Jetro Jormalainen

Navigaatio-ohjelman sovittaminen Sailfish-alustalle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

13.05.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jetro Jormalainen Navigaatio-ohjelman sovittaminen Sailfish-alustalle 52 sivua 13.05.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Juha Kämäri, Lehtori Jorma Räty, Lehtori

Insinöörityössä tutustutaan Qt-kehitysympäristöön ja sen perusominaisuuksiin. Työssä esitellään myös Qt:n uudempaa mobiilikehitykseen tarkoitettua QML-kieltä ja käytetään sitä käyttöliittymän luomiseen. Työssä painotutaan käyttöliittymän kehittämisessä Jollan SailfishOS-alustaan ja sen Silica-komponentteihin. Silica-komponenteilla voi luoda Sailfishin tyyliin sopivan käyttöliittymän. Lukijalla kannattaa olla pohjatietoa ohjelmoinnista C++:lla.

Tekstissä kerrotaan Qt:n omien työkalujen lisäksi Sailfishin SDK:sta ja siitä, kuinka ohjelmat saadaan yhteensopivaksi SailfishOS-alustan kanssa.

Raportin ohella tehtiin esimerkkiprojekti MoNav-navigointiohjelman sovittamisesta Jollan SailfishOS-alustalle. Ohjelmalle kehitettiin uusi käyttöliittymä QML-kielellä. Sovituksessa tuli esiin joitain ongelmia ja niitä käsitellään, sekä niihin kerrotaan ratkaisuja työssä. Ohjelma sovitettiin onnistuneesti uudelle alustalle ja julkaistiin OpenRepos.net-palvelussa.

Avainsanat	Qt, QML, C++, SailfishOS, Ohjelmointi

Author(s)	Jetro Jormalainen
Title	Porting Navigation Software to Sailfish Platform
Number of Pages	52 pages
Date	Wednesday 13th May, 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor(s)	Juha Kämäri, Lecturer Jorma Räty, Lecturer

The thesis introduces the Qt development environment and its basic properties. The study also introduces the QML language for Qt's newer mobile development and it is used for the user interface creation. The study highlights user interface development for Jolla's SailfishOS platform and Silica components. Silica components can create a user interface style suitable for Sailfish. Readers should have a basic knowledge of programming in C++.

The study tells about Sailfish SDK in addition to Qt's own tools and how programs can be made compatible with the SailfishOS platform.

Alongside with the report, an example project of porting the MoNav navigation program to Jolla's SailfishOS platform was created. A new user interface was developed for the program with the QML language. While porting the program there came up some problems which are discussed and also solutions are provided. The program were ported successfully to the new platform and was published on the OpenRepos.net service.

Keywords	Qt, QML, C++, SailfishOS, Programming

Sisällys

Lyhenteet

C++	Ohjelmointikieli.
D-Bus	Palvelu tietojen välitykseen ohjelmien välillä.
Geoclue	Sijaintitietopalvelu.
JSON	JavaScript Object Notation, tiedonvälitysmuoto.
mce	Palvelu puhelimen tilojen hallintaan.
Mer	Mobiilikäyttöjärjestelmän ydin.
moc	Meta-object compiler, kääntäjä Qt:n käyttämille lisäyksille C++-kieleen.
QML	Qt Meta Language, merkintäkieli käyttöliittymille.
Qt	Kehitysympäristö.
RPM	RPM Packet Manager, ohjelmapakettiformaatti ja -järjestelmä.
SDK	Software Development Kit, paketti ohjelmien kehitykselle.
SDL2	Simple DirectMedia Layer, alusta graafisille ohjelmille.
SSH	Secure Shell, salattu yhteys etäkomentotulkki-istunnoille.
XDG	Tunnus freedesktop.orgin määrittelyissä.
XML	Extensible Markup Language, merkintäkieli.
yaml	YAML Ain’t Markup Language, merkintäkieli.

1 Johdanto

Jolla-puhelimelle ja sen käyttämälle Sailfish-käyttöjärjestelmälle ei ole ollut natiivia helppokäyttöistä navigointohjelmaa, joka toimisi täysin ilman verkkoyhteyttä. Vastaavia Android-pohjaisia sovelluksia on, jotka Jollan käyttöjärjestelmällä toimivat hyvin, mutta tarve natiiville ohjelmalle on kuitenkin olemassa. Esimerkiksi natiivi ohjelma osaa näyttää tietoja ohjelman kannessa, kun se on pienennettynä. Käyttäjä ei välttämättä pysty tai halua myöskään asentaa Android-tukea Jolla-puhelimeensa.

MoNav on navigointisovellus, jota on kehitetty Qt:llä. Ohjelma toimii näin ollen useilla eri alustoilla, mutta Sailfish-käyttöjärjestelmässä se ei enää uusimmissa päivityksissä toimi. Tämä johtuu siitä, että ohjelma käyttää vanhaa tapaa käyttöliittymän luomiseksi, jota ei ilmeisesti Sailfishissä tueta. MoNav sisältää yleisimmät ominaisuudet navigointiin ja toimii täysin ilman verkkoyhteyttä. Sen karttadata tulee crowdsourcingilla karttadatan muodostaneesta OpenStreetMapista, joten se on erittäin hyvä mm. pyöränavigointiin.

Työssä kerrotaan Qt-kehitysympäristön käytöstä ja käyttöliittymän luomisesta sen avulla. Käyttöliittymän luomiseen käytetään Qt:n uutta QML-kieltä. Lisäksi tutustutaan Sailfish-käyttöjärjestelmän Silica-komponentteihin käyttöliittymän luomiseksi.

Työn tavoitteena on tutkia valmiin MoNav-karttasovelluksen sovittamista uudelle alustalle. Sovellusta ei siis ole tarkoitus kirjoittaa kokonaan uudestaan, vaan tarkoitus on muuttaa se yhteensopivaksi uuden alustan kanssa. Sailfish-alusta käyttää eri Qt:n versiota ja ominaisuuksia, mitä MoNav alun perin käyttää. Sovelluksen sovittamiseksi luodaan uusi käyttöliittymä käyttämällä uuden alustan kirjastoja. Käyttöliittymään kuuluu osoitehaku, navigoinnin aloitus- ja lopetuspaikkojen muuttaminen ja karttamoduulien vaihtami-

nen. Tavoitteena on myös saada ohjelma vastaamaan julkaisuvaatimuksia Jolla Harbour -julkaisupalvelussa.

2 Tutustuminen ohjelmointikieleen

2.1 Qt:n ja QML-kielen esittely

2.1.1 Yleistä Qt:stä

Qt on kehitysympäristö graafisten sovellusten tuottamiseen [1]. Se käyttää ohjelmointikielenä C++:aa ja lisää kieleen ominaisuuksia C++-kääntäjän lisäksi käytettävän oman Metaobject compilerin (moc) avulla. Näitä ominaisuuksia ovat mm. signalit ja slotit. [1, Qt Core.] Graafisten käyttöliittymien luonnin lisäksi Qt:ssä on luokkia monien eri toimintojen luomiseen, kuten verkkoyhteyksin käyttäminen ja käyttäjän sijainnin selvittäminen. Se myös lisää tuen monille järjestelmän ominaisuuksille, kuten säikeille ja sisältää ison osan korvaavia luokkia C++:n standardikirjastolle. Qt toimii useilla eri alustoilla, mukaan lukien mobiilialustat. [1.]

Qt sai alkunsa norjalaisessa yrityksestä nimeltä Trolltech. Se perustettiin vuonna 1994 ja perustajina olivat Haavard Nord ja Eirik Eng. [2.] Qt:tä on kehitetty kuitenkin jo vuodesta 1992 lähtien ja ensimmäinen kaupallinen versio julkaistiin vuonna 1995 [3]. Qt 1.0 tuki Windows- ja Unix/X11-järjestelmiä [4]. Se julkaistiin kaupallisella ja ilmaisella lisenssillä. Toisin kuin ilmaisella, kaupallisella lisenssillä pystyi julkaisemaan omia kaupallisia ohjelmia ja luomaan ohjelmia myös Windowsille. [5.]

Jo ensimmäisissä versioissa käytettiin signaleeja ja slotteja ja alkuperäisen järjestelmän käyttöliittymän tyyliä emuloitiin. Eri C++-luokkia oli noin 140 ja niiden avulla voitiin käyttöliittymän luomisen lisäksi mm. käyttää käyttöjärjestelmän toimintoja, kuten tiedostonkäsittelyä. [6.]

Suurimpia Qt:tä käyttäviä projekteja oli KDE-työpöytäympäristö. Tulevaisuutta ajatellen luotiin yhdistys, jonka tarkoituksena oli mahdollistaa Qt:n ilmaisen version saatavuus, jos se lakkautettaisiin. Yhdistys koostui Qt:n ja KDE:n jäsenistä. [7.] Qt:n ilmainen versio kuitenkin julkaistiin myöhemmin 4.3.1999 heidän omalla QPL-avoimen lähdekoodin lisenssillään [8].

- Qt 2.0 julkaistiin 25.6.1999 ja sen suurimpia uudistuksia oli tuki kansainvälistämiselle ja teemat [9].
- Qt 2.2 julkaistiin 6.9.2000, jonka mukana tuli graafinen käyttöliittymien kehitystyökalu Qt Designer [10].
- Trolltech julkaisi Qt:n Windows version ilmaisella lisenssillä 26.6.2001 [11].
- Qt 3.0 julkaistiin 15.10.2001. Uusi versio tuki myös Mac OS X -käyttöjärjestelmää. [12.]
- Qt 4.0 julkaistiin 28.6.2005. Uusia ominaisuuksia oli mm. Visual Studio tuki, päivitetty grafiikkakirjasto ja parempi monisäikeistystuki. [13.]
- Nokia osti Trolltechin 28.1.2008 [14]. Trolltechistä tuli virallisesti osa Nokiaa ja sai uuden nimen Qt Software 27.9.2008 [15].
- Qt 4.4 julkaistiin 6.5.2008. Sen mukana tuli tuki WebKitille ja Windows CE:lle. [16.]
- Qt 4.5 julkaistiin 3.3.2009, jonka mukana tuli myös uusi Qt Creator -ohjelmistokehitystyökalu [17].
- Qt 4.6 julkaistiin 1.12.2009 ja tukee ensimmäistä kertaa Symbiania ja monikosketusta [18]. Pari kuukautta aiemmin siitä julkaistiin ennakkoversio, jossa oli tuki Maemolle [19].
- Qt 4.7 julkaistiin 21.9.2010. Uusi iso ominaisuus oli Qt Quick, jota oli ideoitu ja kehitetty kaksi vuotta. [20.]
- Qt 4.8 julkaistiin 25.12.2011 ja lisäsi uutena ominaisuutena Qt Platform Abstractionin, joka uudelleenjärjestää käyttöliittymäpinoa [21].
- Digia osti Qt:n Nokialta 9.8.2012 [22].
- Qt 5.0 julkaistiin 19.12.2012 [23].

Qt julkaistaan oman Qt Commercial Licensen lisäksi avoimen lähdekoodin lisensseillä. Eroavaisuuksia on muun muassa se, että Commercial Licensellä voi muokata Qt:n koodia ilman, että sitä tarvitsee julkaista. Lisäksi Commercial Licensessä tulee joitakin ominaisuuksia, kuten kaaviotyökalut ja paremmat kehitystyökalut. [24.]

2.1.2 Qt:n moduulit

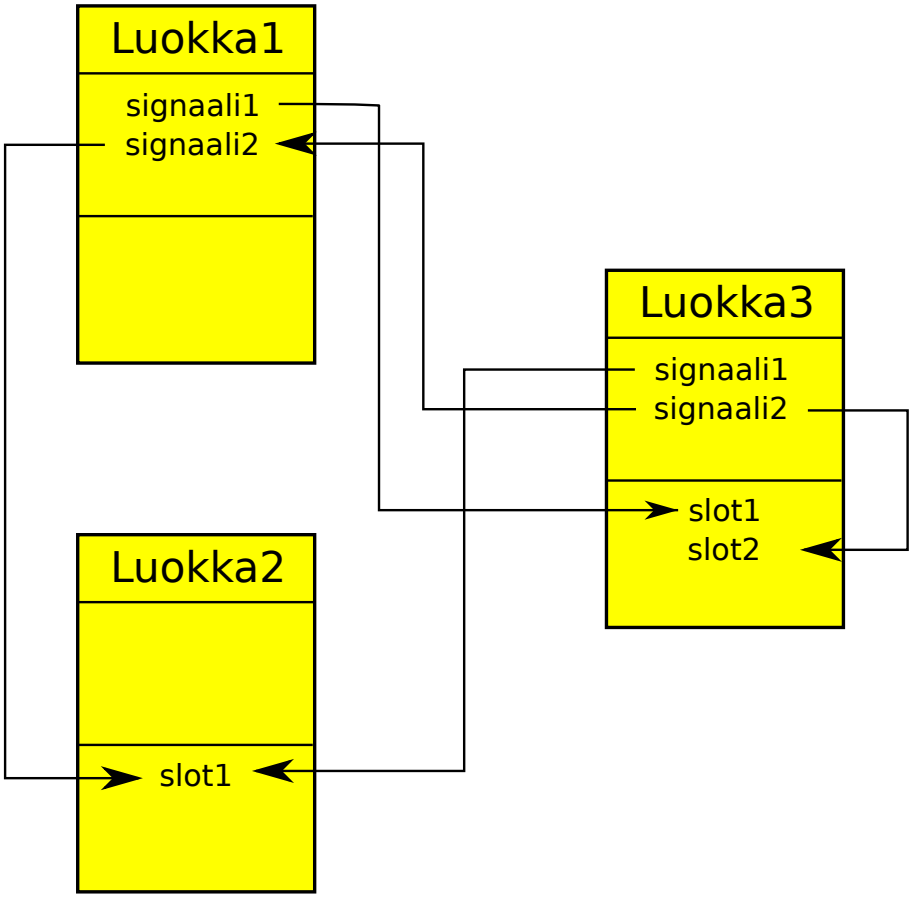
Qt muodostuu useasta eri moduulista. Tärkein moduuli on Qt Core -moduuli, jonka luokkia kaikki muut Qt:n moduulit käyttävät. Qt Coren mukana tulee QObject-luokka, jonka kaikki muut Qt:n luokat perivät. [1, Qt Core.] Qt Corella ei kuitenkaan voida luoda graafista käyttöliittymää, vaan siihen käytetään Qt GUI -moduulia ja muita Qt:n tarjoamia moduuleita [1, Qt GUI].

Qt GUI mahdollistaa ikkunoiden luonnin ja tarjoaa luokat grafiikan piirtämiseen 2D:nä ja 3D:nä [1, Qt GUI]. Käyttöliittymän luomiseen on muutamia moduuleita. Qt Widgets -moduulia käytetään perinteisen työpöytäkäyttöliittymän luomiseen. QtWidgetsin QStyle-luokka pitää huolen, että käyttöliittymän elementit näyttävät järjestelmän natiiveilta elementeiltä. [1, Qt Widgets.]

Qt Quick -moduuli on tarkoitettu luomaan modernimpi käyttöliittymä, joka on tuttu mm. matkapuhelimista. Sen komponentteja ohjelmoidaan Qt QML -moduulin QML-kielellä. [1, Qt Quick.]

2.1.3 Signaalit ja slotit

Signaalit ja slotit ovat Qt:n keskeinen osa Qt:llä ohjelmoimiseen. Niitä käytetään olioiden kesken tiedon välitykseen. Graafisen käyttöliittymän ohjelmoinnissa on yleistä, että käyttöliittymän tapahtumat vaikuttavat ohjelman toimintaan. Esimerkiksi painiketta painamalla siirrytään seuraavalle sivulle. Normaalisti tämä tehtäisiin callback-funktioita käyttäen. [1, Qt Core.]



Kuva 1: Kolmen luokan välillä yhdistetyt signaalit ja slotit

Signaali lähetetään luokassa, kun jokin asia tapahtuu. Esimerkiksi signaali voidaan lähettää painiketta painettaessa, ikkunan sulkeutuessa tai verkkoyhteyden saadessa dataa. Signaalit yhdistetään slotteihin ja sloteissa määritetty koodi suoritetaan, kun siihen kytketty signaali lähetetään. [1, Qt Core.] Qt:ssä esimerkin painikkeella voisi olla clicked-signaali ja sivuja hallitsevalla luokalla changePage-slot. clicked-signaali yhdistettynä changePage-slottiin vaihtaisi näin ollen sivua painiketta painettaessa.

```
1  class Luokka1 : public QObject {
2    Q_OBJECT
3  signals:
4    void signaali1(QString text);
5    void signaali2();
6  };
7
8  class Luokka2 : public QObject {
9    Q_OBJECT
10 public slots:
11   void slot1() {
12     qDebug() << "tulostus Luokka2 slot1";
13   }
14 };
```

```
15
16 class Luokka3 : public QObject {
17   Q_OBJECT
18 signals:
19   void signaali1();
20   void signaali2();
21 public slots:
22   void slot1(QString text) {
23     qDebug() << "tulostus Luokka3 slot1" << text;
24   }
25   void slot2() {
26     qDebug() << "tulostus Luokka3 slot2";
27   }
28 public:
29   void emitSignal() {
30     emit signaali2();
31   }
32 };
33
34 class Controller {
35 private:
36   Luokka1 l1;
37   Luokka2 l2;
38   Luokka3 l3;
39 public:
40   Controller() {
41     QObject::connect(&l1, SIGNAL(signaali1(QString)), &l3,
          SLOT(slot1(QString)));
42     QObject::connect(&l1, SIGNAL(signaali2()), &l2, SLOT(
          slot1()));
43     QObject::connect(&l3, SIGNAL(signaali1()), &l2, SLOT(
          slot1()));
44     QObject::connect(&l3, SIGNAL(signaali2()), &l1, SIGNAL(
          signaali2()));
45     QObject::connect(&l3, SIGNAL(signaali2()), &l3, SLOT(
          slot2()));
46     l3.emitSignal();
47   }
48 };
```

Listaus 1: Signaali ja slot luokissa

Perinteisten C++-luokan metodien lisäksi voidaan luokkiin määrittää signaaleja ja slotteja, kuten listauksessa 1. Listauksessa kuvataan sama tilanne kuin kuvassa 1. Ainut listauksen signaali, joka lähetetään, on Luokka3:n signaali2.

Signaaleja, slotteja sekä muita Qt:n ominaisuuksia käyttäessä luokan tulee periytyä QObject luokasta. Luokat merkitään myös Q_OBJECT-makrolla luokan private-osiossa. Sig-

naalit ja slotit määritetään kuten tavalliset metodit. Signaaleille ei kuitenkaan tehdä metodin toteutusta. Slotit toimivat myös normaaleina metodeina, joten niitä voi kutsua kuten normaaleita metodeita. Signaalia ja slotia yhdistäessä niiden signeerausten täytyy vastata toisiaan. Signaali ei voi palauttaa mitään, joten palautustyyppi on aina void. [1, Qt Core.]

Signaali ja slot voidaan yhdistää QObjectin connect-metodilla. Connect-metodin argumentteina on luokkien olioiden osoittimet, joissa kyseiset signaali ja slot on määritetty. Argumentteinä annetaan myös signaali, joka lähetetään, ja slot, jota kutsutaan, kun signaali lähetetään. Signaalin voi yhdistää moneen slotiin, ja slot voi vastaanottaa monta eri signaalia. [1, Qt Core.]

Signaalin voi lähettää emit-komentoa käyttäen. Signaalin lähettäminen onnistuu vain siinä luokassa, jossa se on määritetty. [1, Qt Core.]

QObject-oliot on liitetty johonkin tiettyyn säikeeseen, jolloin tiedetään, missä säikeessä signaalin tullessa slot suoritetaan. Kyseinen säie voidaan selvittää thread-metodilla. moveToThread-metodilla QObject voidaan siirtää toiseen säikeeseen. Säie voi olla myös nolla, jolloin slottia ei suoriteta. [1, Qt Core.]

2.1.4 Property-järjestelmä

Propertyt ovat kuin tavallisia muuttujia luokassa, mutta niillä on joitakin lisäominaisuuksia. Ne voi määrittää vastaamaan suoraan jotain luokan muuttujaa tai niille voi asettaa luku- ja kirjoitusmetodit arvon lukemista ja asettamista varten. Luokan käyttäjän ei tarvitse miettiä, kuinka propertyn arvo saadaan tai asetetaan luokan sisällä. Property voikin olla ensin vain suoraan määritetty vastaamaan jotain luokan muuttujaa. Arvon asettamisen tai lukemisen monimutkistuessa property voidaan asettaa käyttämään luku- ja kirjoitusmetodeita. [1, Qt Core.]

```
1 class Propertyja : public QObject {
2   Q_OBJECT
```

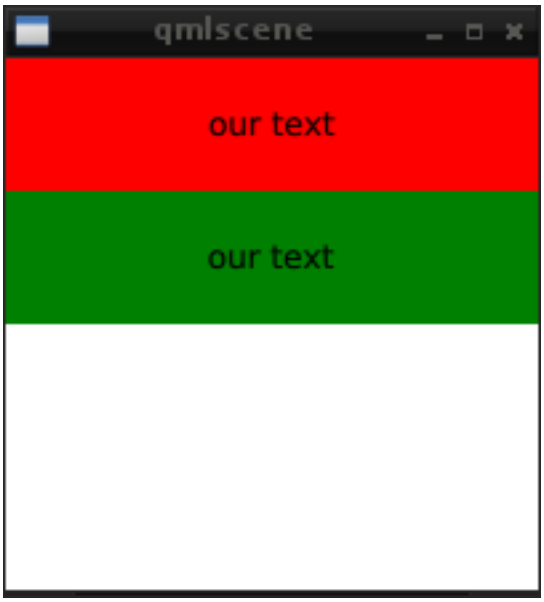
```
3    Q_PROPERTY(QStringList property1 READ getProperty1 WRITE
        setProperty1 NOTIFY property1Changed)
4    Q_PROPERTY(int property2 MEMBER prop2)
5  signals:
6    void property1Changed();
7  public:
8    QStringList getProperty1() {
9      return prop1;
10   }
11   void setProperty1(QStringList prop1) {
12     this->prop1 = prop1;
13     emit property1Changed();
14   }
15 private:
16   QStringList prop1;
17   int prop2;
18 };
```

Listaus 2: Propertyjä määritetty eri tavoilla

Propertyn tyyppi voi olla mikä tahansa tyyppi, jonka QVariant-luokka tukee. Se määritetään C++:ssa Q_PROPERTY-makrolla. Listauksessa 2 määrittää property1-property, jonka tyyppi on QStringList ja sitä luettaessa arvo luetaan käyttäen getProperty1-metodia. Propertyn voi asettaa käyttäen QObjectin setProperty-metodia ja lukea käyttäen property-metodia. Propertyjä pystytään määrittämään vain QObject-luokassa ja ne tulevat myös Qt:n Meta-Object Systemin tietoon. Meta-Object Systemin ansiosta niitä voi käyttää myös QML:ssä. Jos propertyä on mahdollista kirjoittaa, QML:ssä käytettäessä sillä täytyy olla myös NOTIFY-avainsana. Sillä määritetään signaali, joka kuuluu lähettää, kun arvo propertyssä muuttuu. [1, Qt Core.]

2.1.5 QML-kieli ja Qt Quick -elementit

QML on käyttöliittymien suunnitteluun ja ohjelmointiin tarkoitettu kieli. Sillä pystyy määrittämään visuaalisia elementtejä ja miten ne toimivat ja sijoittuvat muihin elementteihin nähden. QML:n syntaksi on helposti luettava ja muistuttaa JSON:ää. Siinä objektit muodostavat puumallin, jossa objekteilla voi olla vanhempi ja lapsia. QML:ssä voi kirjoittaa toimintoja JavaScriptillä. [1, QML Applications.]



Kuva 2: Listauksen 3 koodi avattuna qmlscene-ohjelmalla

Qt Quick tarjoaa tarvittavat elementit käyttöliittymän rakentamiseen, kuten yksinkertaisten suorakulmioiden, kuvien ja tekstikenttien näyttäminen tai toimintoihin, kuten käyttäjän monikosketuksen havainnointi [1, Qt Quick].

```
1  import QtQuick 2.0
2
3  Item {
4      id: main
5      width: 200
6      height: 200
7      property string labeltext: "our text"
8      property alias redlabelalias: redlabel
9
10     Column {
11     anchors {
12       left: parent.left
13       right: parent.right
14     }
15     Rectangle {
16       color: "red";
17       anchors.left: parent.left
18       anchors.right: parent.right
19       height: 50
20       Text {
21         id: redlabel
22         anchors.centerIn: parent
23         text: labeltext
24       }
25     }
26     Rectangle {
27       color: "green"
```

```
28        anchors {
29          left: parent.left
30          right: parent.right
31        }
32        height: 50
33        Text {
34          id: greenlabel
35          anchors.centerIn: parent
36          text: redlabelalias.text
37        }
38      }
39      }
40  }
```

Listaus 3: Elementit QML:ssä

Item määrittää yksinkertaisimmillaan näkyvien elementtien alustan, ja kaikki näkyvät elementit perustuvat siihen. Siinä on toiminnot visuaalisen elementin näyttämiseen, kuten koko ja sijainti sekä tuki näppäinpainallusten hallinnoimiseen. [1, Qt Quick.] Listauksen 3 Column asettelee lapsielementit allekkain, eli kyseinen QML-dokumentti näyttää punaisen ja vihreän Rectangle-alueen allekkain. Alueiden keskelle on määritetty vielä tekstit.

Columnissa asetetaan sen anchors-ominaisuudet niin, että Columnin oikea ja vasen sivu ovat kiinni vanhemman eli Itemin sivuissa. Jos Itemin koko muuttuu, Columnin koko muuttuu myös. Columnissa anchors ominaisuuksien asettelu on ryhmitetty niin, että anchors.left:in sijasta tarvitsee kirjoittaa vain left. [1, Qt Quick.]

Jokainen QML-komponentti luo oman näkyvyyden. Näkyvyys koostuu komponentin sisällä olevien objektien id:istä ja root-objektin propertyistä. [1, Qt QML.] Tämän vuoksi redlabel voi suoraan määrittää textille labeltext propertyn. greenlabel ei kuitenkaan näe redlabelia suoraan vaan joutuu käyttämään sen textiä root-objektissa määritetyn aliaksen kautta.

```
1  Item {
2    id: button
3    signal clicked
4    signal send(string text)
5
6    Component.onCompleted: button.send("ready")
7  }
```

Listaus 4: Signaali QML:ssä

QML:ssä signaali määritellään listauksen 4 mukaisesti. Signaalin voi lähettää tavallisen funktion kutsumisen tapaan, kuten listauksen button.send(”ready”) tekee. [1, Qt QML.]

```
onClicked: {
  //do something
}

function update() {
  //do something
}

Component.onCompleted: {
  instructions.clicked.connect(instructions.update);
}
```

Listaus 5: Signaalin yhdistäminen QML:ssä

QML:ssä signaali voidaan yhdistää kahdella tavalla. Listauksessa 5 clicked yhdistetään suoraan koodiin onClicked-käskyllä, eli listauksen toisen rivin kommentin tilalle voisi JavaScriptillä kirjoittaa signaalin sattuessa suoritettava koodi. Listauksessa näytetään myös toinen tapa, jossa clicked yhdistetään connect-metodilla update-funktioon. [1, Qt QML.]

Component on uudelleenkäytettävä QML-elementti. Yleensä komponentit määritetään komponenttitiedostoina. Tiedostosta tulee automaattisesti komponentti, kun sen nimi alkaa isolla kirjaimella. Komponenttia voi käyttää muualla projektissa heti, jos se on samassa kansiossa.[1, Qt QML.]

```
delegate: ListItem {
  id: listitem
  property Item contextMenu
  ...

  onClicked: {
    ...
    if (!contextMenu)
      contextMenu = defaultcontextmenu.createObject(listitem
          )
    contextMenu.show(listitem)
  }
}

Component {
  id: defaultcontextmenu
  ContextMenu {
    MenuItem {
      text: "Set as a destination"
      ...
    }
    MenuItem {
      text: "Delete"
```

```
23          ...
24        }
25      }
26  }
```

Listaus 6: Komponentin määrittely koodissa

Komponentteja voi määritellä myös koodin sisäisesti. Listauksessa 6 komponentti luodaan propertyyn komponentin createObject-metodilla clicked-signaalissa, jos sitä ei vielä ole olemassa. Tämän jälkeen komponentti näytetään ListItemin lapsena. [1, Qt QML.]

```
1  //MapPackages.h
2  class MapPackages : public QObject {
3    Q_OBJECT
4    Q_PROPERTY(QStringList modulenames READ
        getCurrentModuleNames)
5  ...
6  public slots:
7    void loadMapPackages();
8    void changeMapPackage(int selected);
9    void loadMapModules(int routingindex, int renderingindex,
        int addresslookupindex);
10 signals:
11   void mapPackagesLoaded(QStringList list, int selected,
        QStringList routinglist, QStringList renderinglist,
        QStringList addresslookuplist);
12 private:
13   QStringList getCurrentModuleNames();
14 ...
15 };
16
17 //main.cpp
18 ...
19     qmlRegisterType<MapPackages>("harbour.monav", 1, 0, "
        MapPackages");
20 ...
21
22 //MapPackagesPage.qml
23 ...
24 import harbour.monav 1.0
25 ...
26 MapPackages {
27 ...
28   onMapPackagesLoaded: {
29 ...
30   }
31 }
32 ...
```

Listaus 7: C++-luokan rekisteröinti QML:ssä

QML-kielessä voi käyttää Qt:n ominaisuuksia, jotka ovat määritetty C++:lla. Esimerkiksi listauksen 7 C++:ssa rivillä 11 määritetty signaali voidaan huomioida myös QML:ssä, kuten rivillä 28. Käyttämällä qmlRegisterType-funktiota luokan voi rekisteröidä QML-komponentiksi. Rivillä 24 käytetään import-käskyä liittämään kyseisessä osoitteessa sijaitsevat komponentit käytettäväksi. Listauksen 7 luokassa määritetyt property ja slotit ovat myös samalla tavalla signaalin kanssa QML:ssä käytettävissä. [1, Qt QML.] Jos metodin haluaa saada QML:ssä käytettäväksi, mutta siitä ei tarvitse tehdä slottia, voi sen rekisteröidä Qt:n Metaobject systemiin Q_INVOKABLE-makrolla [1, Qt Core].

2.1.6 Projektitiedostot ja qmake

Qt:ssä on oma projektitiedostomuoto .pro. Projektitiedosto käännetään qmakella ja siitä tulee perinteinen maken Makefile. Makefilellä voi sen jälkeen kääntää ja asentaa ohjelman. [1, QMake Manual.]

```
1  TEMPLATE = app
2  CONFIG += sailfishapp release
3  QT = core gui quick positioning
4
5  OTHER_FILES += \
6      Merkintanakyma.qml \
7      Asetuksetnakyma.qml \
8      Main.qml
9
10 SOURCES += \
11     main.cpp \
12     Model.cpp \
13     Paintteri.cpp
14
15 HEADERS += \
16     Model.h \
17     Paintteri.h
```

Listaus 8: Qt:n projektitiedosto

Qt:n projektitiedostoissa käytetään muuttujia määrittelemään kääntäminen. Listauksessa 8 on esimerkkiprojektitiedosto Qt-ohjelman kääntämiseen. TEMPLATE-muuttuja määrittää, käännetäänkö ohjelma (app) vai kirjasto (lib). CONFIG-muuttujassa määritetään yleisiä projektin asetuksia, mm. onko käännös debug- vai release-käännös. QT-muuttuja määrittää Qt:tä koskevia asetuksia, mm. käytettävät moduulit. OTHER_FILES-muut-

tujalla voi lisätä asennusvaiheeseen käännöksen ulkopuolisia tiedostoja. SOURCES- ja HEADERS-muuttujilla annetaan käännettävät lähdekooditiedostot kääntämistä varten. [1, QMake Manual.]

```
1 TEMPLATE = subdirs
2 SUBDIRS = sailfishclient plugins
3 sailfishclient.depends = plugins
4 plugins.file = plugins/client_plugins.pro
```

Listaus 9: Projektitiedosto määritetty kääntämään useampi projekti

Projektitiedoston voi myös määrittää kääntämään useampi projekti kerrallaan. Esimerkiksi näin voi kääntää ohjelman tarvitsemat kirjastot ennen kuin itse ohjelma käännetään. Tätä varten käytetään TEMPLATE-muuttujassa subdirs-valintaa. Tällöin projektitiedostossa ei voida kääntää itse ohjelmaa. SUBDIRS-muuttujaan listataan alikansiot, jotka sisältävät käännettävän projektin. Käännettävän projektin projektitiedoston nimi on hyvä olla sama kuin kansion nimi. Kuitenkin, jos nimi on eri, file-määreellä voi valita käännettävän projektitiedoston, kuten listauksessa 9. depends-määreellä voi kertoa, jos jokin projekti on riippuvainen toisesta. subdirs-projektitiedostoa käyttäessä se ajaa siinä asetetut projektitiedostot qmakella ja sen jälkeen tuotetun Makefilen makella. [1.]

2.1.7 Laajennusten luominen Qt:ssä

Qt:ssä voi luoda laajennuksia rajapintoja vasten. Laajennukset lisäävät ohjelman toimintoja, eli laajennuksen rajapintaa vastaavan luokan ohjelmakoodi ladataan ohjelmaan. Ne ovat mahdollista ladata ohjelman ajon aikana. [1, Qt Core.] Vastaavan toiminnan mahdollistaa POSIX-järjestelmissä käytetty rajapinta dlopen-funktion ja siihen liittyvien funktioiden avulla. Windowsissa vastaavan toiminnan mahdollistaa LoadLibrary-funktio [25]. Qt:ssä kuitenkin laajennusten lataaminen tapahtuu C++:n tyylisesti kokonaisen rajapinnan määrittävän luokan lataamisella. Laajennuksesta luodaan tavallinen kirjasto (.so/.dll). [1, Qt Core.]

```
1 class IFind {
2 public:
3   virtual ~IFind() {}
4
5   virtual QString findName(QString name) = 0;
6 };
7 Q_DECLARE_INTERFACE(IFind, "ourprogram.IFind/1.0")
```

Listaus 10: Laajennuksen rajapinta

Listauksessa 10 luodaan rajapintaluokka ja määritetään sille puhdas virtuaalimetodi. Q-_DECLARE_INTERFACE-makrolla ilmoitetaan rajapinnasta Qt:lle antamalla sille luokan nimi ja yksilöllinen IID.

```
1 QPluginLoader pl("libfind.so");
2 IFind *f = qobject_cast<IFind *>(pl.instance());
```

Listaus 11: Laajennuksen lataaminen pääohjelmassa

QPluginLoader-luokkaa käytetään listauksessa 11 lataamaan laajennus ohjelmaan. QPluginLoader lataa automaattisesti laajennuksen, kun instance-metodia kutsutaan. Jos lataaminen ei onnistu, osoittimen arvoksi tulee 0. qobject_cast muuttaa laajennuksen tyypin QObjectista rajapinnan tyypiksi, jos mahdollista. Jos tyypin vaihto ei ole mahdollista, osoittimen arvoksi tulee jälleen 0. [1, Qt Core.]

```
1 class Find : public QObject, public IFind {
2   Q_OBJECT
3   Q_INTERFACES(IFind)
4   Q_PLUGIN_METADATA(IID "ourprogram.IFind/1.0")
5 public:
6   QString findName(QString name);
7 };
```

Listaus 12: Laajennuksen luominen

Luokka Find toteuttaa IFind-rajapinnan listauksessa 12. Rajapintaa käyttävällä luokalla tulee olla Q_INTERFACES-makro, jolla kerrotaan Qt:lle käytettävä rajapinta. Q_PLUGIN-_METADATA-makrolla kerrotaan Qt:lle, että tätä luokkaa käytetään laajennuksena. Laajennuksen rajapinta annetaan IID:nä. Laajennuksella voi olla vain yksi rajapinta. Jos laajennuksen täytyy toteuttaa useampia rajapintoja, voidaan laajennusta käyttää luomaan olio, jolla on useampia rajapintoja. [1, Qt Core.]

Laajennuksen kääntämiseen käytetään erillistä .pro-tiedostoa [1, Qt Core].

Laajennuksen voi myös liittää ohjelmaan staattisesti, jolloin sitä ei tarvitse erikseen ladata. Tällöin ohjelman joutuu kääntämään uudellen, kun laajennusta muutetaan. Staattista laajennusta lisättäessä projektin .pro-tiedostoon täytyy lisätä “CONFIG += static” ja lisäämällä laajennukset käyttäen .pro-tiedostossa “LIBS += -Lplugins -lfind”. -Lplugins osoittaa plugins-kansioon, jossa laajennukset on ja -lfind lisää kyseisen laajennuksen. Itse ohjelmassa täytyy käyttää Q_IMPORT_PLUGIN-makroa, jolle annetaan laajennuksen nimi. Nimi on sama kuin laajennuksen TARGET-määrityksessä käytetty projektin nimi. [1, Qt

Core.] Kuitenkin oman kokeilun perusteella nimi on pluginina toimivan luokan nimi, koska Q_IMPORT_PLUGIN-makro tekee viittauksen qt_static_plugin_NIMI-symboliin, jossa pluginiin tulee NIMI-kohdan tilalle luokan nimi. QPluginLoader::staticPlugins-metodilla saadaan lista staattisesti liitetyistä plugineista [1, Qt Core].

2.1.8 Ohjelman asetukset Qt:ssä

QSettings-luokka on Qt:ssä ohjelman asetuksien tallentamista auttava luokka. Se tallentaa asetukset alustariippumattomasti, joten luokan käyttäjän ei tarvitse huolehtia, mihin asetukset kuuluu alustassa tallentaa. Esimerkiksi Windowsissa asetukset tallennetaan Windowsin rekisteriin ja Mac OS X:ssä asetukset tallennetaan XML-tiedostoon. QSettingsiä on myös mahdollista laajentaa omilla tiedonsäilytyslaajennuksilla, jos asetukset halutaan tallentaa omalla tavalla. [1, Qt Core.]

Jokaisella tiedolla asetuksissa on avain ja arvo. Avaimen tyyppi on QString ja arvon tyyppi on QVariant. QVariant voi sisällyttää yleisimmät Qt:ssä tuetut tietomuodot. Arvoksi voi siis tallentaa mitä tahansa tyyppiä, jota QVariant tukee. Lisäksi omia tyyppejä on mahdollista tallentaa, kun ne on rekisteröity Qt:ssä tyypeiksi. [1, Qt Core.]

```
1  void Bookmarks::addBookmark(QString name) {
2    QSettings settings( "MoNavClient" );
3    settings.beginGroup( "Bookmarks" );
4
5    //luetaan names-avaimen arvo
6    QStringList names = settings.value( "names" ).toStringList
         ();
7    names.push_back( name );
8    //kirjoitetaan arvo takaisin names-avaimeen
9    settings.setValue( "names", names );
10
11   UnsignedCoordinate pos = RoutingLogic::instance()->target
         ();
12   //asetettavan avaimen nimeksi tulee "luku.coordinates.x",
         jossa luku on järjestysnumero. arvoksi asetetaan
         RoutingLogicilta saatu sijainti
13   settings.setValue( QString( "%1.coordinates.x" ).arg(
         names.size() - 1 ), pos.x );
14   settings.setValue( QString( "%1.coordinates.y" ).arg(
         names.size() - 1 ), pos.y );
```

```
15 }
```

Listaus 13: Kirjanmerkin lisääminen MoNav-ohjelmassa QSettingsiä käyttäen

QSettingsille voidaan luodessa antaa kaksi argumenttiä: organisaation nimi ja ohjelman nimi. Tällöin esimerkiksi Unixeissa organisaation kaikkien ohjelmien asetukset tallennetaan samaan kansioon. Listauksessa 13 QSettings luodaan käyttämällä vain organisaation nimeä, jolloin organisaatio saa itselleen vain yhden asetustiedoston. Tässä ohjelmassa organisaation nimenä käytetään ohjelman nimeä. [1, Qt Core.]

Asetustiedostossa on asetusryhmiä, jotka erotellaan nimellä. Ryhmän käyttäminen aloitetaan beginGroup-metodilla ja lopetetaan endGroup-metodilla. beginGroup-metodi saa argumentikseen ryhmän nimen. Ryhmän voi myös määrittää samalla, kun käytetään avaimen nimeä. Nimi annetaan silloin muodossa “ryhmä/avain”. Asetusarvon voi lukea value-metodilla, joka saa arvokseen avaimen nimen. Arvon voi asettaa setValue-metodilla, jolle annetaan argumentteinä avaimen nimi ja arvo. [1, Qt Core.]

QSettings-luokka ei kirjoita asetuksia heti tiedostoon, vaan pitää tiedon hetken muistissa. Muistissa olevan tiedon voi kirjoittaa heti kutsumalla sync-metodia. [1, Qt Core.]

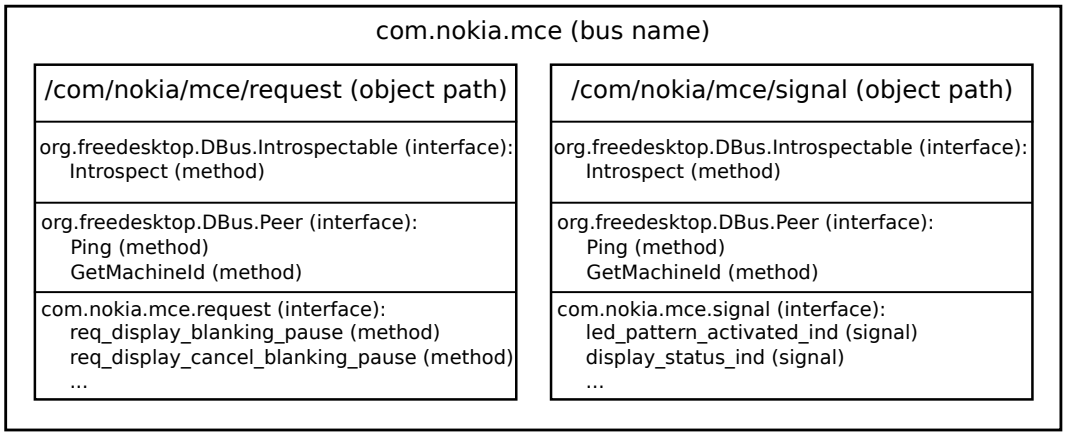
```
1 int count = settings->beginReadArray("trackqueue");
2 for (int i = 0; i < count; i++) {
3   settings->setArrayIndex(i);
4   trackqueue.push_back(settings->value("name").toString());
5 }
6 settings->endArray();
```

Listaus 14: QSettingsin arrayn käyttö

QSettings tukee myös QVariantin tukemien tyyppien lisäksi omaa array-tietomuotoa. Jokaisella arrayn indeksillä voi olla useampia avaimia ja arvoja. Listauksessa 14 jokaisella indeksillä on vain name-arvo. Arrayn lukeminen aloitetaan beginReadArray-metodilla, jolle annetaan argumenttinä arrayn nimi. Metodi palauttaa arrayn koon. Tällä hetkellä luettavan indeksin voi vaihtaa setArrayIndex-metodilla. Asetuksen arvo luetaan tämän jälkeen tavallisesti value-metodilla. Arrayn lukemisen voi lopettaa endArray-metodilla. [1, Qt Core.]

2.1.9 QtDBus-moduuli ja sen käyttäminen

QtDBus on moduuli, jonka avulla voi tehdä prosessien välistä kommunikointia D-Bus-järjestelmää käyttäen. D-Bus on yleinen UNIX-järjestelmissä. D-Busissa käytetään mm. object patheja, metodeita ja signaaleja sekä interfaceja. Object pathit ovat osoitteita, joilla voidaan osoittaa johonkin olioon. Interfacet määrittävät olion tyypin, eli sen metodit ja signaalit. Metodit käynnistävät jonkin operaation. Signaalit ovat kuunneltavia lähetyksiä, joita olio voi lähettää. [26.]



Kuva 3: Ohjelman com.nokia.mce D-Bus-toiminnot

Ohjelmilla on bus name, joka erottaa ohjelmat toisistaan. Se voi olla kaksoispisteellä alkava ja numeroista koostuva yksilöivä nimi tai kirjaimista koostuva yleinen ohjelman määrittämä nimi. Yleensä D-Busilla on system- ja session-ilmentymät, joita käytetään erikseen. [26.]

```
1 QDBusConnection bus = QDBusConnection::systemBus();
2 QDBusInterface *interface = new QDBusInterface("com.nokia.
     mce", "/com/nokia/mce/request", "com.nokia.mce.request",
     bus, this);
3 if (navigation) {
4   interface->call("req_display_blanking_pause");
5 } else {
6   interface->call("req_display_cancel_blanking_pause");
7 }
```

Listaus 15: Näytön pitäminen päällä QtDBusin avulla

Listauksessa 15 on yksinkertainen esimerkki, kuinka puhelimen kuvassa 3 näytetylle mce-palvelulle lähetetään käsky pitäämään näyttöä päällä. systemBus-metodilla luodaan yhteys D-Bus-palveluun. QDBusInterfacella määritetään paikallinen olio edustamaan etä-

oliota. Muodostimen parametreinä ovat etäohjelman bus name, käytettävän olion object path, käytettävä interface ja äsken luotu yhteys. QDBusInterfacen call-metodia käyttämällä voidaan kutsua etäolion metodia. [1, Qt D-Bus.]

2.1.10 Qt Resource Systemin esittely

Qt Resource System on tarkoitettu tallettamaan binäärimuotoisia tiedostoja ohjelman suoritustiedostoon. Talletettavat tiedostot voivat olla mm. kuvia. Tiedostot talletetaan ohjelmaan käännösvaiheessa. [1, Qt Core.]

```
1 <RCC>
2     <qresource>
3         <file>images/map.png</file>
4         <file>images/route.png</file>
5     </qresource>
6 </RCC>
```

Listaus 16: .qrc-tiedostossa määritetty lisättävät kuvat

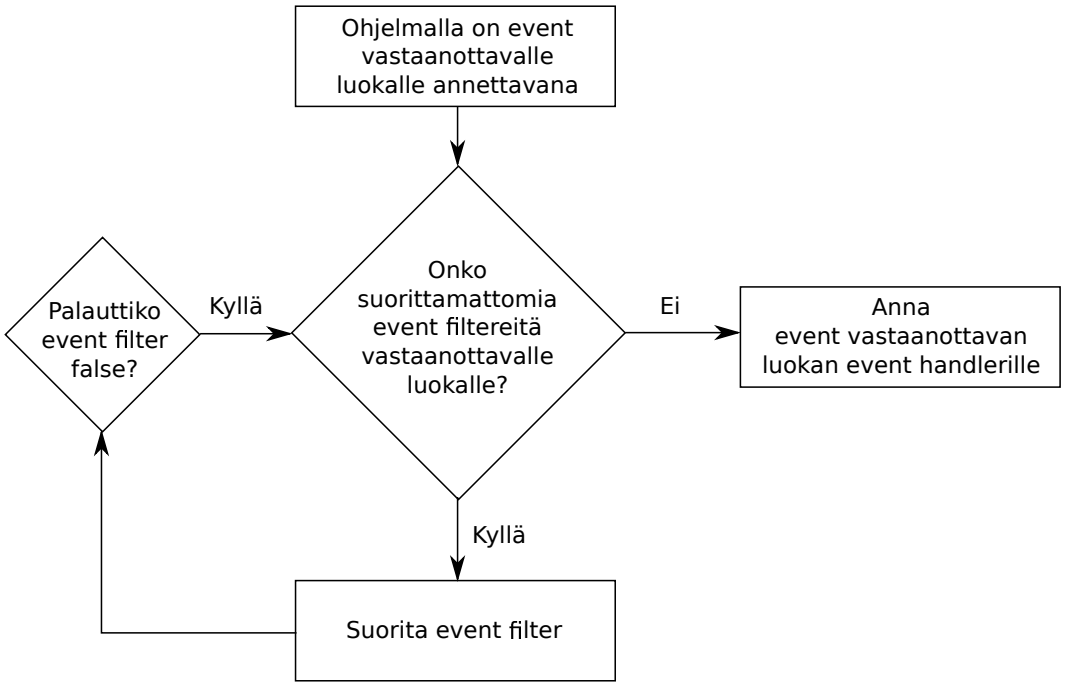
Ohjelmaan talletettavat tiedostot määritetään .qrc-tiedostossa. Se on XML-pohjainen tiedosto, jossa on annettu lisättävien tiedostojen polut. Listauksessa 16 on esimerkki .qrc-tiedostosta, jossa lisätään ohjelmaan kaksi kuvaa. [1, Qt Core.]

Kuvia voi käyttää ohjelmassa samalla polulla kuin .qrc-tiedostossa annettu polku. Siihen on kuitenkin lisättävä :/-etuliite tai qrc://-etuliite URL:nä käytettäessä. Esimerkiksi “QImage(”:/images/map.png”)” luo olion ohjelmaan lisätystä kuvasta. [1, Qt Core.]

Lisättävä .qrc-tiedosto määritetään projektitiedostossa RESOURCES-muuttujalla. Esimerkiksi “RESOURCES += images.qrc” lisää images.qrc-tiedostossa määritetyt tiedostot ohjelmaan. [1, Qt Core.]

2.1.11 Eventien hallitsemimnen

Eventit ovat tapahtumia, jotka voivat tulla itse ohjelmasta tai ohjelman ulkopuolelta. Eventit kohdistuvat QObject-luokkaan ja kyseinen luokka voi käsitellä sen. Kun event tapahtuu, Qt luo jonkin QEvent-luokan perivän luokan. Eventejä voi olla esimerkiksi QResizeEvent, kun ikkunan kokoa muutetaan tai QMouseEvent, kun hiirtä painetaan. Jokaisella eventillä on Event::Type-tyyppi, josta voi nopeesti saada selville, minkä tyyppinen event on. [1, Qt Core.]



Kuva 4: Eventin käsittelemisen järjestys

Eventit käsitellää Event Handlereilla. QObjectilla on event-metodi, joka on yleinen eventien käsittelijä. Se siis vastaanottaa eventin, kun se QObjectille tulee. event-metodin voi ylikirjoittaa omassa ohjelmassa tarvittaessa. [1, Qt Core.]

```
1 bool ActivationChangeEventFilter::eventFilter(QObject *obj,
    QEvent *event) {
2   if (event->type() == QEvent::ApplicationActivate) {
3     emit applicationActive();
4     return false;
5   } else {
6     return QObject::eventFilter(obj, event);
7   }
8 }
9
```

```
10 QCoreApplication::instance()->installEventFilter(
       activation_change_event_filter);
```

Listaus 17: Event filter asetettu vastaanottamaan event

Toiset oliot voidaan myös asettaa kuuntelemaan muiden olioiden eventejä. Tätä varten ovat event filterit. Event filter asetetaan oliolle installEventFilter-metodilla, esimerkiksi: “eventin_saava_luokka->installEventFilter(luokka_jolla_event_filter)”. eventFilter-metodi käsittelee event filterin saaman eventin. Se saa argumentteinä eventin saavan luokan ja itse eventin. eventFilter-metodi saa eventin ennen kuin sen oikeasti saava luokka sen saa, kuten kuvassa 4 on näytetty. Metodissa voidaankin palautusarvona antaa true, jos halutaan estää eventin saapuminen sen oikeaan luokkaan. [1, Qt Core.] Listauksessa 17 on luokka, jolla on eventfilter ja se asetetaan kuuntelemaan pääohjelman eventejä rivillä 10 olevalla kutsulla.

Eventin voi myös itse lähettää QCoreApplication-luokan sendEvent-metodilla. Metodi saa argumentteinä vastaanottavan QObject-luokan ja eventin QEvent-luokan. sendEvent-metodi odottaa niin kauan, että event on käsitelty. postEvent-metodi lisää eventin vain jonoon. [1, Qt Core.]

2.2 Sailfish Silica -komponenttien esittely

2.2.1 Yleistä SailfishOS-käyttöjärjestelmästä

SailfishOS on Jolla-puhelimen käyttämä käyttöjärjestelmä. Sen käyttölittymän ohjaaminen perustuu eleisiin. Sailfishissä on täydellinen moniajo, ja sen käyttöliittymästä näkee avoinna olevat ohjelmat helposti yhdellä kertaa. [27.] Käyttöjärjestelmän käyttöliittymä pohjautuu Qt:hen ja on kirjoitettu suurelta osin QML:llä [28]. Se käyttää perinteistä Linuxia taustalla ja pohjautuu Mer-projektiin [29]. Merissä on taustapalvelut mobiilikäyttäjärjestelmän vaatimuksille ja on avoimen lähdekoodin projekti [30].

Sailfishille kehitetään ohjelmia sen oman SDK:n avulla.

2.2.2 ApplicationWindow

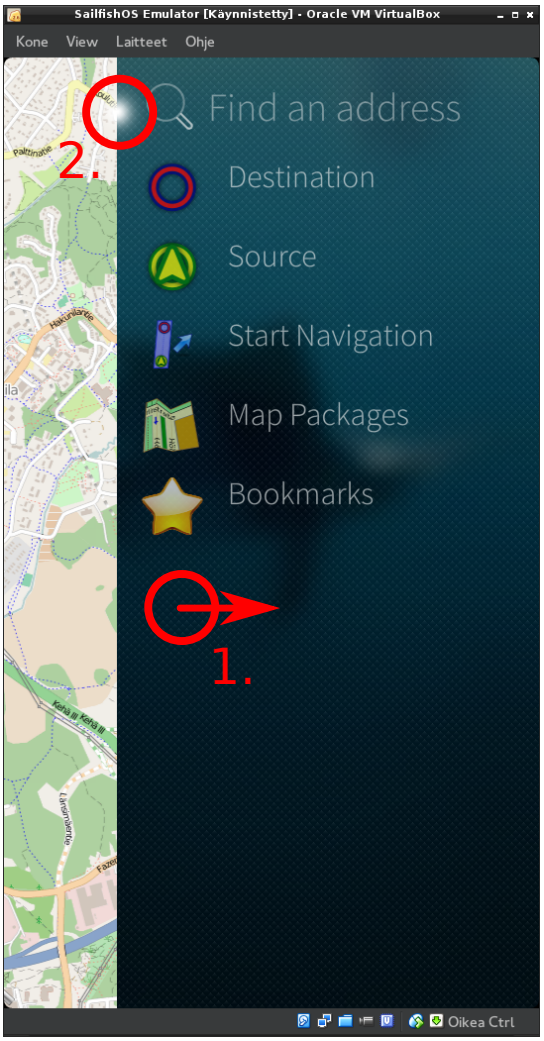
ApplicationWindow-komponenttiä käytetätään pääkomponenttinä Sailfish-ohjelmissa. Se siis on ylimpänä elementtihierarkiassa. ApplicationWindow suoritetaan ensimmäisenä QML--ohjelmasta. [31.]

ApplicationWindow tietää ensimmäisen avattavan sivun (Page-komponentti) ohjelmassa ja lataa sen. Ensimmäinen sivu lisätään ApplicationWindowin PageStackiin, joka pitää kirjaa sivuista. ApplicationWindow lataa myös ohjelmalle kannen, joka näkyy, kun ohjelma on piennennettynä. Ladattava sivu asetetaan initialPage-propertyllä ja ladattava kansi cover-propertyllä. [31.]

ApplicationWindow hallitsee näkymän kääntämistä, ja käyttäjä voi valita mahdolliset kääntämiset allowedOrientations-propertyn avulla. Kun ohjelma käyttää PageStackia, Pagen vastaavia ominaisuuksia tulisi käyttää ApplicationWindowin näkymän kääntämisen sijasta. [31.]

2.2.3 PageStack, Page ja PageHeader

Sailfish-käyttöjärjestelmässä käyttöliittymässä liikutaan eleillä. Useimmissa näkymissä voidaan liikkua seuraavaan tai edelliseen näkymään pyyhkäisemällä sormella oikealle tai vasemmalle lähes mistä ruudun kohdasta tahansa. Kuvassa 5 kohdassa 1 on käyttäjä siirtymässä edelliselle sivulle pyyhkäisemällä sormella oikealle. Tätä tapaa käytetään useimmissa Sailfish-käyttöjärjestelmän käyttöliittymissä navigointiin. Sivun yläreunassa on vaihtoehtoisesti myös painikkeet siirtymiselle. Kuvassa 5 kohdassa 2 näkyy edelliselle sivulle siirtymistä varten painike valkoisena pallona. [31.]



Kuva 5: Edelliselle Pagelle siirtyminen PageStackissa

PageStack pitää yllä kirjaa näytettävistä sivuista. PageStackin pinossa päälimmäisenä oleva sivu näytetään käyttäjälle. Sivuja voidaan lisätä ja poistaa pinosta push- ja pop-metodeilla. Päälimmäisen sivun yläpuolella voi olla myös liitettyjä sivuja, joille käyttäjä voi halutessaan siirtyä. Liitettyjä sivuja voi lisätä ja poistaa pushAttached- ja popAttached-metodeilla. [31.]

```
1 var mainviewpage = pageStack.find(function (page) {
2   return page.objectName == "mainviewpage";
3 });
4 mainviewpage.mainview.instructionsLoaded.connect(
      instructions.update);
```

Listaus 18: PageStackin find-metodin käyttö

PageStackista voi etsiä Pageja omalla etsintäfunktiolla. Etsintään käytetään PageStackin find-metodia. find-metodi käy Paget läpi pinon päältä alas ja kutsuu niille find-metodille annettua funktiota. find-metodi palauttaa ensimmäisen Pagen, jolle metodille annettu

funktio palauttaa true. Listauksessa 18 findille annettu funktio palauttaa truen, kun Pagen objectName on mainviewpage. [31.]

Page on itse näkymä, joka sisältää elementtejä. Elementit näkyvät, kun Page on PageStackin päällimmäisenä eli näkyvissä. [31.]

Jokaisella Pagella on omat näkymän kääntämisen asetukset. allowedOrientations-propertyn avulla voidaan valita, missä asennoissa näkymä näytetään. [31.]

```
1 onStatusChanged: {
2   if (status == PageStatus.Activating) {
3     view.update(bookmarks.bookmarks);
4   }
5 }
```

Listaus 19: Pagen signaalien käyttö MoNav-ohjelmassa

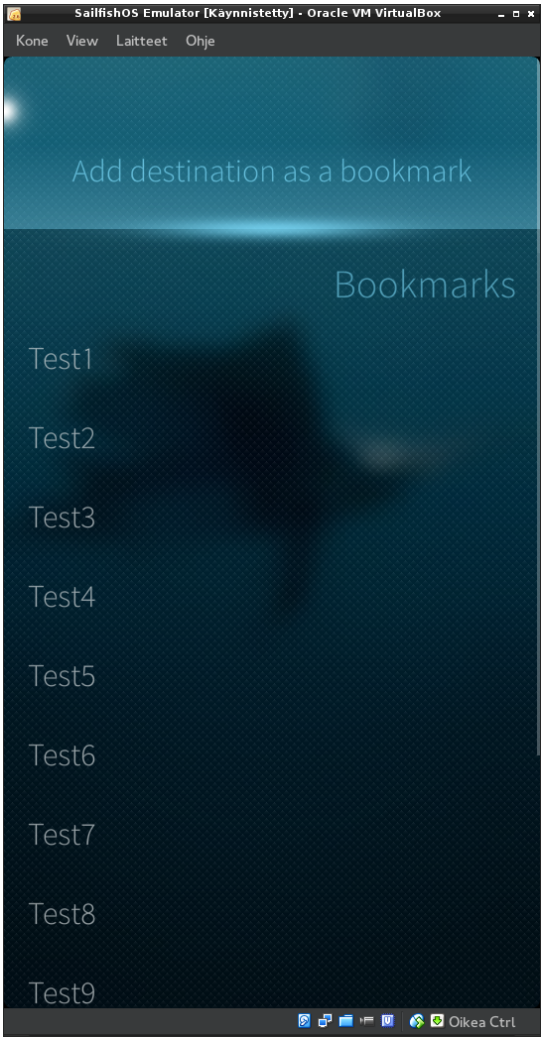
Pagella on status-property, joka muuttuu, kun sivua vaihdetaan. Pagen status voi olla aktiivinen, ei näkyvissä ja siirtymät animaationa niiden välillä. Animaationa siirtymisen status ei kuitenkaan ole vielä silloin, kun sormella raahaa ja vilkaisee edelliselle sivulle, vaan sitten, kun sormen päästää irti ja animaatio alkaa. [31.] Page lähettää onStatusChanged-signaalin, kun status vaihtuu. Signaalin tullessa voidaan tarkistaa Pagen status ja tehdä sitä vastaava toiminto, kuten listauksessa 19. onStatusChanged-signaalia ei ole tällä hetkellä dokumentoitu Sailfish Silican dokumentaatiossa. [32.]

PageHeader-komponentillä voi lisätä Pagelle otsikon. PageHeader on tarkoitus lisätä Pagen ylälaitaan. [31.]

2.2.4 SilicaFlickable-, PullDownMenu- ja ScrollDecorator-komponentit

Vieritettäviä näkymiä varten Sailfish Silica sisältää SilicaFlickable-komponentin. SilicaFlickable periytyy QtQuick-kirjaston Flickable-komponentistä ja näin sillä on sen ominaisuudet. Näkymää voi vierittää sormella liikuttamalla. [31.]

Vieritettävän sisällön koko on laskettava ja asetettava Flickable-komponentin contentWidth- ja contentHeight-propertyihin, jotta sisällöstä tulee vieritettävä. Sisällön nykyinen kohta Flickable-komponentin sisällä on contentX- ja contentY-propertyissä. Flickable-komponentin vierityksen voi ottaa pois käytöstä käyttäjältä interactive-propertyä käyttäen. [31.]



Kuva 6: PullDownMenun avaaminen SilicaFlickablea liikuttamalla

SilicaFlickableen voi lisätä alas vedettävän valikon käyttäen PullDownMenu-komponenttia ja lisäämällä se SilicaFlickablen lapseksi. PullDownMenu toimii niin, että SilicaFlickablen ollessa yläreunassa, reunasta voi vierittää vielä lisää ja valikko aukeaa. Kuvassa 6 valikko on avattu sormella listaa alas vierittämällä. Valikossa on eri valintoja, joita voi valita liikuttamalla SilicaFlickablea ylös tai alas sormea painallettuna pitäen. Kun sormen päästää irti, valinta suoritetaan. Valinnat lisätään PullDownMenuun MenuItem-komponentin avulla. [31.]

```
1 SilicaFlickable {
2    id: flick
```

```
3    anchors.fill: parent
4    contentHeight: searchfield.height + findlist.height + view
        .height
5    ...
6  }
7
8  ScrollDecorator { flickable: flick }
```

Listaus 20: ScrollDecorator lisätty SilicaFlickableen MoNav-ohjelmassa

ScrollDecorator lisää SilicaFlickableen vierityspalkit, jotka näyttävät nykyisen sijainnin SilicaFlickablen vieritettävässä sisällössä. SilicaFlickable näyttää pitkää pystysuoraa sisältöä vierittäessä painikkeet nopealle vieritykselle. Painikkeita painettaessa SilicaFlickablen sisältö siirtyy sen ylä- tai alareunaan. Listauksessa 20 näytetään, kuinka ScrollDecoratorille annetaan flickable-propertylle SilicaFlickablen id, jotta ScrollDecorator asettuu oikealle SilicaFlicakblelle. ScrollDecoratoria ei suositella asetettavaksi SilicaFlickablen lapseksi suorituskyvyn vuoksi. [31.]

2.2.5 SilicaListView- ja ContextMenu-komponentit

SilicaListView on tarkoitettu listamaisen tiedon näyttämiseen elementteinä. Se periytyy QtQuickin ListView-komponenteista. ListViewillä on kaksi olennaista propertyä: model ja delegate. [31.]

```
1  model: ListModel {
2    ListElement {
3      itemtext: "Destination"
4      itemicon: "qrc:///images/target.png"
5    }
6    ListElement {
7      itemtext: "Source"
8      itemicon: "qrc:///images/source.png"
9    }
10   ...
11 }
```

Listaus 21: Tap Menun ListModel MoNav-ohjelmassa

SilicaListViewin model-property on tietolähde, josta listan tiedot saadaan. Siihen voidaan asettaa mm. ListModel-elementti. ListModel on yksinkertainen tietosäilö, jonka sisältö on vapaamuotoinen. ListModelin yksi tietoalkio asetetaan ListElement-elementillä. ListElementit määritetään kuten muutkin elementit, mutta sisältää role-määrityksiä propertyjen

sijaan. Listauksessa 21 on määritetty ListModel ja ListElementtejä. ListElementit sisältävät itemtext- ja itemicon-rolet. [31.]

```
1 function update(list, placeNames) {
2   searchresults.clear();
3
4   for (var i = 0; i < list.length; i++) {
5     searchresults.append({itemtext: list[i], itemtext2:
          placeNames[i]});
6   }
7 }
```

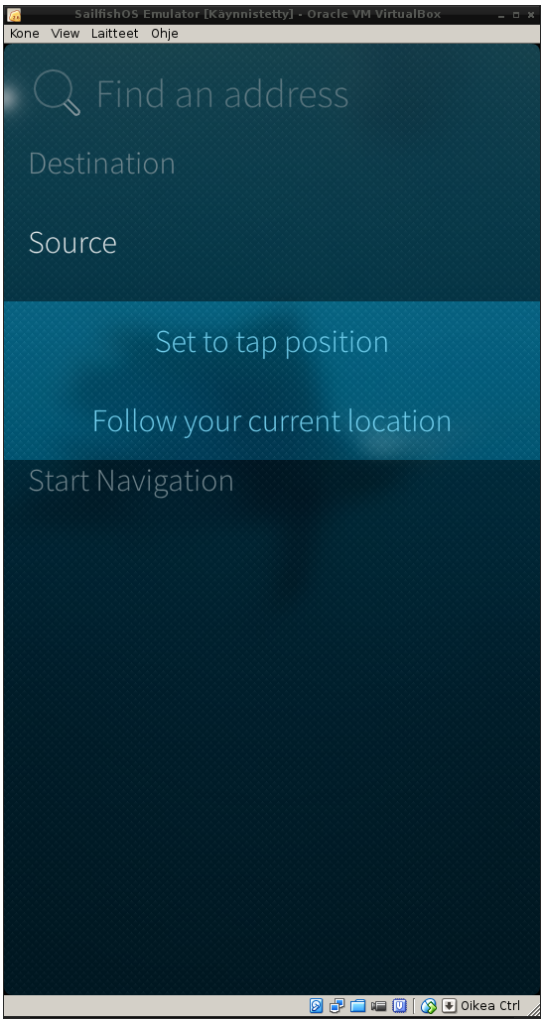
Listaus 22: Tap Menun hakutulosten päivittäminen MoNav-ohjelmassa

ListModelin tietoja voi muuttaa ohjelman ajon aikana sen metodeilla. Listauksessa 22 searchresults on ListModel, joka ensin tyhjennetään clear-metodilla ja sitten lisätään uudet elementit append-metodilla. [31.]

```
1  delegate: ListItem {
2    id: listitem
3    ...
4    Image {
5      id: listitemicon
6      source: itemicon
7      ...
8    }
9
10   Label {
11     text: itemtext
12     ...
13   }
14   ...
15 }
```

Listaus 23: Tap Menun valintapainikkeiden määritys MoNav-ohjelmassa

SilicaListViewin delegate-propertyyn asetetaan komponentti, josta luodaan elementti jokaista modelin alkiota kohden. Jokaista modelin alkiota kohden ruudulla näkyy siis yksi delegaten elementti listamaisessa muodossa. Luotu elementti voi käyttää sen alkion tietoja, josta se on luotu. Esimerkiksi listauksen 23 Image ja Label käyttävät listauksen 21 elementeissä määritettyjä itemtext- ja itemicon-roleja. SilicaListViewissä delegatessa käytetään ListItem-komponenttiä, joka mukautuu Sailfish Silican yleiseen tyyliin. ListItemiä voi käyttää painikkeena käyttäen onClick-signaalia. [31.]



Kuva 7: SilicaListView ja sen ListItemin ContextMenu avattuna

Sailfish Silican käyttöliittymään kuuluu ContextMenu-alivalikko ListItemeille. ContextMenun voi avata esimerkiksi klikkaamalla ListItemiä. Se asetetaan ListItemin menu-propertyyn, joko komponenttinä tai oliona. ListItemin koko muuttuu tarpeen mukaan, kun valikko avataan. ContextMenuun lisätään MenuItemeitä, kuten PullDownMenuunkin. Kuvassa 7 SilicaListViewin ListItemiä ollaan painettu ja ContextMenu on avautun näyttämään lisävaihtoehtoja. [31.]

```
1 id: findlist
2 ...
3 property Item contextMenu
4 ...
5 delegate: ListItem {
6   ...
7   property bool menuOpen: findlist.contextMenu != null &&
        findlist.contextMenu.parent === listitem
8   height: menuOpen ? findlist.contextMenu.height + Theme.
        itemSizeSmall : Theme.itemSizeSmall
9   ...
```

```
10    onClicked: {
11      if (!findlist.contextMenu) {
12        findlist.contextMenu = contextMenuComponent.
              createObject(findlist)
13      }
14      findlist.contextMenu.show(listitem)
15    }
16    ...
17  }
18  ...
19  Component {
20    id: contextMenuComponent
21    ContextMenu {
22      MenuItem {
23        text: "Set as destination"
24        ...
25      }
26      MenuItem {
27        text: "Set as source"
28        ...
29      }
30    }
31  }
```

Listaus 24: Saman ContextMenun näyttäminen ListItemeissä MoNav-ohjelmassa

ContextMenuja voi olla esillä vain yksi samaan aikaan, joten halutessaan samaa ContextMenua voi käyttää jokaiselle MenuItemille. Listauksessa 24 määritetään contextMenuComponent-komponentti. ListItemiä ensimmäisen kerran painettaessa komponentistä luodaan olio createObject-metodilla. ContextMenu näytetään sen show-metodilla. show-metodi liittää ContextMenun alareunan parametrinä saadun elementin alareunaan. Parametrinä saatu elementti tulee myös ContextMenun vanhemmaksi. Jos ContextMenua ei aseteta ListItemin menu-propertyyn, niin ListItemin koko ei automaattisesti muutu, vaan koko määritetään manuaalisesti height-propertyllä. Listauksessa ContextMenua ei aseteta menu-propertyyn, koska yksi ListItem ei yksin omista sitä, vaan ContextMenu on jaettu kaikkien ListItemien kesken. [31.]

2.2.6 ComboBox-elementin käyttäminen

ComboBox on pudotusvalikko, jossa on valittuna yksi valinta monesta vaihtoehdosta. ComboBoxissa lukee label-propertyn teksti ja valitun vaihtoehdon text-property. Vaihtoeh-

dot on talletettu MenuItemeinä ContextMenuun. ContextMenu avautuu kun ComboBoxia painaa. Kuvassa 8 ComboBoxia on painettu ja avautunut ContextMenu näyttää vaihtoehdot uudeksi ComboBoxin arvoksi. [31.]

```
ComboBox {
  ...
  menu: ContextMenu {
    Repeater {
      model: ListModel {
        id: packageselectmodel
      }
      MenuItem { text: itemtext }
    }

    onActivated: {
      mappackages.changeMapPackage(index);
    }
  }

  function update(list, selected) {
    packageselectmodel.clear();

    for (var i = 0; i < list.length; i++) {
      packageselectmodel.append({itemtext: list[i]});
    }
    ...
  }
  ...
}
```

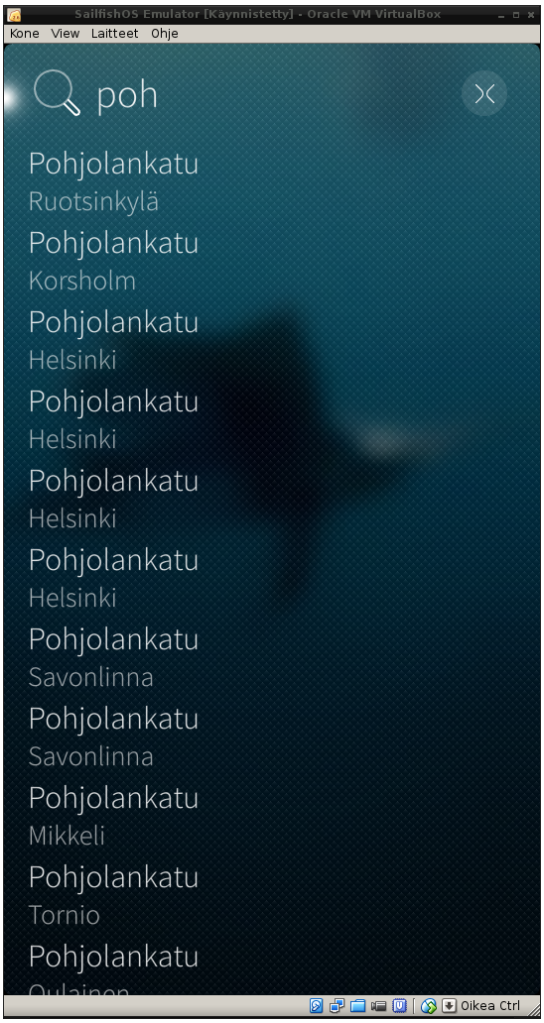
Listaus 25: ComboBoxin valintojen päivittäminen MoNav-ohjelmassa

ComboBoxin sisältöä voi olla usein tarve päivittää. Listauksessa 25 näytetään, kuinka ContextMenun sisältö luodaan Qt Quickin Repeater-tyyppiä apuna käyttäen [1, Qt Quick]. Listauksen update-funktio lisää Repeaterin modeliin vaihtoehdot ja niitä vasten Repeater luo MenuItemeitä. Listauksessa näytetään myös, kuinka ContextMenussa tehty valinta lähettää activated-signaalin. activated-signaalissa on parametrinä valitun MenuItemin index ContextMenussa. [31.]

Kuva 8: ComboBox avattuna näyttäen tällä hetkellä valittua vaihtoehtoa

2.2.7 TextField- ja SearchField-elementin käyttäminen

TextField on Sailfishin tyyliin sopiva kenttä, johon voi kirjoittaa yhden rivin tekstiä. TextField voi näyttää väliaikaisesti oikean tekstin sijasta kentän tarkoituksen näyttävän sijaistekstin, kun oikeaa tekstiä ei ole kenttään vielä kirjoitettu. Tämä sijaisteksti määritetään placeholderText-propertyssä. [31.] Sailfishin dokumentaatiossa ei ole kerrottu signaalista, mutta tekstisisällön vaihtuessa textChanged-signaali lähetetään.



Kuva 9: SearchFieldiin hakusana kirjoitettuna

SearchField-elementti on hakusanan kirjoittamiseen tarkoitettu kenttä. Se perustuu Sailfish Silican TextFieldiin. SearchFieldin eroavaisuudet TextFieldiin ovat suurennuslasin kuva ja painike, joka tyhjentää tekstikentän. Kuvassa 9 käytetään SearchFieldiä kadun nimen hakemiseen hakusanalla. [31.]

2.2.8 Dialog- ja DialogHeader-elementit

Dialog on kuten Page, mutta on tarkoitettu pyytämään käyttäjän syötettä. Dialogin ero Pageen on se, että käyttäjä voi peruta syötteen menemällä sivulta taaksepäin tai hyväksyä syötteen menemällä eteenpäin. Dialog antaa accepted- tai rejected-signaalit sen mukaan, mihin käyttäjä sivulta siirtyy. Dialogilla on myös done-signaali, joka lähetetään kummas-

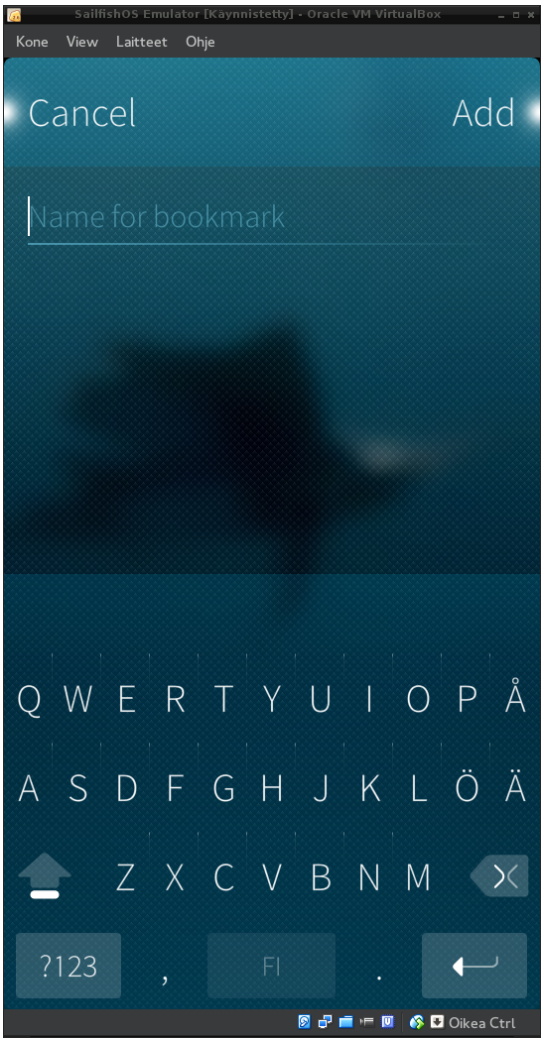
sakin tapauksessa. Käyttäjän tekemän valinnan voi lukea tässä tapauksessa result-propertystä. [31.]

DialogHeader-elementti lisää otsikon Dialogiin. Sillä voi myös muuttaa eteenpäin ja taaksepäin näyttävien nappien tekstiä käyttäen acceptText- ja cancelText-propertyjä. [31.]

```
1  ...
2  onClicked: {
3    pageStack.push(adddialog);
4  }
5  ...
6  Component {
7    id: adddialog
8    Dialog {
9      Column {
10       anchors.fill: parent
11
12       DialogHeader {
13         acceptText: "Add"
14       }
15
16       TextField {
17         id: bookmarkname
18         anchors.left: parent.left
19         anchors.right: parent.right
20         placeholderText: "Name for bookmark"
21       }
22     }
23
24     onDone: {
25       if (result == DialogResult.Accepted) {
26         bookmarks.addBookmark(bookmarkname.text);
27       }
28     }
29   }
30 }
```

Listaus 26: Dialogin luominen komponenttinä MoNav-ohjelmassa

Listauksessa 26 lisätään käyttäjän painaessa uuden dialogin PageStackiin komponentista. Listauksen kuvaama Dialog näkyy kuvassa 10. Dialogilla on myös DialogHeader muuttamaan hyväksymisen tekevälle napille tekstin. Listauksessa tarkastetaan myös, minkä valinnan käyttäjä tekee.



Kuva 10: Dialog avoinna ja tekstikenttä näkyvissä kirjoitusta varten

2.2.9 Sailfish-sovelluksen kansi

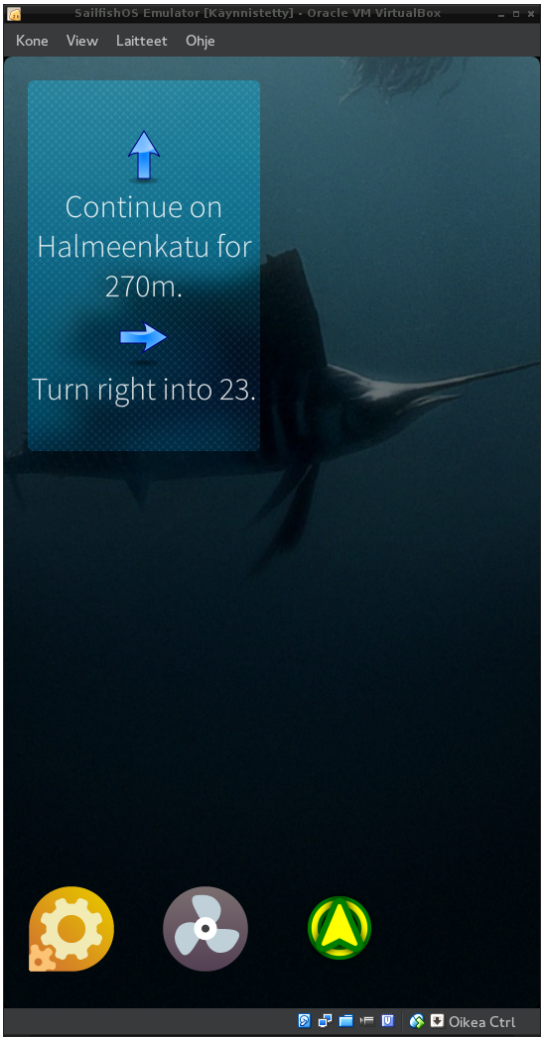
Sailfish-sovelluksilla on kansi, joka näytetään, kun ohjelma on pienennettynä. Pienennettynä ohjelmasta voi näyttää sen tilaa, ja käyttäjä voi tehdä pieniä toimintoja ohjelmalle. Toiminnon voi suorittaa painamalla sormella kantta ja siirtämällä sormen oikealle tai vasemmalle. Kantta painamalla ohjelman voi jälleen suurentaa ja jatkaa sen käyttöä.

```
1 CoverBackground {
2   ...
3   CoverPlaceholder {
4     id: placeholder
5     text: "MoNav"
6     icon.source: "qrc:///images/source.png"
7     visible: !instructions.visible
8   }
9
```

```
10    Column {
11      id: instructions
12      visible: false
13      ..
14    }
15    ...
16  }
```

Listaus 27: Kannen näyttäminen MoNav-ohjelmassa

CoverBackground-elementtiä käytetään yleensä pohjimmaisena elementtinä. Se näyttää käyttöjärjestelmän tyyliin sopivan taustan kannelle. Siihen voi lisätä elementtejä, kuten Sailfishin sivuille normaalisti. Listauksessa 27 lisätään CoverPlaceholder ja Column-elementit kannelle. CoverPlaceholder näytetään esimerkissä, jos ohjelmassa ei ole reittitietoja. Muutoin näytetään Column, jossa kyseisessä ohjelmassa näytetään reittitietoja. CoverPlaceholder näyttää yksinkertaisesti määritetyn kuvan ja tekstin. Kuvassa 11 on näkyvissä kanteen lisätyt reittiohjeet, kun ohjelma on pienennetty. [31.]



Kuva 11: Kansi näyttämässä reittiohjeita

2.3 OpenRepos.net-palvelu

OpenRepos.net on yhteisön ylläpitämä pakettilähdepalvelu. Kehittäjät voivat julkaista omia tai muiden kehittämiä ohjelmia palvelussa. Jokaisella käyttäjällä on oma lähde, jonka muut käyttäjät voivat asentaa. Tämän jälkeen asennetun lähteen ohjelmat tulevat käyttäjälle saataville. [33.]

Käyttäjä hakee ohjelmaa hakusanalla, esimerkiksi ohjelman nimellä tai tarvittavalla ominaisuudella. Ohjelmille annetaan tageja ja ne luokitellaan eri kategorioihin löytämisen helpottamiseksi. Käyttäjillä on mahdollisuus arvostella ja kommentoida ohjelmia.

OpenRepos.net-palvelua voi käyttää puhelimella myös Warehouse-ohjelmaa käyttäen. Ohjelmalla voi selata palvelusta löytyviä ohjelmia sekä asentaa, poistaa ja päivittää niitä. Käyttäjä voi ohjelmalla ylläpitää laitteeseensa asennettuja pakettilähteitä. [34.]

2.4 Jolla Harbour -sivuston vaatimukset ohjelman julkaisulle kaupassa

Jolla Harbour on alusta sovelluksien julkaisuun Jollan kaupassa. Valmiit sovellukset voidaan lisätä sivustolle, jonka jälkeen ne tarkastetaan ja tarkastuksen jälkeen julkaistaan. Harbouriin voi lisätä natiiveja Sailfish-sovelluksia sekä Android-sovelluksia. [35.]

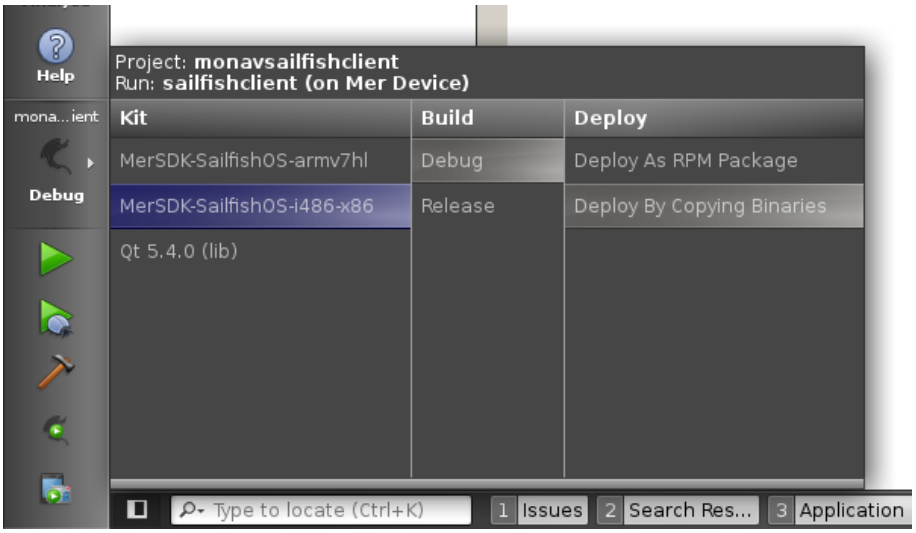
Harbouriin ohjelmaa lisättäessä tulee ottaa huomioon joitakin vaatimuksia ohjelmalta. Vaatimukset liittyvät mm. ohjelman nimeämiseen, sen sijaintiin tiedostojärjestelmässä ja mitä kirjastoja ohjelma voi käyttää. Ennen Harbouriin lähettämistä ohjelma kannattaa testata Sailfish SDK:n mukana tulevalla testaustyökalulla, joka ilmoittaa, mitä korjattavaa ohjelmassa on. Ohjelma löytyy Qt Creatorista. Ohjelman tulos ei kuitenkaan kerro täysin, hyäksytäänkö ohjelma julkaistavaksi. [36.]

Ohjelman tulisi noudattaa ainakin seuraavia vaatimuksia:

- Ohjelman paketin, kansion ja suoritustiedoston nimi tulee alkaa “harbour-”-etuliitteellä.
- Suoritustiedosto on asennettava /usr/bin-kansioon.
- .desktop-tiedosto, jonka avulla ohjelma lisätään ohjelmavalikkoon, lisätään /usr/share/applications/harbour-ohjelma.desktop-polkuun.
- Ohjelman kuvake, jonka koko on 86x86, lisätään polkuun /usr/share/icons/hicolor/86x86/apps/harbour-ohjelma.png.
- Muut ohjelman käyttävät datatiedostot tulee laittaa polkuun /usr/share/harbour-ohjelma.
- Ajon aikaiset tiedot tallennetaan käyttäjän kotikansioon käyttämällä XDG-määritysten mukaisia polkuja. Tämän polun alle ohjelmalla tulee olla vielä oma kansio, joka luodaan ohjelman ajon aikana. Esimerkiksi XDG_DATA_HOME-ympäristömuuttuja kertoo polun ohjelman tietojen tallettamiseen.
- Sailfishissä on vain tietyt kirjastot ohjelman käytettävissä, mm. Qt ja SDL2. Ohjelma voi asentaa omia kirjastoja, mutta ne täytyvät olla polussa /usr/share/harbour-ohjelma/lib.
- Useimmat Qt:n moduulit ovat Harbourin ohjelmien käytettävissä, mutta esimerkiksi QtWidgets- ja QtPositioning-moduulit eivät ole käytettävissä.
- Omia QML-moduuleita voi lisätä ohjelmaan, mutta ne täytyy nimetä tyyliin harbour-.ohjelma.moduuli. [36.]

2.5 Sailfish SDK ja emulaattorin / puhelimen käyttö Qt Creatorin kanssa

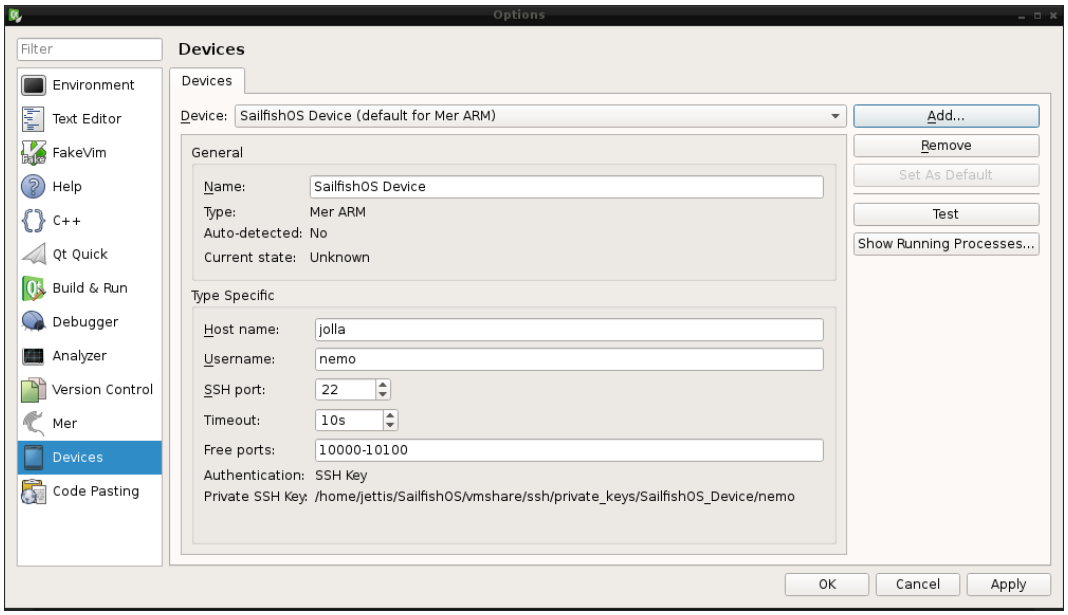
Sailfishin SDK:ta tarvitaan kääntämään ohjelmia SailfishOS:lle. Se ladataan yhtenä pakettina, jonka mukana tulee Qt- ja Sailfish-kirjastot ohjelmien kääntämiseen. Käännösympäristö toimii VirtualBoxissa yhteensopivuuden takaamiseksi eri käyttöjärjestelmien välillä. SDK:ssa käytetään Scratchbox2:ta ohjelmien käännösympäristöjen hallintaan. Paketin mukana tulee Qt Creator -kehittämisympäristö, jonka avulla on helppo hallita myös SDK:n toimintaa. Ohjelmia voi ajaa helposti mukana tulevalla emulaattorilla tai suoraan oikealla laitteella. [37.]



Kuva 12: Alustan valinta ja painikkeet kääntämisen ja SDK:n hallintaan

Ennen ohjelman kääntämistä SDK täytyy käynnistää. SDK käynnistyy painamalla kuvassa 12 näkyvää nappia Qt Creatorin vasemmasta alareunasta. Koska SDK toimii VirtualBoxissa, sen käynnistäminen vaatii Linuxissa vboxdrv-kernelmoduulin lataamista.

Ohjelman voi kääntää eri alustoille. Emulaattorilla ajettaessa ohjelma on käännettävä x86-arkkitehtuurille ja laitteelle käännettäessä sen omalle arkkitehtuurille. Halutun käännöksen tyypin ja arkkitehtuurin voi valita kuvassa 12 avoinna olevasta valikosta.



Kuva 13: Laitteen asetukset

Oman laitteen voi lisätä Qt Creatorin Options-valikosta. Laitteeseen otetaan yhteys SSH:lla, joten se täytyy laittaa laitteen kehitystilan asetuksista päälle. Kuvassa 13 on lisätty laite Options-valikossa ja käännetty ohjelma voidaan nyt ajaa siinä. Emulaattori toimii kuten erillinen laite ja Qt Creator ottaakin siihen yhteyden samalla tavalla SSH:n avulla.

Ohjelmia voi kääntää myös ilman Qt Creatoria käynnistämällä SDK suoraan VirtualBoxista. SDK:n käynnistyttyä, siihen voi ottaa yhteyden SSH:lla. Esimerkiksi “ssh -p 2222 -i /SailfishOS/vmshare/ssh/private_keys/engine/mersdk mersdk@localhost”. [37.]

SDK:ssa kotikansioon on share-kansioon liitetty nykyisen käyttäjän kotikansio. Komennolla “sb2-config -l” Scratchbox tulostaa mahdolliset kohdevaihtoehdot. Scratchboxilla voi suorittaa haluamansa komennon scratchboxin sisällä käyttäen kyseistä kohdevaihtoehtoa esimerkiksi komennolla “sb2 -t SailfishOS-armv7hl komento”. Ohjelman kääntämiseen käytettävät komennot voisivat olla esimerkiksi “sb2 -t SailfishOS-armv7hl qmake monavsailfishclient.pro” ja “sb2 -t SailfishOS-armv7hl make”.

mb2 on Qt Creatorin käyttämä työkalu paketin luomista varten. Sen avulla voi myös itse helposti luoda paketin. Esimerkiksi “mb2 -t SailfishOS-armv7hl build” kääntää ohjelman ja luo siitä RPM-paketin automaattisesti alusta loppuun.

2.6 Sovittaminen Qt4:stä Qt5:een

Qt4 ja Qt5 eivät ole yhteensopivia. Muutokset eivät ole suuria, mutta joitakin muutoksia koodiin joutuu tekemään. Oleellisia muutoksia ovat mm. Qt widgetsin muuttuminen omaksi moduuliksi ja vanhemman laajennusjärjestelmän poistuminen käytöstä. [38.]

Qt4:ssä Qt GUI -moduuli sisältää Qt widgetsit. Qt5:ssä ne ovat siirtyneet omaan Qt Widgets -moduuliin ja Qt GUI -moduuli on enää yleiseen ikkunointiin ja piirtämiseen tarvittava moduuli. Tämän vuoksi koodissa on include-määritykset muutettava QtGui-otsikkotiedos-

tosta QtWidgets-otsikkotiedostoksi. Projektitiedostoon on myös lisättävä “QT += widgets”. [38.]

Qt5:ssä laajennukset määritetään eri tavalla kuin Qt4:ssä. Qt4:ssä käytetty Q_EXPORT-_PLUGIN2-makro ei ole enää käytössä. Qt5:ssä käytetään Q_PLUGIN_METADATA-makroa laajennuksen määrittelyyn. [38.]

Myös pienempiä muutoksia on, jotka on otettava huomioon. QStringin toAscii-metodi on poistunut käytöstä ja korvautuu toLatin1-metodilla Qt5:ssä [38]. QtMsgHandler-funktio-osoitintyyppi on muuttanut nimeään QtMessageHandleriksi ja kyseisen funktin prototyyppi on muuttunut. Prototyypissä käytetään char * -tyypin sijasta QStringiä. Lisäksi prototyypissä on QMessageLogContext-luokka lisätietojen antamiseksi. [1, Qt Core.] QGeoPositionInfoSource-luokassa on uusi puhdas virtuaalimetodi error, joka täytyy perivässä luokassa määrittää [1, Qt Positioning].

2.7 Paketointi yaml-tiedostoa käyttäen

Ohjelmasta on tehtävä paketti, jotta se on helposti käyttäjän asennettavissa laitteeseen. Sailfish käyttää RPM-pakettimuotoa. Laitteen pakettienhallinta hoitaa laitteessa paketin asennuksen, päivittämisen ja poiston. Tehdyn paketin voi myös julkaista sovelluskaupassa tai OpenRepos.net-palvelussa.

Sailfishille pakettia tehdessä käytetään yaml-tiedostoa. Tiedostosta luodaan Mer-projektin specify-työkalua käyttäen spec-tiedosto, jossa on tarkemmin määritetty ohjelman kääntäminen ja paketin asentaminen. [39.]

```
1 Name: harbour-monav
2 Summary: MoNav offers very fast and exact routing with
     OpenStreetMap data for mobile devices as well as desktop
     computers.
3 Version: 0.1
4 Release: 1
5 Group: Qt/Qt
6 URL: https://code.google.com/p/monav/
7 License: GPL
```

```
 8 Description: |
 9   MoNav offers very fast and exact routing with
       OpenStreetMap data for mobile devices as well as
       desktop computers.
10 PkgConfigBR:
11 - sailfishapp >= 0.0.10
12 - Qt5Core
13 - Qt5Qml
14 - Qt5Quick
15 Requires:
16 - sailfishsilica-qt5 >= 0.10.0
17 Files:
18 - '%{_datadir}/icons/hicolor/86x86/apps/%{name}.png'
19 - '%{_datadir}/applications/%{name}.desktop'
20 - '%{_datadir}/%{name}/qml'
21 - '%{_bindir}'
```

Listaus 28: MoNav-ohjelman yaml-tiedosto

Yaml-tiedostossa useimmat tietokentät ovat helposti ymmärrettävissä. Listauksessa 28 on esimerkki yaml-tiedostosta. Name määrittää paketin nimen, jolla paketti tunnistetaan pakettienhallinnassa. Summary on yhden rivin pituinen kuvaus ohjelmasta. Descriptionissa kerrotaan tarkemmin ohjelmasta. [40.]

Version kertoo paketissa olevan ohjelman version. Se kuuluu olla mahdollisimman samankaltainen alkuperäisen ohjelman version kanssa. Release on paketin versio. Se kertoo lukuna kuinka mones paketointi tämä on kyseistä ohjelman versiota kohden. Group-merkinnällä voi lajitella ohjelman ryhmään, joka kertoo sen ominaisuuksista. Tieto ryhmästä annetaan polkumuotoisena. [40.]

PkgConfigBR-merkinnässä kerrotaan listamuotoisena, mitä muita paketteja tarvitaan ohjelman kääntämiseen. Requires taas kertoo tarvittavat paketit ohjelman suorittamiseen. Tarvittavan paketin nimen perään voi määritellä vaadittava versio paketeista käyttämällä merkintöjä <, >, <=, >=, = ja kirjoittamalla vaadittava versionumero perään. [40.]

Files-merkintään annetaan kaikki pakettiin sisällytettävät tiedostot. Tiedostojen polkuna käytetään sitä, mihin ne ovat asennusvaiheessa määrietty asentumaan. Tämä tarkoittaa esimerkiksi polkua, mihin tiedosto asentuu, kun antaa käskyn “make install” pelkästään

perinteistä Makefileä käytettäessä. [41.] Annettaessa poluksi hakemisto, kaikki sen sisältämät tiedostot lisätään [42].

Poluissa voi käyttää myös makroja. Käytettävät makrot voi tulostaa komennolla “rpm –showrc”. Esimerkiksi %{_datadir} osoittaa järjestelmän ohjelmistojen käyttämään tietohakemistoon. [43.]

3 Käyttöliittymän toteuttaminen

3.1 Alkuperäisen ohjelman esittely

MoNav on navigaatio-ohjelma, joka toimii yhteydettömässä tilassa. Se on avoimen lähdekoodin ohjelma, joka on alun perin julkaistu Google Code -palvelussa. Reititys ohjelmassa perustuu Contraction Hierarchies -algoritmiin. Reitti on tällä mahdollista laskea todella pienessä ajassa. MoNavin on alun perin kirjoittanut Christian Vetter. Siihen on osallistunut myös useat muut kehittäjät. Kehitys on kuitenkin ollut monta vuotta pysähtyneenä. [44.]

MoNav on luotu Qt:llä ja sen käyttöliittymä käyttää Qt Widgetsejä. Tarkoitus on sovittaa ohjelma käyttämään QML:ää, Qt Quickia ja Sailfish Silicaa.

Ohjelman eri ominaisuudet muodostuvat laajennuksista, kuten karttarenderöinti, reitinetsintä ja osoitteenhaku. Näitä laajennuksia käytetään pääohjelmassa, jossa on karttanäkymä kartan ja reitin näyttämiseen sekä käyttöliittymä reitin valintaan ja muille asetuksille.

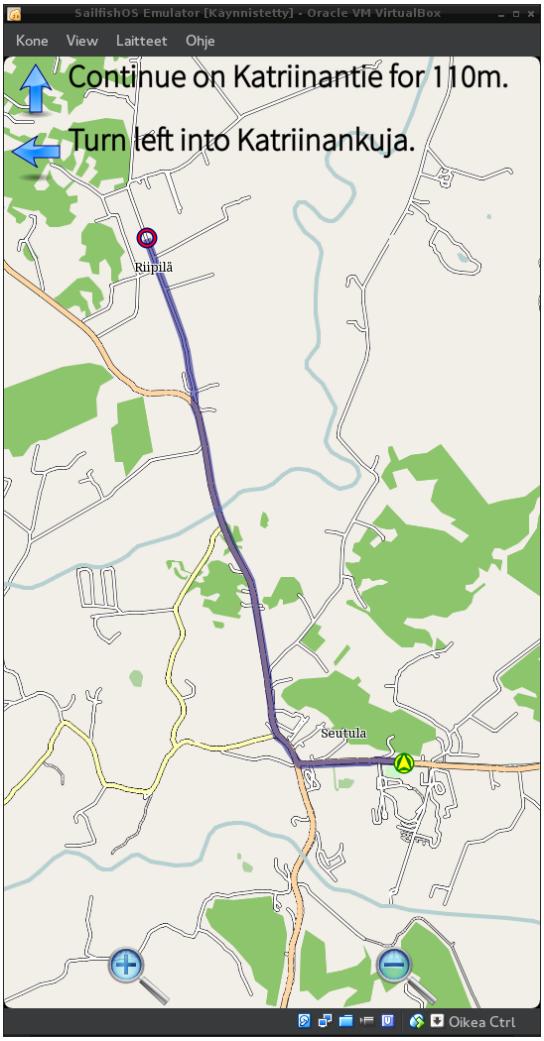
3.2 Käyttöliittymän vaatimukset

MoNavin uuden käyttöliittymän on tarkoitus olla helposti kosketusnäytöllä ohjattavissa. Käyttöliittymän yleisimmät toiminnot ovat saatavilla mahdollisimman vähillä valinnoilla.

Usein käytetyt valinnat vain yhden painalluksen päässä. Hakutoiminto on yksinkertainen, eikä tarvitse käyttäjältä tavallisesti useaa eri syötettä. Käyttöliittymä mukautuu yleisesti Sailfishin käyttöliittymän tyyliin.

3.3 Käyttöliittymän rakenne

MoNavin uudessa käyttöliittymässä on karttasivu 14, jossa näytetään kartta, reittiohjeet ja painikkeet zoomaukselle. Karttasivulla voi koskettamalla jotain kartan kohtaa avata kosketusvalikon. Karttasivu voi olla myös navigointitilassa, jossa karttaa ei voi koskettaa, kartta kääntyy kulkusuunnan mukaan ja näyttö pysyy koko ajan päällä. Navigointitilasta voi poistua sormella avaamalla PullDownMenun ja valitsemalla navigoinnin lopettavan valinnan.



Kuva 14: Karttasivu MoNav-ohjelmassa

Kosketusvalikon on tarkoitus toimia ohjelman päävalikkona. Siinä voi käyttää osoitehakua tai asettaa navigoinnin määränpään tai aloituskohdan kartalla kosketettuun kohtaan. Kosketusvalikosta löytyy myös kirjanmerkit ja asetukset karttapakettien ja karttamoduulien hallintaan. Kirjanmerkkejä voi lisätä ja poistaa valikosta. Kirjanmerkeillä voidaan asettaa jokin talletettu sijainti nykyiseksi navigoinnin kohteeksi.

Karttapakettien hallinnassa näytetään kartalla nykyinen valittu karttapaketti. Karttapaketin vaihtaessa valikossa karttamoduulivalikot päivittyvät näyttämään paketista löytyviä moduuleja. Karttapakettisivun sulkiessa valinnat tulevat heti käyttöön, ja uusi karttapaketti latautuu. Karttapakettien hallinta tulee näkyviin myös heti ohjelman käynnistyessä, kun karttapakettia ei ole vielä valittu tai aikaisemmin valitun karttapaketin lataamisessa on ongelmia.

Ohjelman ollessa pienennettynä, siinä näytetään kantta. Kannessa on tämän hetken reittiohjeet, jos navigointi on käynnissä.

3.4 Alkuperäisen karttanäytön sovittaminen QML-komponentiksi

3.4.1 QPainter-luokan käyttäminen

MoNavin kartan piirtämisestä huolehtii PaintWidget-luokka. PaintWidget luokka periytyy QWidget-luokasta. QWidget-luokassa kutsutaan paintEvent-metodia, kun piirtämistä tarvitaan. paintEvent-metodissa luodaan QPainter-luokka, jonka vanhemmaksi annetaan PaintWidget. Tämän jälkeen QPainter-luokka hoitaa kartan piirtämisen.

Qt Quickissa on QQuickPaintedItem-luokka, jolla pystyy luomaan itse piirretyn komponentin QML:ään. Sovitetun MoNavin PaintWidget-luokassa on perittävä QWidget-luokan sijasta QQuickPaintedItem-luokka. QWidgetsistä on helppo siirtyä QQuickPaintedItemin käyttöön, koska piirtämisessä käytetään myös QPainter-luokkaa. QQuickPaintedItemis-

sä on paint-metodi, jota kutsutaan, kun piirtämistä tarvitaan. paint-metodin argumenttinä on tarvittavan QPainter-luokan osoitin. [1, Qt Quick.]

3.4.2 Monikosketustuki karttanäytölle

MoNavissa ei alunperin ole mitään tukea monikosketukselle. Nykyään suurimmassa osassa matkapuhelimien karttasovelluksista käytetään monikosketusta. Koska käyttäjät ovat tottuneet käyttämään monikosketusta kartan zoomaukseen, se on tarvittava ominaisuus myös MoNavissa.

Koska alkuperäisessä MoNavissa ei ole portaatonta zoomausta, sellainen oli luotava. Alkuperäisessä MoNavissa on kuitenkin virtual zoom, jolla pystyy portaallisesti zoomaamaan karttaa suurempitarkkuuksisille näytöille. Muutin virtual zoomin reaaliluvuksi, jota käytetään oikeiden zoomausportaiden välillä. Virtual zoom siis vain skaalaa renderöityä karttaa jokaisen zoomausportaan välillä. Tämän vuoksi jouduin luomaan välimuistin skaalatuille karttatiileille, jotta skaalausta ei tarvitse tehdä uudelleen jokaisella ruudunpäivityksellä.

Karttaa zoomataan kahden näytön kosketuskohdan avulla. Kosketuskohtien välisen matkan muutoksen avulla lasketaan muutos zoomaukselle kartalla. Jokaisella kartan zoomausportaalla matka kartalla kaksinkertaistuu. Kosketuskohtien välisen matkan muutoksesta saa siis zoomasteikon muutoksen logaritmillä, jonka kantana on kaksi.

3.5 Osoitehaun toteuttaminen ja yksinkertaistaminen

Alkuperäisen MoNavin osoitehaku toimii niin, että ensin syötetään kaupunki ja sitten kadun nimi. Tämä tapa on kuitenkin sekava, koska käyttäjällä on yleensä osoitteesta tien nimi ensimmäisenä mielessä. Yksinkertaistin hakua niin, että hakusanaksi kirjoitetaan vain kadun nimi ja hakutulokseksi annetaan hakua vastaavat kadun nimet ja niiden kaupun-

git. Tämä yksinkertaistaa käyttäjän hakuvaiheita. Käyttäjän tarvitsee käyttää hakukenttää vain kerran.

Lisätarve osoitehaun muuttamiselle tulee, koska MoNav ei tällä hetkellä ota huomioon kaupunkien rajoja, vaan tie lisätään lähimpään kaupunkiin. Näin tie saattaa joutua naapurikaupungille. Jos käyttäjä joutuu ensin valitsemaan kaupungin ja sitten etsimään tien nimeä, niin väärään kaupunkiin kuuluvaa tietä ei varmasti löydy.

Huono puoli uudessa hakutavassa on yleisten tien nimien, kuten Kirkkotie, hakemisen vaikeutuminen. Oikean kaupungin joutuu selaamaan pitkästä listasta kaupunkeja. Myös hakudatan formaattia joutuu muuttamaan, joten vanhemmat hakudatat eivät toimi nyt muutetussa versiossa. Tästä johtuen ohjelman julkaisussa OpenRepos.net-palvelussa käyttäjät käyttivät alkuperäisen ohjelman dataa ja ohjelma kaatui käyttäjän hakiessa tien nimeä. Formaatin yhteensopivuuden selvittäminen on siis tärkeätä ja alkuperäisen formaatin tukemista kannattaa miettiä. Ohjelman seuraavassa versiossa yhteensopivuus tarkastettiin paremmin.

MoNavissa käytetään UnicodeTournamentTrie-moduulia osoitehakuun. Se käyttää puumallista rakennetta tiedonhakuun, jossa jokaisella solmulla on seuraavat hakusanassa käytettävät kirjaimet ja nykyisen hakusanan antamien hakutulosten osoittimet tietoihin. Moduulissa on kolme datatiedostoa: main, sub ja ways. Main-tiedostossa on hakupuu kaupunkien nimille, sub-tiedostossa teiden nimille ja ways-tiedostossa on lopullinen data hakutuloksen tiestä. Kaupungeille ja teille annetaan myös suuruusjärjestys, jonka mukaan ne näytetään hakutuloksessa.

Muutetussa versiossa main-tiedostoa ei enää tarvita, koska hakua kaupungin nimellä ei enää tehdä. Sen sijaan sub-tiedostoon lisätään jokaiselle solmulle sen teiden paikkakuntien nimet. Näin hakutulokseen voidaan liittää tien nimeä hakiessa myös kaupungin nimi.

3.6 QML Timerin käyttö ongelmien kiertämiseksi

Joissain QML:n komponenteissa tuntuu tarvitsevan odottaa hetken toiminnon jälkeen, ennen kuin uutta toimintoa voi tehdä. Tämän kiertämiseksi käytin QML Timeriä odottamaan hetken ennen uutta toimintoa.

QML Timer on ajastin. Sillä on aika, jonka se odottaa ja lähettää sitten signaalin triggered. Aika annetaan interval-propertyllä. Timer voi lähettää signaalin kerran tai toistuvasti, jonka voi määrittää repeat-propertyllä. Metodit start, stop ja restart ohjaavat Timerin käynnistystä ja pysäyttämistä. [1, Qt QML.]

PageStack ei suostu tekemään uutta toimintoa, jos sivun siirtymäanimaatio on käynnissä, vaan antaa esimerkiksi virheen “Cannot pop while operation is in progress”. PageStackilla on metodi completeAnimation, jonka pitäisi lopettaa animaatio heti, mutta siltikään sen kutsumisen jälkeen uutta toimintoa ei voi heti suorittaa. Ongelman voi kiertää käyttämällä Timeriä katsomaan tietyn ajan välein, onko PageStackin busy-property vielä tosi. [31.]

ComboBoxilla on currentIndex-property, jota muuttamalla voi vaihtaa valitun vaihtoehdon [31]. Kuitenkin, jos ComboBoxin ContextMenun sisältöä muuttaa, niin ComboBoxin päivittämiseen kuuluu hetki aikaa. Tästä syystä currentIndex-propertyä ei voi muuttaa heti ContextMenun sisällön muuttamisen jälkeen. Tämänkin ongelman voi kiertää käyttämällä Timeriä ja odottamalla hetki ennen currentIndex-propertyn päivittämistä.

3.7 Muutokset alkuperäiseen ohjelmaan toiminnallisuuteen

Koska useat alkuperäisen ohjelman asetusikkunoista käyttivät QWidgetsiä, niitä ei voi sovitetussa versiossa käyttää. Alkuperäinen ohjelma luki joitakin asetuksiaan suoraan näissä ikkunoissa olevista kentistä. Sovitetussa versiossa ei vielä ole asetusvalikkoa jokaiselle moduulille, joten jotkin asetukset on korvattu vakioarvoilla.

Alkuperäinen ohjelma käyttää Qt Positioning -moduulia sijainnin hakemiseen, mutta se ei ole Jolla Harbourissa hyväksytty moduuli. Sijainnin hakeminen onnistuu myös D-Busin avulla käyttäen Geocluea ja se onkin jo toteutettu osittain ohjelman sovitettuun versioon. [36.]

Navigoidessa alkuperäinen ohjelma ei pitänyt näyttöä päällä. Sovitetussa versiossa näyttöä pidetään päällä käyttäen D-Busin avulla Sailfishin mce-palvelua. Ohjelmaa pienennettäessä näytön pitäminen päällä loppuu ja suurrennettaessa ohjelma kuuntelee suurrennuseventin saapumista ja aktivoi jälleen näytön päällä pitämisen.

Alkuperäisen ohjelman vektorirenderöinnissä oli bugi, jossa ohjelma pystyi lataamaan vain tietyn verran tien pisteitä. Tästä johtuen seuraavan tien lukeminen alkoi edellisen tien pisteistä ja karttarenderöinti hajosi. Asia korjaantui yksinkertaisesti asettamalla ohjelma lukemaan niin monta tiepistettä kuin niitä on mahdollista olla.

4 Yhteenveto

Tässä projektissa on esitelty Qt:n ominaisuuksia ja sitä, kuinka ne soveltuvat käyttöliittymän luomiseen puhelimelle. Työn käsittelemä ohjelma on sovitettu yleiskäytön osalta uudelle alustalle. Käyttöliittymä on nopea ja helppo käyttää.

Ohjelma vastaa Sailfish Harbourin vaatimuksia suurelta osin, mutta käyttää vielä Qt Positioning -moduulia paikannukseen. Vastaava ominaisuus on jo osittain kirjoitettu suositellulla tavalla, mutta ei ole vielä käytössä.

Ohjelma julkaistiin OpenRepos.net-palvelussa. Julkaisun yhteydessä tullut ongelma liittyi uuden osoitehaun formaattiin. Osa käyttäjistä käyttivät alkuperäistä formaattia, joka kaatoi ohjelman. Ohjelman julkaisu sujui muuten hyvin.

Lähteet

1 Qt 5.2.1 Reference Documentation; 2013.

2 About Trolltech;. Saatavilla: https://web.archive.org/web/20030602181402/http://www.trolltech.com/company/index.html [viitattu 25 maaliskuuta, 2015].

3 Company Information;. Saatavilla: https://web.archive.org/web/20040117202409/http://www.troll.no/qt/troll.html [viitattu 25 maaliskuuta, 2015].

4 Qt Professional Development License;. Saatavilla: https://web.archive.org/web/19961218153523/http://www.troll.no/pricing.html [viitattu 25 maaliskuuta, 2015].

5 Files for Downloading;. Saatavilla: https://web.archive.org/web/19970627014503/http://www.troll.no/dl/ [viitattu 25 maaliskuuta, 2015].

6 Qt Product Information;. Saatavilla: https://web.archive.org/web/19961218153447/http://www.troll.no/qtinfo.html [viitattu 25 maaliskuuta, 2015].

7 KDE Free Qt Foundation;. Saatavilla: https://web.archive.org/web/20030814174429/http://www.trolltech.com/newsroom/announcements/00000004.html [viitattu 25 maaliskuuta, 2015].

8 Trolltech Makes the Next Generation of Qt Free Edition Open Source;. Saatavilla: https://web.archive.org/web/20030819010153/http://www.trolltech.com/newsroom/announcements/00000068.html [viitattu 25 maaliskuuta, 2015].

9 Qt Version 2.0 Released;. Saatavilla: https://web.archive.org/web/20030402001950/http://www.trolltech.com/newsroom/announcements/00000064.html [viitattu 25 maaliskuuta, 2015].

10 Trolltech Announces the Release of Qt 2.2 and Qt Designer;. Saatavilla: https://web.archive.org/web/20030401232358/http://www.trolltech.com/newsroom/announcements/00000041.html [viitattu 25 maaliskuuta, 2015].

11 Trolltech Makes Qt/Windows Available under New Non-Commercial License;. Saatavilla: https://web.archive.org/web/20030413194457/http://www.trolltech.com/newsroom/announcements/00000014.html [viitattu 25 maaliskuuta, 2015].

12 Trolltech Releases Qt 3.0;. Saatavilla: https://web.archive.org/web/20030402005932/http://www.trolltech.com/newsroom/announcements/00000075.html [viitattu 25 maaliskuuta, 2015].

13 Trolltech Launches Major New Version of Qt;. Saatavilla: https://web.archive.org/web/20060315162314/http://www.trolltech.com/newsroom/announcements/00000209.html [viitattu 25 maaliskuuta, 2015].

14 Nokia to acquire Trolltech to accelerate software strategy;. Saatavilla: https://web.archive.org/web/20080225064738/http://trolltech.com/company/newsroom/announcements/press.2008-01-28.4605718236 [viitattu 25 maaliskuuta, 2015].

15 The Nokia acquisition;. Saatavilla: https://web.archive.org/web/20081122080038/http://trolltech.com/about/the-nokia-acquisition [viitattu 25 maaliskuuta, 2015].

16 Qt 4.4 Framework Broadens Rich Application Development with Integration of Web Content and Portability to Mobile Devices;. Saatavilla: https://web.archive.org/web/20080511172056/http://trolltech.com/company/newsroom/announcements/press.2008-05-02.5256347247 [viitattu 25 maaliskuuta, 2015].

17 Qt 4.5, Qt Creator released;. Saatavilla: https://web.archive.org/web/20110812200005/http://qt.nokia.com/about/news/nokia-releases-new-qt-developer-offerings-to-increase-productivity-and-performance/ [viitattu 25 maaliskuuta, 2015].

18 Nokia Releases Qt 4.6;. Saatavilla: https://web.archive.org/web/20110812195826/http://qt.nokia.com/about/news/nokia-releases-qt-4.6 [viitattu 25 maaliskuuta, 2015].

19 Nokia announces official Qt port to Maemo 5;. Saatavilla: https://web.archive.org/web/20091013053821/http://qt.nokia.com/about/nokia-announces-official-qt-port-to-maemo-5 [viitattu 25 maaliskuuta, 2015].

20 Qt 4.7.0 now available;. Saatavilla: http://blog.qt.io/blog/2010/09/21/qt-4-7-0-now-available/ [viitattu 25 maaliskuuta, 2015].

21 Qt 4.8.0 Released;. Saatavilla: http://blog.qt.io/blog/2011/12/15/qt-4-8-0-released/ [viitattu 25 maaliskuuta, 2015].

22 Digia to Acquire Qt from Nokia;. Saatavilla: https://web.archive.org/web/20120920003428/http://qt.digia.com/About-us/News/Digia-to-Acquire-Qt-from-Nokia/ [viitattu 25 maaliskuuta, 2015].

23 Introducing Qt 5.0;. Saatavilla: http://blog.qt.io/blog/2012/12/19/qt-5-0/ [viitattu 25 maaliskuuta, 2015].

24 Download Qt;. Saatavilla: http://www.qt.io/download/ [viitattu 25 maaliskuuta, 2015].

25 Run-Time Dynamic Linking;. Saatavilla: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685090%28v=vs.85%29.aspx [viitattu 2 huhtikuuta, 2015].

26 D-Bus tutorial;. Saatavilla: http://dbus.freedesktop.org/doc/dbus-tutorial.html [viitattu 23 maaliskuuta, 2015].

27 User Experience;. Saatavilla: https://sailfishos.org/design/ [viitattu 30 maaliskuuta, 2015].

28 Lipstick;. Saatavilla: https://github.com/nemomobile/lipstick/blob/master/README [viitattu 5 toukokuuta, 2015].

29 All about us and the OS;. Saatavilla: https://sailfishos.org/about/ [viitattu 30 maaliskuuta, 2015].

30 Mer;. Saatavilla: http://www.merproject.org/ [viitattu 30 maaliskuuta, 2015].

31 Sailfish Silica Reference Documentation; 2012.

32 [SailfishDevel] pagestack signals for eg pageback;. Saatavilla: https://lists.sailfishos.org/pipermail/devel/2013-August/000635.html [viitattu 23 maaliskuuta, 2015].

33 About OpenRepos;. Saatavilla: https://openrepos.net/content/basil/about-openrepos [viitattu 23 maaliskuuta, 2015].

34 Warehouse for SailfishOS;. Saatavilla: https://openrepos.net/content/basil/warehouse-sailfishos [viitattu 23 maaliskuuta, 2015].

35 Welcome to Jolla Harbour;. Saatavilla: https://harbour.jolla.com/ [viitattu 30 maaliskuuta, 2015].

36 Sailfish FAQ;. Saatavilla: https://harbour.jolla.com/faq [viitattu 30 maaliskuuta, 2015].

37 Software Development Kit;. Saatavilla: https://sailfishos.org/develop/sdk-overview/ [viitattu 23 maaliskuuta, 2015].

38 The Transition from Qt 4.x to Qt 5;. Saatavilla: http://qt-project.org/wiki/Transition_from_Qt_4.x_to_Qt5 [viitattu 23 maaliskuuta, 2015].

39 Spectacle Tutorial;. Saatavilla: https://github.com/mer-tools/spectacle/blob/master/README.md [viitattu 23 maaliskuuta, 2015].

40 Tags: Data Definitions;. Saatavilla: http://www.rpm.org/max-rpm/s1-rpm-inside-tags.html [viitattu 23 maaliskuuta, 2015].

41 Creating the Spec File;. Saatavilla: http://www.rpm.org/max-rpm/s1-rpm-build-creating-spec-file.html [viitattu 23 maaliskuuta, 2015].

42 The %files List;. Saatavilla: http://www.rpm.org/max-rpm/s1-rpm-inside-files-list.html [viitattu 23 maaliskuuta, 2015].

43 Packaging:RPMMacros;. Saatavilla: http://fedoraproject.org/wiki/Packaging:RPMMacros [viitattu 23 maaliskuuta, 2015].

44 monav;. Saatavilla: https://code.google.com/p/monav/ [viitattu 23 maaliskuuta, 2015].