# Создание своего плагина

Не уверен, что это хороший способ создания динамической библиотеки, но он работает..

Для создание плагина, да и для дальнейшей работы нам понадобится Qt.Поставить его можно либо из репозитория, для этого надо ввести в терминале что-то вроде

sudo yum install qt-devel qt-creator qt-doc

либо скачать с сайта <https://www.qt.io/download-open-source/>, затем перейти в папку загрузки, сделать его испольняемым и выполнить, а дальше появится обычный инсталлер. То же самое, только из терминала:

wget <http://download.qt.io/official_releases/online_installers/qt-opensource-linux-x64-online.run>

chmod +x qt-opensource-linux-x64-1.6.0-8-online.run

./qt-opensource-linux-x64-1.6.0-8-online.run

В репозитории находится qt 4.8.6. Тестовый плагин писался именно для этой версии, как и сама программа. В установочном скрипте используется улилита qmake-qt4, я не знаю, есть ли она в qt5, или какой там у неё аналог, поправьте скрипт в этом случае.

Создадим Qt проект, в котором соберём нашу библиотеку:

1. Заходим в Qt Creator.

2. Файл -> Новый файл или проект ->Другой проект -> Пустой проект Qt->Вводите название проекта -> и т.д.

3. Появился пустой проект и пустой же .pro файл. Этот файл опесывает конфигурацию сборки приложения. На его основе утилитой qmake генерируется make-файл, а уже с его помощью собирается сама программа или плагин.

4. Добавляем в ./pro файл

CONFIG += plugin

CONFIG -= qt

TEMPLATE = lib

5. Внутри папки с проектом предлагается создать следующую структуру каталогов:

include

---models Заголовочные файлы, относящиеся к вашей модели

---RWDFramework Интерфейсы, api RWDFramework

src

---models Исходники вашей модели

можно, наверное, обойтись и без неё, но во-первых так понятно организовано, во-вторых не будет проблем с инклюдами (из-за api RWDFramework). Со временем инклюды, которые лежат в RWDFramework перенесутся в /usr/RWDFramework, но пока лучше оставить так. Отключаем теневую сбоку. Может работает и с ней, но я не пробовал.

6. В папку include/RWDFramework копируем файл Model.h. см аттач..

Подготовительная часть закончена. Теперь перейдём к написанию плагина.

Напишем простейший плагин, которые описывает математический маятник с частотой 2:

1. Создадим новый заголовочный файл osc-model.h. Реализуем в нём интерфейс Integrable,

#ifndef OSCMODEL\_H

#define OSCMODEL\_H

#include <iostream>

#include <include/RWDFramework/Model.h>

using namespace RWDFramework;

class harm\_osc : public Integrable {

public:

~*harm\_osc*() {

}

harm\_osc() {

}

void *operator*() (const state\_t &x , state\_t &dxdt,const double t)

{

dxdt[0] = x[1];

dxdt[1] = -params["w\_0"]\*x[0];

}

};

#endif // OSCMODEL\_H

Поле params объявлено в абстрактном классе Integrable..

2. Реализуем «фабрику» объектов в create\_model

#ifndef CREATE\_MODEL\_H

#define CREATE\_MODEL\_H

#include <include/RWDFramework/Model.h>

using namespace RWDFramework;

extern "C" Integrable \* createIntegrable();

#endif // CREATE\_MODEL\_H

#include <include/models/create\_model.h>

#include <include/models/osc-model.h>

extern "C" Integrable \* createIntegrable() {

Integrable \* osc = new harm\_osc;

return osc;

}

именно эта функция и экспортируется из библиотеки. extern ''C'' значит, что не используется декорирование функций из С++, подробнее см name mangling.

ВАЖНО! Название фабрики должно быть именно createIntegrable с именно такой сигнатурой!

3. Сборка -> Сообрать всё. В корне проекта появится файл lib%ProjectName%.so. Плагин создан.