Application Plugins

To follow this tutorial you should have finished the previous chapter, Standalone Applications.

A application plugin is a DLL that is run from BPS workplace. There are more plugin types, for examle for group access, device interfaces or even SQL drivers. Basically those other plugin types follow the same basics as application plugins, but are out of scope of this tutorial.

Before we start lets create another feature file:

bpsappplugin.prf

This feature copies a built dll file to the `plugins\bpsapp` folder of the proper bps installation.

```
load(bpsinit.prf)
DLLDESTDIR = $$bpsdir/plugins/bpsapp/
```

Create New Project

- Right-click on the main project (`custom`) and run *New Subproject...* to start the *New Subproject* wizard.
- Page *Choose a template*
  - *Library - C++ Library*
  - *Choose...*
- Page *Introduction and Project Location*
  - Name: `myhelloapp`
  - Create in: `C:\dev\mybps\custom`
  - *Next*
- Page *Select Required Madules*
  - [x] QtCore
  - [x] QtGui
  - [x] QtWidgets
  - [x] QtSql
  - [x] QtScript
  - *Next*
- Page *Class Information*
  - Class name: `MyHelloApp`
  - Header file: `myhelloapp.h`
  - Source file: `myhelloapp.cpp`
  - *Next*
- Page *Project Management*
  - *Finish*

This creates the new subproject `myhelloapp` with the qmake file `myhelloapp.pro` and the source files `myhelloapp.cpp`, `myhelloapp.h` and `myhelloapp_global.h`.

We dont need `myhelloapp_global.h` because we write a plugin, not a typical library. So right-click

on `myhelloapp_global.h` and press Del, in the delete dialog check *Delete file permanently* and then *Ok*.

Instead we want another class and create it this way:

- Right-click on the project (`myhelloapp`) and run *Add New...* to start the *New File* wizard.
- Page *Choose a template*
  - *C++ - C++ Class*
  - *Choose...*
- Page *Define Class*
  - ClassName: `MyHelloWindow`
  - Base class: *<Custom>*
    - `BpsApplicationWindow`
  - Header file: `myhellowindow.h`
  - Source file: `myhellowindow.cpp`
  - Path: `C:\dev\mybps\custom\myhelloapp`
  - *Next*
- Page *Project Management*
  - *Finish*

Finally there is another special file needed for Qt 5 plugins:

- Right-click on the project (`myhelloapp`) and run *Add New...* to start the *New File* wizard.
- Page *Choose a template*
  - *General - Empty File*
  - *Choose...*
- Page *Location*
  - Name: `myhelloapp.json`
  - Path: `C:\dev\mybps\custom\myhelloapp`
  - *Next*
- Page *Project Management*
  - *Finish*

Project File

Edit `myhelloapp.pro` to:

```
TARGET = myhelloapp

TEMPLATE = lib

QT += widgets sql script

CONFIG += plugin bpsgui bpsappplugin

SOURCES += \
    myhelloapp.cpp \
    myhellowindow.cpp

HEADERS += \
```

```
    myhelloapp.h\
    myhellowindow.h

OTHER_FILES += \
    myhelloapp.json
```

By adding `plugin` to the `CONFIG` variable Qt knows that a plugin shall be created. With `bpsgui` the BPS API is included and finally `bpsappplugin` tells to copy the created DLL into the right plugins folder of BPS.

JSON File

In the JSON file some meta data can be defined for special Qt plugins, however BPS plugins don't need any and so we create just an empty JSON object.

Edit `myhelloapp.json` to:

```
{}
```

(Thats right, just an opening and closing curly bracket.)

MyHelloApp Class

`myhelloapp.h`:

```
#ifndef MYHELLOAPP_H
#define MYHELLOAPP_H

#include <bpsapplicationplugin.h>

class MyHelloApp : public QObject, public BpsApplicationPlugin
{
    Q_OBJECT
    Q_INTERFACES(BpsApplicationPlugin)
    Q_PLUGIN_METADATA(IID BpsApplicationPlugin_IID FILE "myhelloapp.json")

public:
    MyHelloApp();
    virtual bool init(BpsDatastore* aDatastore);
    virtual QString groupText() const;
    virtual QString text() const;
    virtual QString toolTip() const;
    virtual QIcon icon() const;
    virtual QWidget* createWidget();

private:
    BpsDatastore* mDatastore;
```

```
};

#endif // MYHELLOAPP_H
```

myhelloapp.cpp:

```
#include "myhelloapp.h"
#include "myhellowindow.h"
#include <bpsgui.h>

MyHelloApp::MyHelloApp()
{
} // constructor

bool MyHelloApp::init(BpsDatastore* aDatastore)
{
    mDatastore = aDatastore;
    return true;
} // init

QString MyHelloApp::groupText() const
{
    return tr("My Apps");
} // groupText

QString MyHelloApp::text() const
{
    return tr("My Hello App");
} // text

QString MyHelloApp::toolTip() const
{
    return tr("This is my first BPS application plugin");
} // toolTip

QIcon MyHelloApp::icon() const
{
    return bpsGui->icon(bStr("about"));
} // icon

QWidget* MyHelloApp::createWidget()
{
    return new MyHelloWindow(mDatastore);
} // createWidget
```

This sets up the required information for BPS to know how to integrate our app in the workplace application list, and how to create the window when requested.

MyHelloWindow Class

The *MyHelloWindow* implements a similar window as our standalone application `hello`, therefore also the source looks very familiar:

`myhellowindow.h`:

```
#ifndef MYHELLOWINDOW_H
#define MYHELLOWINDOW_H

#include <bpsapplicationwindow.h>

class BpsDatastore;

class MyHelloWindow : public BpsApplicationWindow
{
    Q_OBJECT

public:
    MyHelloWindow(
        BpsDatastore* aDatastore,
        QWidget *aParent = 0,
        Qt::WindowFlags aFlags = 0);
    virtual void init();
    virtual QSize sizeHint() const;

private:
    BpsDatastore* mDatastore;
};

#endif // MYHELLOWINDOW_H
```

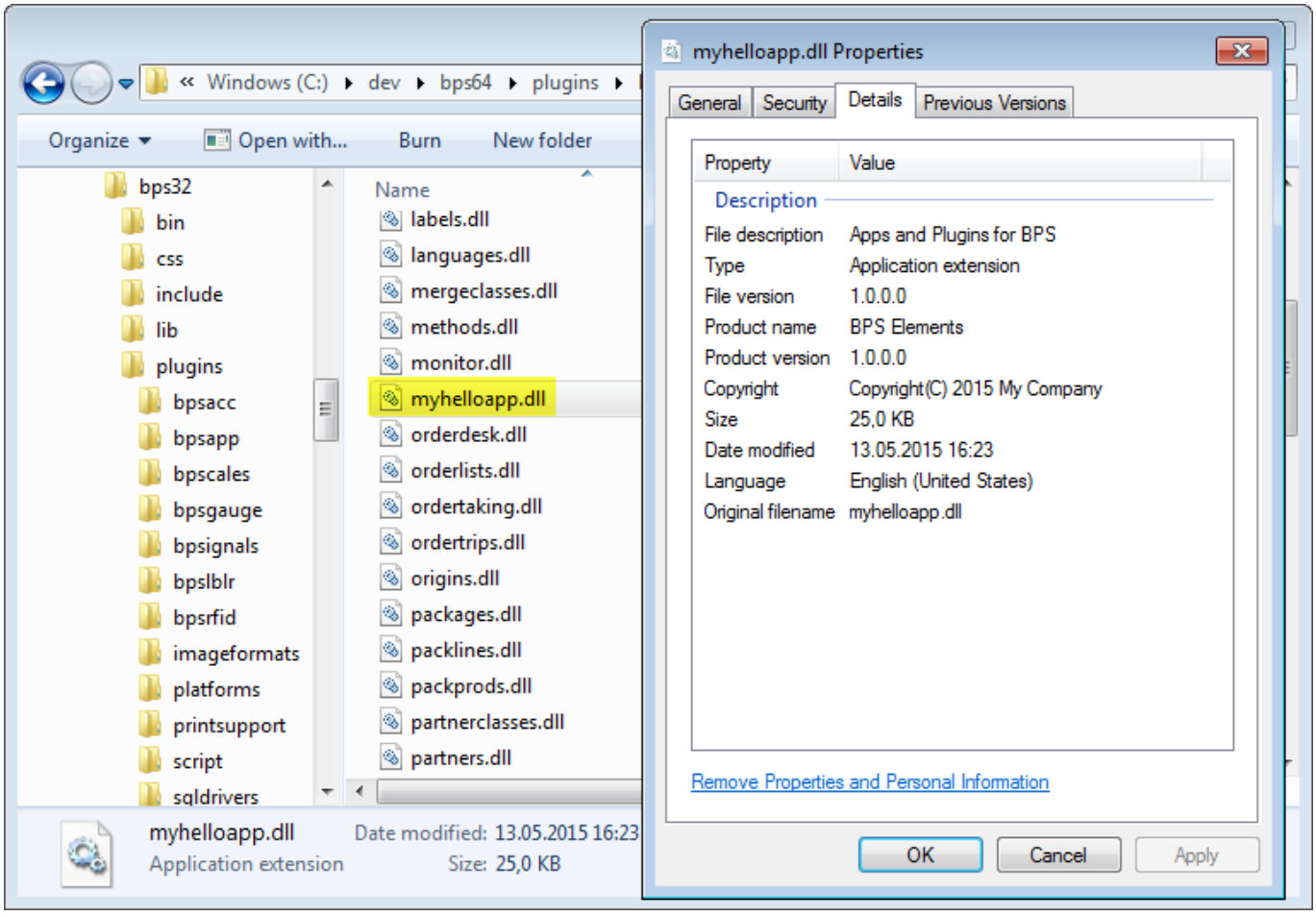
`myhellowindow.cpp`:

```
#include "myhellowindow.h"
#include <bpsgui.h>
#include <QLabel>

MyHelloWindow::MyHelloWindow(
    BpsDatastore* aDatastore,
    QWidget *aParent,
    Qt::WindowFlags aFlags) :
    BpsApplicationWindow(aParent, aFlags)
{
    mDatastore = aDatastore;
    setWindowTitle(tr("Hello BPS"));
    setWindowIcon(bpsGui->icon(bStr("about")));
    QLabel* label = new QLabel;
    label->setPixmap(bpsGui->pixmap(bStr("gears_run"), 64));
    label->setAlignment(Qt::AlignCenter);
```

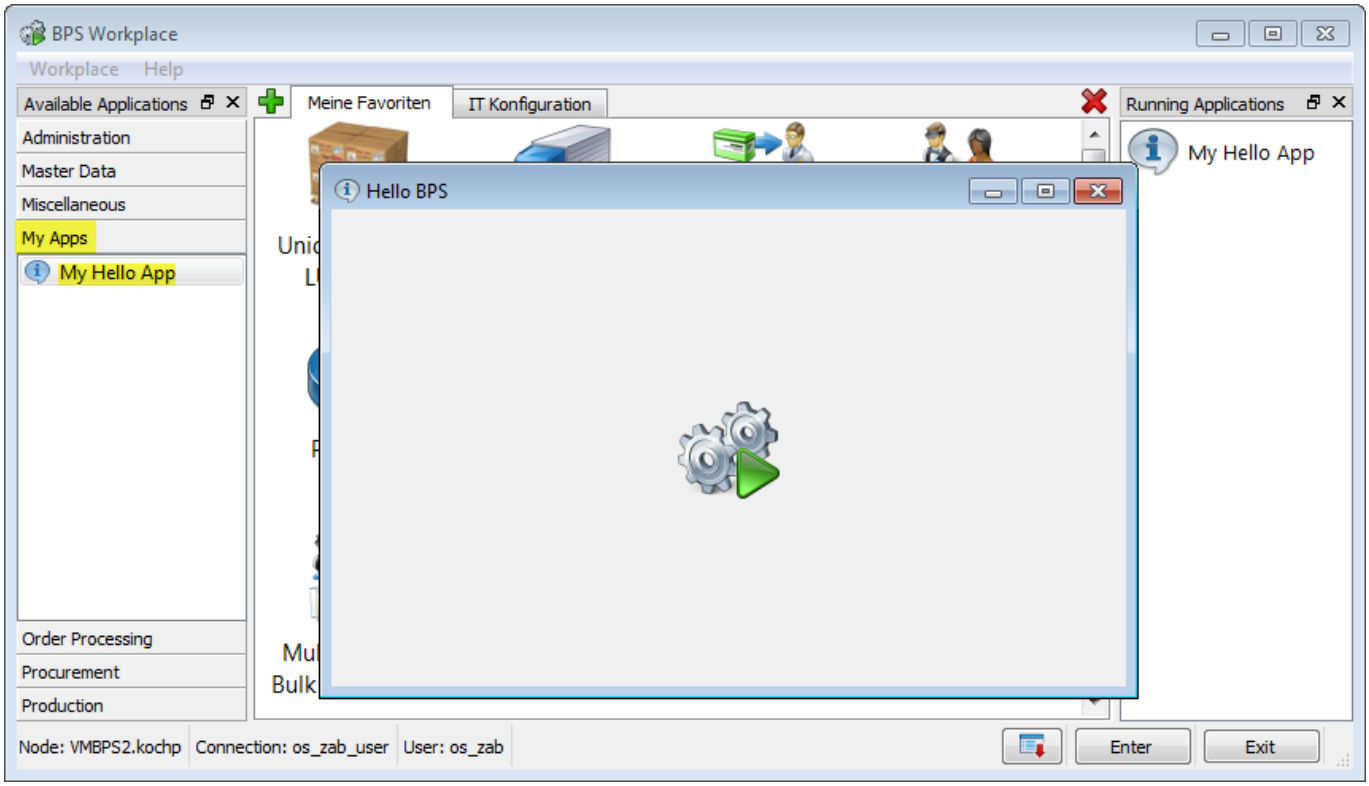
```
    setCentralWidget(label);
} // constructor

void MyHelloWindow::init()
{
    // Could do some initializations here
} // init

QSize MyHelloWindow::sizeHint() const
{
    // Set the initial window size
    return QSize(500, 300);
} // sizeHint
```

Build and Test

Press Ctrl+Shift+B to build all. Try to resolve possible issues by checking that you really followed all instructions exactly. Finally the DLL `myhelloapp.dll` should be created in the `plugins/bpsapp` directory of your BPS installation:



Now run BPS and test your app:



This concludes the jump start into BPS C++ development. Shure there is still much to learn, however learning general C++ and Qt is beyond this tutorial.

Now you should have a good starting point where you can derive your own component developments from.