Creating a basic IVI Compositor with QML

Submitted by alexander.wingard on Fri, 2019-12-13 14:02

This example shows how to create and use a basic custom In-Vehicle Infotainment (IVI) compositor using the WaylandCompositor in QML. We will create an IVI compositor that splits the display into two IVI-surfaces. The orientation and position of the split is defined by command line arguments.

To begin, we create a new project using the CrossControl QtQuick2 Application template, selecting resolution, orientation and kit for the device we're working with. In this example, we'll be targeting a CCpilot VS.

We want to be able to control the orientation and position of the split with command line parameters, so we modify the main(...) function in main.cpp to look for them.
First we set default values in case the parameters are not provided.

```
// Init default values for arguments
    int orientation        = 90;
// Portrait mode, USB connectors downwards
    int bottomScreenHeight = 500;
```

Then we add code that scans the argument vector for the ones we're interested in and replaces our default values if found.

```
// Read arguments
    for (int i = 0; i < (argc - 1); i += 2) {
        QString [1] argument(argv[i + 1]);
        if (argument == QStringLiteral("-bottomHeight")) {
            bottomScreenHeight = atoi(argv[i + 2]);
        }
        else if (argument == QStringLiteral("-orientation")) {
            // Normalize orientation to [0-360] degrees

/** NOTE: In a real case we likely want to limit values
             * to 90*n steps here.
             * Non-orthogonal rotations are supported for
             * demonstration purposes                  */
            orientation = (atoi(argv[i + 2])) % 360;
            while (orientation < 0)
                orientation += 360;
        }
   }
```

We also need to switch from the default QQuickView to the more generic QQMLApplicationEngine to support the new WaylandCompositor node we'll be using as root node in our QML. We do this by replacing

```
QQuickView *view = new QQuickView;
    QObject [2]::connect(view->engine(), &QQmlEngine::quit, &app, &
QGuiApplication::quit);


    ...
```

```
     view->rootContext()->setContextProperty("targetARM", QVariant [3](
targetARM));
    view->rootContext()->setContextProperty("displayWidth", QVariant [3](
displayWidth));
    view->rootContext()->setContextProperty("displayHeight", QVariant [3]
(displayHeight));
     view->setSource(QStringLiteral("qrc:/main.qml"));
    view->showFullScreen();
```

with

```
QQmlApplicationEngine engine;
    QObject [2]::connect(&engine, &QQmlApplicationEngine::quit, &app, &
QGuiApplication::quit);

    ...
     engine.rootContext()->setContextProperty("targetARM", QVariant [3](
targetARM));
    engine.rootContext()->setContextProperty("displayWidth", QVariant [3]
(displayWidth));
    engine.rootContext()->setContextProperty("displayHeight", QVariant
[3](displayHeight));
     engine.load(QUrl [4](QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;
```

Finally, we expose our new arguments to the QML engine by adding the following next to the existing setContextProperty lines.

```
// Expose arguments to QML layer
    engine.rootContext()->setContextProperty("orientation", QVariant [3](
orientation));
    engine.rootContext()->setContextProperty("bottomScreenHeight",
QVariant [3](bottomScreenHeight));
```

Now we have what we need to build our compositor in QML. The template's default main.qml will not be needed so we will completely replace its content with our own compositor code. First, we need to import the QML modules we need.

```
import QtQuick 2.12
import QtWayland.Compositor 1.0
import QtQuick.Window [5] 2.12
```

As root object we use a WaylandCompositor

```
import QtQuick 2.12
import QtWayland.Compositor 1.0
import QtQuick.Window [5] 2.12
 WaylandCompositor {
}
```

Inside it, we define the compositor's "window" structure by adding a WaylandOutput element and building a normal QML structure underneath using Rectangles to represent IVI surface areas. We start with a Window element as the structure's root, and create an Item element of same size in its center to handle screen rotation. We also determine if we're in a portrait or landscape mode to know if we need to invert width and height. Since this example allows for non-orthogonal rotation we pick the mode that is closest.

```
import QtQuick 2.12
import QtWayland.Compositor 1.0
```

```
import QtQuick.Window [5] 2.12
 WaylandCompositor {
    WaylandOutput {
        sizeFollowsWindow:
 true        window: Window [5] {
            id:
 base            width:
 displayWidth            height:
 displayHeight            visible:
 true
            Item [6] {
                id:
 screenRoot                anchors.centerIn:
 parent                height: portrait ? parent.width : parent.height
                width: portrait ? parent.height : parent.width
                rotation:
 orientation
                // Since we support non-orthogonal orientations,
                // we make a best fit decision on portrait mode here
                property bool portrait: (orientation%360 > 45 &&
 orientation%360 < 135) ||
                                        (orientation%360 > 225 &&
 orientation%360 < 315)
            }
        }
    }
}
```

Under the screenRoot Item, we add our Rectangles and their positioning logic. We also add some hints about which Rectangle is used for which IVI surface ID.

```
import QtQuick 2.12
import QtWayland.Compositor 1.0
import QtQuick.Window [5] 2.12
 WaylandCompositor {
    WaylandOutput {
        sizeFollowsWindow:
 true        window: Window [5] {
            id:
 base            width:
 displayWidth            height:
 displayHeight            visible:
 true
            Item [6] {
                id:
 screenRoot                anchors.centerIn:
 parent                height: portrait ? parent.width : parent.height
                width: portrait ? parent.height : parent.width
                rotation:
 orientation
                // Since we support non-orthogonal orientations,
                // we make a best fit decision on portrait mode here
                property bool portrait: (orientation%360 > 45 &&
 orientation%360 < 135) ||
                                        (orientation%360 > 225 &&
 orientation%360 < 315)
                Rectangle [7] {
                    id:
 topArea                    anchors {
                        left: parent.left
```

```
                        right: parent.right
                        top: parent.top
                        bottom: bottomArea.top
                    }
                    color: "cornflowerblue"
                    Text [8] {
                        anchors.centerIn:
 parent                         text: "Ivi surface with id 1"
                    }
                }
                Rectangle [7] {
                    id:
 bottomArea                     anchors {
                        left: parent.left
                        right: parent.right
                        bottom: parent.bottom
                    }
                    height:
 bottomScreenHeight                     color: "burlywood"
                    Text [8] {
                        anchors.centerIn:
 parent                         text: "Default ivi surface"
                    }
                }
            }
        }
    }
}
```

We need to define what types of wayland surfaces our compositor support. This is done by instantiating a node for each supported surface type directly under the Wayland Compositor element. In this example we only support IVI-surfaces. We also need to provide a component that should be instanced when a wayland application connects.

```
WaylandCompositor {
    WaylandOutput {
        sizeFollowsWindow:
 true        window: Window [5] {
            id:
 base            width:
 displayWidth            height:
 displayHeight            visible:
 true
            Item [6] {
                id:
 screenRoot                anchors.centerIn:
 parent                height: portrait ? parent.width : parent.height
                width: portrait ? parent.height : parent.width
                rotation:
 orientation
                 // Since we support non-orthogonal orientations,
                // we make a best fit decision on portrait mode here
                property bool portrait: (orientation%360 > 45 &&
 orientation%360 < 135) ||
                                        (orientation%360 > 225 &&
 orientation%360 < 315)
                Rectangle [7] {
                    id:
 topArea                    anchors {
```

```
                        left: parent.left
                        right: parent.right
                        top: parent.top
                        bottom: bottomArea.top
                    }
                    color: "cornflowerblue"
                    Text [8] {
                        anchors.centerIn:
 parent                         text: "Ivi surface with id 1"
                    }
                }
                Rectangle [7] {
                    id:
 bottomArea                     anchors {
                        left: parent.left
                        right: parent.right
                        bottom: parent.bottom
                    }
                    height:
 bottomScreenHeight                     color: "burlywood"
                    Text [8] {
                        anchors.centerIn:
 parent                         text: "Default ivi surface"
                    }
                }
            }
        }
    }
     Component [9] {
        id:
 chromeComponent        ShellSurfaceItem {
            anchors.fill:
 parent             onSurfaceDestroyed: destroy()
            onWidthChanged: handleResized()
            onHeightChanged: handleResized()
            function handleResized() {
                shellSurface.sendConfigure(Qt [10].size(width, height));
            }
        }
    }
     IviApplication {
        onIviSurfaceCreated: {
            var surfaceArea = iviSurface.iviId === 1 ? topArea :
 bottomArea;
            var item = chromeComponent.createObject(surfaceArea, {
"shellSurface": iviSurface } );
            item.handleResized();
        }
    }
}
```

Our compositor is now done. To test run our compositor directly from Qt Creator we need to define a few run settings first since we're not going to be targeting the default Weston compositor. First we tell the program to run directly on EGLFS by adding the command line argument **-platform eglfs** in the *Command line arguments:* field in Project -> -> Run settings
We can also take this opportunity to test our own custom arguments so we add those too. `-platform eglfs -orientation 270 -bottomHeight 300`

We also need to set some environment variables. We do this in Run Environment under Run settings for the VS kit. [Projects -> -> Run -> Run Environment -> expand Details -> Add]
Some of these are required, some are optional

- XDG_RUNTIME_DIR=/run/user/root # (required) Sets the compositor to run as root
- QT_QPA_EVDEV_TOUCHSCREEN_PARAMETERS=/dev/input/touchscreen0 # (required for touch) Tell eglfs to listen to touch events from the VS touchdisplay
- FB_MULTI_BUFFER=3 # (recommended) Use tripple buffering
- QT_QPA_EGLFS_HIDECURSOR=1 # (recommended) Hides the mouse cursor since we're working with a touch display

And finally, after ensuring that the Weston compositor is not running on our CCpilot device (if it is, stop it with **/etc/init.d/westion stop** from console), we can hit Run in Qt Creator and we should see our compositor running, ready to receive IVI-applications!

One final note - to really verify that it works we, of course, want to run some ivi-application. Any normal wayland application can be run as an ivi-application by defining the following environment variables for them and adding the command line argument **-platform wayland**.

- XDG_RUNTIME_DIR=/run/user/root
- QT_WAYLAND_SHELL_INTEGRATION=ivi-shell
- QT_IVI_SURFACE_ID=NNNN (where NNNN is the target surface id the compositor should place the application on)

In our compositor we defined the top area as ID 1 and the bottom as default meaning any application with surface ID other than 1, or none, will end up in the bottom area, and those with it set to 1 in the top.

Environment and Versions:
Qt-5.12.0 LinX Designer 4.0 iMX 6 platforms
Category:
QtQuick / QML-Programming [11]
Applies to version:
Qt 5.12

Links
[1] http://doc.trolltech.com/latest/qstring.html
[2] http://doc.trolltech.com/latest/qobject.html
[3] http://doc.trolltech.com/latest/qvariant.html
[4] http://doc.trolltech.com/latest/qurl.html
[5] http://qt-project.org/doc/qt-5/qml-qtquick-window-window.html
[6] http://qt-project.org/doc/qt-5/qml-qtquick-item.html
[7] http://qt-project.org/doc/qt-5/qml-qtquick-rectangle.html
[8] http://qt-project.org/doc/qt-5/qml-qtquick-text.html
[9] http://qt-project.org/doc/qt-5/qml-qtqml-component.html
[10] http://qt-project.org/doc/qt-5/qml-qtqml-qt.html
[11] https://support.crosscontrol.com/kb/qtquick-qml-programming