Python KDE Qt Framework

Blockchain & DAGs

Smart Contracts

Open Source

Building hardware with Arduino

Travelling

Home Automation

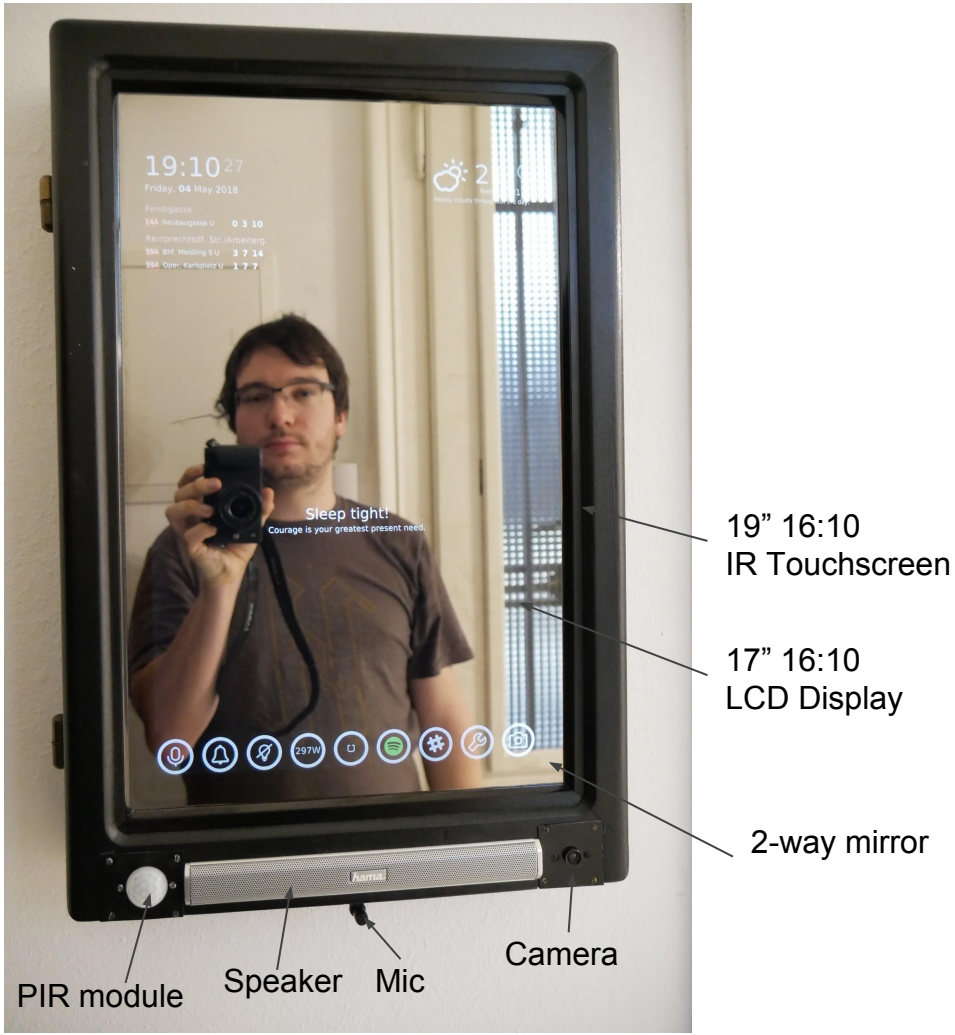
Internet Of Things

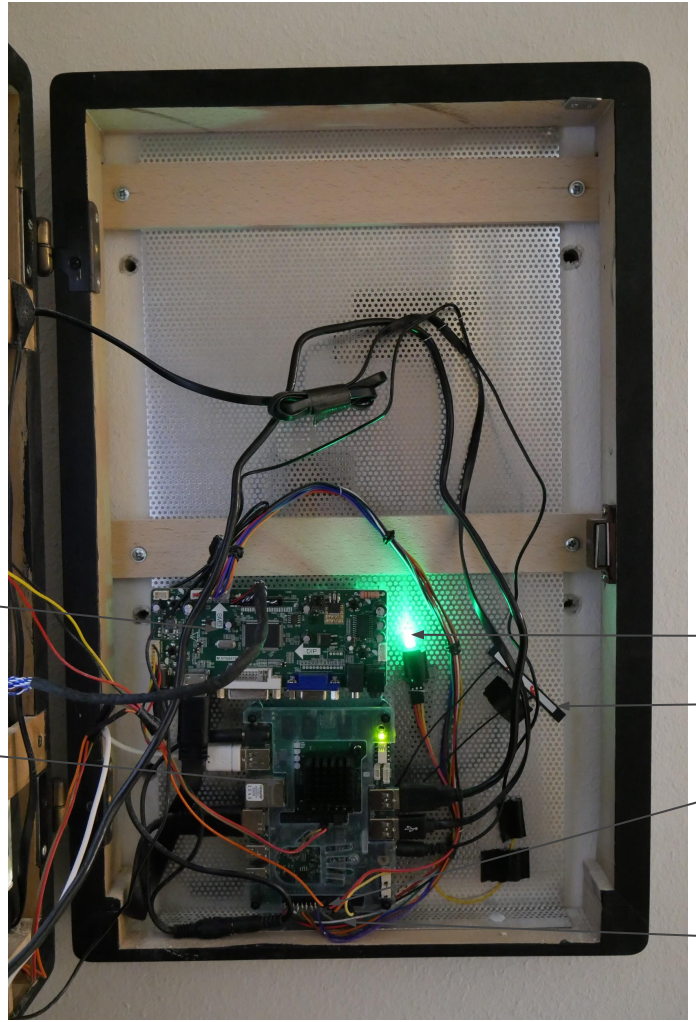
# Python & QML


## Lukas Hetzenecker

blog.hetzenecker.me • lukas@hetzencker.me

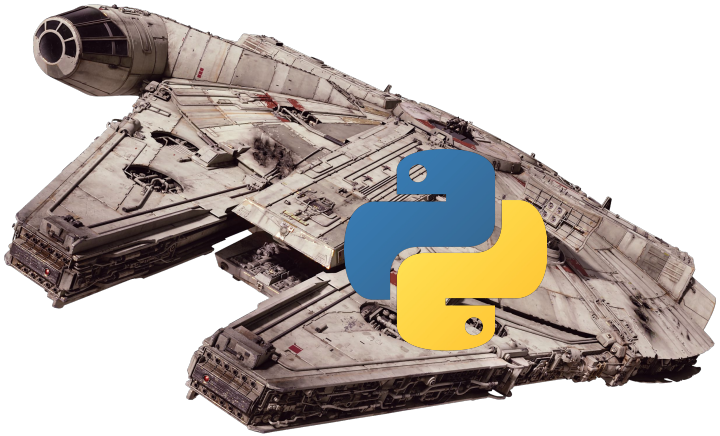
# Hardware


19:10 27
Friday, 04 May 2018
Sleep tight!
Courage is your greatest present need.
297W
19” 16:10
IR Touchscreen
17” 16:10
LCD Display
2-way mirror
Camera
Mic
Speaker
PIR module


Display controller
udoo x86
LED
Wifi antenna
Arduino-compatible GPIO
Monitor keyboard input

# Software

# Qt Framework

# Qt Integration

---

- ✓ QObject signals
- ✓ QObject slots
- ✓ QObject properties
- ✓ Q*Model

PyQt

PySide

# PyQt

---

sip binding generator

- Python bindings for Qt 5.x
- Windows, Linux, UNIX, Android, OS X and iOS
- Riverbank Computing
- **GPL v3**

# PySide

---

shiboken2 binding generator

- Python bindings for Qt 4.x
- Supports all platform from Qt
  - Windows, Linux, Android, macOS + iOS, Embedded Linux, QNX,...
- Officially from The Qt Company (initiated by Nokia)
- **LGPL v2.1**
  - free/open source + proprietary software development

# PySide: Roadmap

---

**PySide2**

renamed to **Python for Qt**

LGPL, GPL & commercial licenses

official technical support from Qt

full integration into QtCreator

# Python for Qt: Roadmap

June 13th, 2018

## Qt for Python 5.11 released

We are happy to announce the first official release of Qt for Python (Pyside2).

[...] still a **Technical Preview**

aim is to release Qt for Python 5.12 without the Tech Preview flag

Source: http://blog.qt.io/blog/2018/06/13/qt-python-5-11-released/

# PyQt5 + QML integration

- Python types sub-classed from *QObject* can be registered with **QML**.
- Instances of registered Python types can be created and made available to QML scripts.
- Python properties, signals and slots can be given revision numbers that only those implemented by a specific version are made available to QML.
- Registering Python types with QML is done in the same way is it is done with C++ classes, i.e. using the qmlRegisterType(), qmlRegisterSingletonType(), qmlRegisterUncreatableType() and qmlRegisterRevision() functions.

# Wiener Linien

## Plugin for QML

# C++ plugin for QML

```
// wienerlinienplugin.h
#include <QQmlExtensionPlugin>

class WienerLinienPlugin : public QQmlExtensionPlugin
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID QQmlExtensionInterface_iid)
public:
    void registerTypes(const char *uri);
};

// wienerlinienplugin.cpp
#include "wienerlinien.h"
#include "wienerlinien.cpp"
#include <qqml.h>

void WienerLinienPlugin::registerTypes(const char *uri)
{
    qmlRegisterType<WienerLinienService>(uri, 1, 0, "WienerLinienService");
}
```

# Python plugin for QML

```
from PyQt5.QtQml import qmlRegisterType, QQmlExtensionPlugin
from wienerlinienservice import WienerLinienService

class WienerLinienPlugin(QQmlExtensionPlugin):
    def registerTypes(self, uri):
        qmlRegisterType(WienerLinienService, "WienerLinien", 1, 0, "WienerLinienService")
```

# WienerLinien service

```
class WienerLinienService(QObject):
    rblsChanged = pyqtSignal(list)
    keyChanged = pyqtSignal(str)
    modelChanged = pyqtSignal(QAbstractListModel)
```

# WienerLinien service

```
class WienerLinienItem(ListModelItem):
    roles = {
        Qt.UserRole + 1: b'rbl',
        Qt.UserRole + 2: b'stop',
        Qt.UserRole + 3: b'line',
        Qt.UserRole + 4: b'direction',
        Qt.UserRole + 5: b'departures',
    }
```

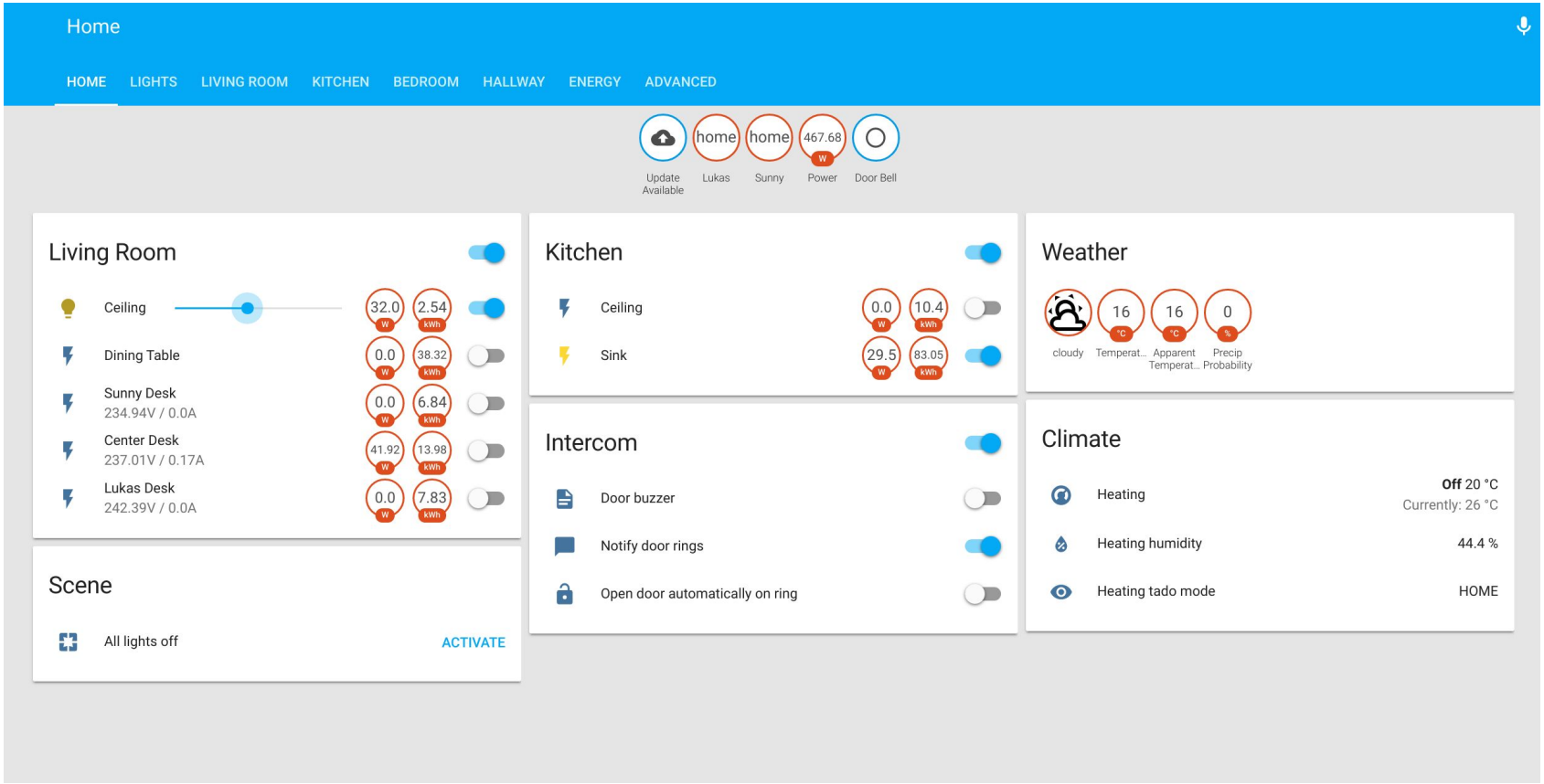
```
async def _update(self):
  self._model.beginResetModel()
  self._model.clear()

  async with aiohttp.ClientSession() as session:
    params = MultiDict([('sender', self._key)] + [('rbl', rbl) for rbl in self._rbls])

    async with session.get(base_url, headers=HEADERS, params=params) as resp:
      data = await resp.json()
      monitors = data['data']['monitors']
      for monitor in sorted(monitors, key=lambda m:
                            m['locationStop']['properties']['attributes']['rbl']):
        properties = monitor['locationStop']['properties']
        rbl = properties['attributes']['rbl']
        [...]
        self._model.addItem(WienerLinienItem(self._model,
                                             rbl=rbl,
                                             stop=stop,
                                             [...]))

        self._model.endResetModel()

  QTimer.singleShot(5000, lambda: asyncio.ensure_future(self._update()))
```

```
$ ls -l plugins/WienerLinien/
total 24
lrwxrwxrwx 1 lukas lukas   75 Dec 24 17:16 libpyqt5qmlplugin.so -> /home/lukas/Qt/gcc_64/plugins/PyQt5/libpyqt5qmlplugin.so
-rwxrwxr-x 1 lukas lukas 1036 May  5 07:31 plugins.qmltypes
-rwxrwxr-x 1 lukas lukas   43 Oct 22  2017 qmldir
-rwxrwxr-x 1 lukas lukas  460 Nov  1  2017 wienerlinienplugin.py
-rwxrwxr-x 1 lukas lukas 4192 Apr 15 01:57 wienerlinienservice.py

$ cat plugins/WienerLinien/qmldir
module WienerLinien
plugin pyqt5qmlplugin

$ qmlplugindump WienerLinien 1.0 > plugins/WienerLinien/plugins.qmltypes
$ head plugins/WienerLinien/plugins.qmltypes
import QtQuick.tooling 1.2

// This file describes the plugin-supplied types contained in the library.
// It is used for QML tooling purposes only.
//
// This file was auto-generated by:
// 'qmlplugindump WienerLinien 1.0'

Module {
   dependencies: ["QtQuick 2.8"]
   Component { name: "ListModel"; prototype: "QAbstractListModel" }
   Component {
       name: "WienerLinienService"
       prototype: "QObject"
       exports: ["WienerLinienService 1.0"]
       exportMetaObjectRevisions: [0]
       Property { name: "rbls"; type: "QVariantList" }
       Property { name: "key"; type: "string" }
       Property { name: "model"; type: "QObject"; isReadonly: true; isPointer: true }
       Signal {
           name: "rblsChanged"
           Parameter { type: "QVariantList" }
       }
```

```
import WienerLinien 1.0

Item {
    // get the RBLs from https://till.mabe.at/rbl/
    property var rbls;
    property string key;

    WienerLinienService {
        id: wienerLinienService
        rbls: wienerLinienPanel.rbls
        key: wienerLinienPanel.key
    }

    ListView {
        anchors.top: parent.top; anchors.bottom: parent.bottom; width: parent.width
        interactive: false
        model: wienerLinienService.model

        delegate: StopDelegate { }

        section.property: "stop"
        section.criteria: ViewSection.FullString
        section.delegate: sectionHeading
```

```
// The delegate for each section header
 Component {
     id: sectionHeading

     Label {
         width: 200
         height: 50
         text: section
         topPadding: 10
         font.pixelSize: 25
         color: Material.color(Material.Red, Material.Shade100)
     }
 }
```

# Home Assistant


Home
HOME
LIGHTS
LIVING ROOM
KITCHEN
BEDROOM
HALLWAY
ENERGY
ADVANCED
Update Available
home
Lukas
home
Sunny
467.68
W
Power
Door Bell
Living Room
Ceiling
32.0
W
2.54
kWh
Dining Table
0.0
W
38.32
kWh
Sunny Desk
234.94V / 0.0A
0.0
W
6.84
kWh
Center Desk
237.01V / 0.17A
41.92
W
13.98
kWh
Lukas Desk
242.39V / 0.0A
0.0
W
7.83
kWh
Scene
All lights off
ACTIVATE
Kitchen
Ceiling
0.0
W
10.4
kWh
Sink
29.5
W
83.05
kWh
Intercom
Door buzzer
Notify door rings
Open door automatically on ring
Weather
cloudy
16
°C
Temperat...
16
°C
Apparent Temperat...
0
%
Precip Probability
Climate
Heating
Off 20 °C
Currently: 26 °C
Heating humidity
44.4 %
Heating tado mode
HOME

homeassistantservice.py

```
class HomeAssistantService(QObject):
  @pyqtSlot()
  def start(self):
    asyncio.ensure_future(self._start())

  async def _start(self):
     try:
       url = '%s://%s:%s/api/websocket' % ('wss' if self._secure else 'ws', self._hostname, self._port)
       logger.debug('Connecting to home-assistant websocket %s...', url)
       self._websocket = await websockets.connect(url)
       auth_message = await self._websocket.recv()

       await self._websocket.send(
                    json.dumps({'id': SUBSCRIBE_EVENTS_ID, 'type': 'subscribe_events', 'event_type': 'state_changed'}))
       while True:
         data = await self._websocket.recv()
         message = json.loads(data)
         if message['type'] == 'event':
           event = message['event']
           if event['event_type'] == 'state_changed':
             entity_id = event['data']['entity_id']
             new_state = event['data']['new_state']
             self._update_state(entity_id, new_state)
     except Exception as e:
       logger.error(e)
       await asyncio.sleep(10.0)
       asyncio.ensure_future(self._start())
```

homeassistantservice.py

```
def add_state_listener(self, entity_id, listener):
    self._state_listeners[entity_id].append(listener)
    return self._states.get(entity_id, {})

def remove_state_listener(self, entity_id, listener):
    self._state_listeners[entity_id].remove(listener)

def _update_state(self, entity_id, state):
    self._states[entity_id] = state
    for listener in self._state_listeners[entity_id]:
        listener(state)
```

```python
class HomeAssistantStateListener(QObject):
    entityIdChanged = pyqtSignal(str)
    connectionChanged = pyqtSignal(HomeAssistantService)
    stateChanged = pyqtSignal(str, arguments=['state'])

    def __init__(self, parent=None):
        super().__init__(parent)
        self._entityId = None
        self._connection = None
        self._state = {}

    @pyqtProperty(str, notify=entityIdChanged)
    def entityId(self):
        return self._entityId

    @entityId.setter
    def entityId(self, entityId):
        if self._connection is not None and self._entityId is not None:
            self._connection.remove_state_listener(self._entityId)

        self._entityId = entityId
        state = self._connection.add_state_listener(entityId, self._update)
        self.entityIdChanged.emit(self._entityId)
        self._update(state)
    [...]
    @pyqtProperty(str, notify=stateChanged)
    def state(self):
        return self._state.get('state', 'unknown')
```

homeassistantservice.py

```python
    def _update(self, state):
        self._state = state
        self.stateChanged.emit(self.state)
```

main.qml

```
// main.qml

import HomeAssistant 1.0

ApplicationWindow {

    HomeAssistantService {
        id: homeAssistantService
        hostname: "home-assistant.org"
        port: 443
        secure: true
        password: secret.homeAssistantPassword

        Component.onCompleted: {
            homeAssistantService.start()
        }
    }
}
```

HomeAssistantButton.qml

```
import QtQuick 2.0

import HomeAssistant 1.0

import "../../"

AppButton {
    id: homeassistantButton
    property HomeAssistantService connection
    property string entityId
    property string state: listener.state

    property HomeAssistantStateListener listener: HomeAssistantStateListener {
        connection: homeAssistantService
        entityId: homeassistantButton.entityId

        onStateChanged: {
            console.log("new state!!!", state, this.state)
        }
    }

}
```

# Putting everything together

main.qml

```
import QtQuick 2.5
import QtQuick.Controls 2.1
import QtQuick.Controls.Material 2.1
[...]
import "../config"
import "apps/wienerlinien"
[...]
import HomeAssistant 1.0
import Slack 1.0
[...]
ApplicationWindow {
    Material.theme: Material.Dark
    Material.background: "black"
    Material.foreground: "white"
    Material.accent: "#8cd3ff"
    Material.primary : Material.Blue

    SlackService {
        id: slackService
        token: secret.slackToken
        Component.onCompleted: {
            slackService.start()
        }
    }

    Timer {
        id: displayTimeoutTimer
        interval: 60000
        onTriggered: {
            serialService.turnDisplayOff()
        }
    }

    [...]

   FortunePanel {
        id: fortunePanel
        anchors.top: greetingsPanel.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.topMargin: 5
    }

    [...]

    AppPanel {
        id: appPanel
    }
}
```

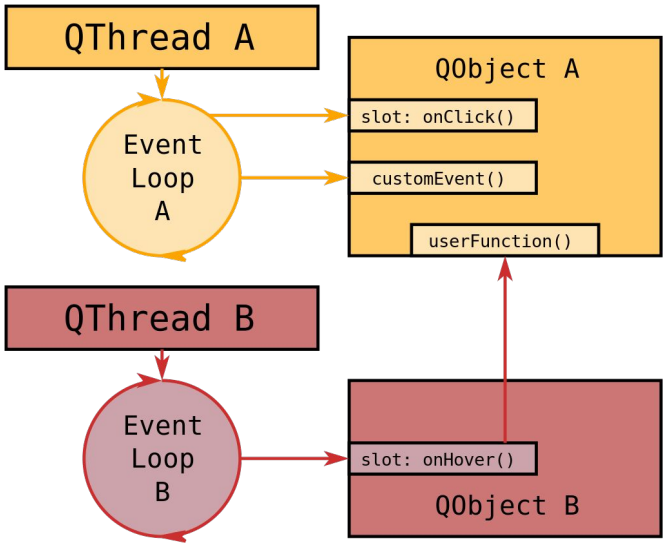
main.py

```
#!../bin/python
from PyQt5.QtGui import QGuiApplication, QTouchDevice
from PyQt5.QtQml import qmlRegisterType, QQmlComponent, QQmlEngine, QQmlApplicationEngine
from quamash import QEventLoop
from dbus.mainloop.pyqt5 import DBusQtMainLoop

DBusQtMainLoop(set_as_default=True)

app = QApplication(sys.argv)

touchscreens = list(filter(lambda d: d.type() == QTouchDevice.TouchScreen, QTouchDevice.devices()))
if touchscreens:
    app.setOverrideCursor(QCursor(Qt.BlankCursor))

loop = QEventLoop(app)
asyncio.set_event_loop(loop)

# Create a QML engine.
engine = QQmlApplicationEngine()
engine.load('./ui/main.qml')

win = engine.rootObjects()[0]
win.show()

with loop:
    loop.run_forever()
```

# Quamash

---

- asyncio needs a (pluggable) event loop
- Qt provides an event loop Implementation (*QEventLoop*)
  - Each QThread instance can have its own event loop
  - signals can be sent across threads
- Quamash bridges those together



# Future possibilities

- Python plasmoids
- Smart-Mirror plasmashell
- Smart-Mirror plasma components

# Thank you!

Any questions?