);          //create webview-widget
    view->setGeometry(0,30,1020,700);   //set webview window geometry

    view->page()->settings()-
      >setAttribute(QWebSettings::JavascriptEnabled, true);
    view->page()->settings()-
      >setAttribute(QWebSettings::JavascriptCanOpenWindows, true);
    view->page()->settings()->setAttribute(QWebSettings::JavaEnabled,
      true);
    view->page()->settings()->setAttribute(QWebSettings::ZoomTextOnly,
      true);

    url = new QUrl("http://www.forum.nokia.com/document/Mobile_Hands-
      on_Labs/?javascript:fn_toggleNavigation(false);");
    //url->addQueryItem("function"," );
    QString s_url = url->toString(QUrl::FormattingOption(0x0));

    view->load(s_url); //load specified url

    QPushButton *button = new QPushButton("Reload Page",
      this);     //create reload button
    button->setGeometry(0, 0, 101,
      24);                              //set button position
    QObject::connect(button, SIGNAL(clicked()),view ,SLOT(reload()));
      //set signal to reload on button press

    QProgressBar *pbar = new QProgressBar(this);        //create
      progressbar
    pbar->setGeometry(102, 0, 250, 24);                 //set
      progressbar position
    QObject::connect(view, SIGNAL(loadProgress(int)),pbar
      ,SLOT(setValue(int)));    //set signal to indicate load progress

    label = new QLabel(this);                           //create text
      label
    label->setGeometry(353, 0, 400, 24);                //set label
      position
```

```
    label->setText("Loading Page, Please wait...");     //set label
      text
    QObject::connect(view, SIGNAL(loadFinished(bool)),this
      ,SLOT(loadFin(bool)));   //set signal to read finished load or
      error
    QObject::connect(view, SIGNAL(loadStarted()),this
      ,SLOT(loadStart()));          //set signal to read started load

}
void HandsOnWindow::loadFin(bool state)
{
    //if load finished state is true set text to load Finished, else
      ERROR
    if(state==true)
    {

        label->setText("Load Finished!");
    }
    else
    {
        label->setText("Error loading page!");
    }
}
void HandsOnWindow::loadStart()
{
    //on load set text...
    label->setText("Loading Page, Please wait...");
}
```