MeeGo Application and Platform Development

Exercise Book

digia

Trademarks and Acknowledgements

The exercises for this course have been developed by Digia Plc Training team.

Exercise 1: Quality Assurance

Objectives

- Get familiar with MeeGo quality assurance

Overview

MeeGo quality assurance covers a set of tools to run tests on a target device. Instead of the target device, it is possible to run the tests in QEMU as well.

Practical Outline

1. First, you need to install the basic test tool set. Look at http://wiki.meego.com/Quality/QA-tools/How_to_set_up_repositories how to install the repositories. The installation instructions are given in http://wiki.meego.com/Quality/QA-tools/Testplanner
2. Now you may start the testrunner (`testrunner-ui`) and open for example tests from the MCTS (provided in the memory stick).
3. To run the tests, you need to install tests to the target + additionally install EAT (Enable Automated Test) to your target. The installation and test runs are left as an additional exercise for the participants.

Exercise 2: MeeGo Touch-Based Application

Objectives

- Familiarize yourself with MeeGo Touch classes

Overview

Your task is to design and implement a simple MeeGo Touch application. The minimum requirements are:

- The application must have at least two views
- You may decide yourself how to navigate between the views.
- Each view must have more than one widget in a central widget. The widgets must be organized using a layout and the layout should look nice in both portrait and landscape orientations.

Note! There are several examples to help you in meegotouch library files.

Exercise 3: Adaptive Flashlight

Objectives

Play around with the Sensors API

References

Exercise starting point:

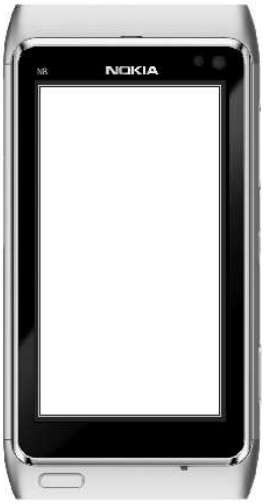
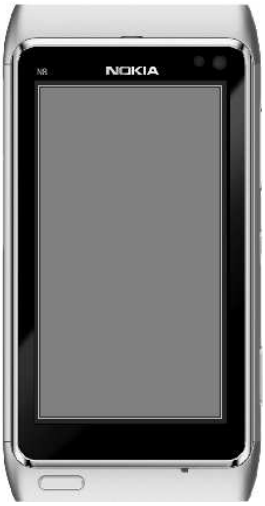
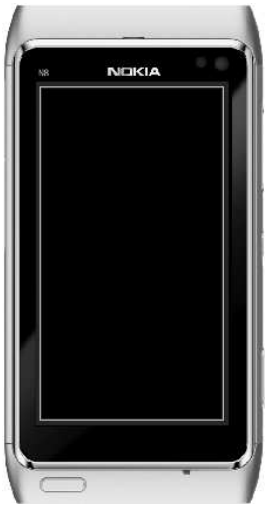
\templates\Flashlight

Exercise example solution:

\solutions\Flashlight

Overview

In this exercise we will create a very simplistic, adaptive "flashlight" application: When it's dark, the screen will be white and when it's less dark it will be gray and in full sunlight it will be black:

		
Dark	Bright	Sunny

Practical Outline

1. Open the **project template** in QtCreator from the set of codes distributed. The template creates a white screen.
2. Try in QtSimulator to see it works.
3. Add a QAmbientLightSensor for your mainwindow
    a. Remember to modify the .pro file and to use the mobility name space
4. Create your own slot function where you will change the color of the screen (graphicsview's background brush) depending on the reading of the sensor
5. Start listening to changes in the reading
6. Voilá! Try in QtSimulator and in device!
7. Optional: Add eye-Candy, publish in Ovi store, make a million dollars

Exercise 4: Service Development

Objectives

- Learn how to write servers providing D-Bus interface with Qt mappings in MeeGo

Overview

MeeGo applications are written using Qt so why not to write the middleware libraries using Qt as well. Remember that D-Bus itself is programming language agnostic, the service can be implemented with any language supporting D-Bus and supported by Qt.

Implement a simple server, which replies to client requests using D-Bus messages. Use Qt to write the server and the D-Bus bindings. Your server may simply return some data to the clients so that you can easily check the communication works.

Practical Outline

1. Study what kind of services there exist in MeeGo.
    a. Launch QEMU with MeeGi image (if you have problems with QEMU, you may do this in your Linux virtual box).
    b. Create a terminal session to QEMU
        i. ssh meego@127.0.0.1 -p 6666
        ii. The password is meego
    c. Study services and interfaces in /usr/share/dbus-1
        i. Which process offers the ring connection manager service we discussed in the lectures?
        ii. How do you get the API of this service?
2. Your actual task in this exercise is to implement the D-Bus interface for a remote controlled car. The solution is provided in `meego_trn\solutions\remotecontrolledcar` and the exercise source code template in `meego_trn\templates\remotecontrolledcar`. The exercise has two processes: The car, which is the server and the controller used to control the car.
3. Study first car.xml file found both in car and controller projects. If you create a D-Bus service of your own, you are expected to create a similar service description file. The library `libqt4-dev` provides you tools to generate the service stub files for the server and the client. You do not need to generate them in this exercise, because they have been created for you:
    a. Server side (adaptor): `qdbusxml2cpp –c car_adaptor –a car_adaptor.h:car_adaptor.cpp car.xml`
    b. Client side: `qdbusxml2cpp –v –c carInterface –p car_interface.h:car_interface.cpp car.xml`
    c. Study the generated files to understand the server and client side APIs.
4. Car
    a. In main.cpp /* TODO */,
    b. Instantiate CarInterfaceAdaptor. Who own this object? Do you need to delete it yourself?
    c. Use the static function in QDBusConnection to get a session sub.
    d. Register your service object as /Car
    e. Register your service.
5. Controller

a. In controller.h /* TODO */ declare four private slots related to UI QPushButtons. Take care of connections as well. Implement the slot functions into the controller.cpp

b. In the Controller constructor in controller.cpp, instantiate CarInterface(use the meber pointer car, which is already declared for you). Use the same service object name you had in the server side.

6. Now both the server and the client are ready. Launch both processes and enjoy driving!