QtE5 - обёртка Qt-5 для D

1. Введение

Библиотека **QtE5** является результатом моих настойчивых попыток воспользоваться возможностями Qt с языка forth, а потом и с D. Она не является полным "зеркалом"реального Qt-5, но вполне работоспособна (проверена мной) на Qt-5.5 в Windows 32/64 (Win XP, 7, 10), Linux 32/64 (Fedora KDE 23), Mac OSX 64 (10.9 Maveric). Работая над ней хотелось избавится от гигантизма полностью инсталированного Qt (4 Gb) и C++ (3 Gb). Думаю, что цель достигнута. Имея dmd + QtE5 + RunTime Qt-5 вполне можно делать реальные приложения, причем весь комплект помещается на флешке в 2 Gb и не требует инсталяции.

Работая с QtE5 желательно знать принципы работы Qt. Без этого тяжело рассказать почему сделано именно так. По большому счету QtE5 это просто способ работы с Qt, просто немного по другому используя интерфейсы C++ самого Qt.

Я не ставил перед собой цели полностью положить Qt на D. Цель была сделать подмножество, достаточное для повседневных задач, чтобы сократить до минимума, а не отказаться полностью от C++. Qt очень избыточный, одни и те же действия можно выполнить разными способами, а в QtE5 все упрощено до предела. Работая с QtE5 можно пользоваться справочный системой Qt, так как все имена методов и классов повторяют оригинальные имена самой Qt.

Особенно она интересна для начинающих программистов D. Лаконичность, простота сборки, скорость работы и колоссальные возможности :-)

Hello world - классика!

```d
// Файл ex2.d - пример Hello world!
// --=--- компиляция ---=---
// dmd ex2 qte5

import qte5;
import core.runtime;

int main(string[] args) {
	// Загрузить библиотеку
	if (1 == LoadQt(dll.QtE5Widgets, true)) return 1;
	// Создать приложение
	QApplication app = new QApplication(&Runtime.cArgs.argc, Runtime.cArgs.argv, 1);
	// Создать Label окошко
	QLabel lb = new QLabel(null);
	// Вписать в него текст (поддерживает HTML)
	lb.setText("<h1>Привет мир!</h1>").show();
	app.exec();
	return 0;
}
```

Компиляция: dmd ex2 qte5

Обратите внимание на “сложнось” сборки приложения. Ни каких километровых ключей, ни огромных Make файлов. Скорость компиляции - мгновенно. Ещё один плюс, это очень подробная документация по Qt.

2. Установка

Для работы с QtE5 нет необходимости устанавливать полную версию Qt-5. Достаточно установить RunTime версию, что есть просто набор несколь-

ких *.DLL (*.so).

3. QtE5.QAction() - замена метакомпилятора C++

Qt работает со слотами и сигналами. Как их смоделировать? Так как нет возможности вызвать метакомпмилятор, я решил сделать набор готовых слотов, которые метакомпмилятор уже будет знать. В начале я создал совершенно отдельный класс и назвал его QSlot. Однако скоро выяснилось, что нужными мне свойствами обладает уже готовый класс Qt, а именно QAction. Сам QAction уже много умеет, помнит свое имя и т.д. Фактически осталось ему научится вызвать мои обработчики (CallBack функции) и все.

По этому я наследовал QAction и определил в нем набор готовых слотов (qte5widgets.h), а также ввел свойство, адрес вызываемой функции, той функции которая будет вызвана при активации слота в данном конкретном экземпляре QAction. Таким образом каждый экземпляр QAction хранит в себе набор слотов и адрес обработчика. Связать любой слот этого QAction с сигналом Qt можно стандартными средствами, connect() например. Работа слота заключается просто в дальнейшем вызове extern (C) функции, адрес которой запомнен в QAction.

```
Qt ===> Слот{QAction.QtE5} ===> externCНашаФункция() ===> методКласса
```

Итак, слот сработал вызвал мою extern (C) функцию, которая в свою очередь и произвела всю работу как обработчик события. Так как вызов из слота моей функции универсальный (extern (C)) то в самом обработчике особо не развернешься и надо передать выполнение дальше, в метод класса. Все хорошо, но как определить какому конкретному экземпляру каккого класса относится моя обработка? Находясь внутри обработчика (extern (C)) не возможно понять, кто его вызвал. Значит, вызывая обработчик, ему надо передать адрес того экземпляра объекта с которым он должен работать. Получается, что слот должен знать с каким экземпляром объекта он должен работать. Значит слот должен запомнить адрес такого экземпляра. К сожалению я не знаю способа определить адрес экземпляра находясь в его конструкторе. По этому использую конструкцию:

```
QWidget win = QWidget(...); win.savethis(&win);
```

Данная конструкция запоминает адрес конкретного экземпляра в нем самом использую общий для всех объектов метотд savethis() который и производит конкретное запоминание. Получается, что экземпляр объекта может и фактически хранит ссылку на самого себя. Соответственно определяя обработчик, мы уже можем воспользоваться этой информацией.

```
QAction acHelp = new QActin(&обработчик, aThis());
```

где первый параметр адрес обработчика, а второй адрес конкретного экземпляра. aThis - это метод, выдающий замомненный адрес собственного экземпляра объекта.

Соответственно в самом обработчике будет следующая конструкция:

```
extern (C) void acFun1(cast(имя-класса*)uk, ...) {
    (*uk).имяМетода(...);
}
```

и в самом классе уже определяем метод, который будет вести реальную обработку. Получается, что и слот в QAction и мой обработчик extern (C) есть всего навсего транзитеры, передающие вызов друг другу. Это плата за отсутствие метакомпмилятора. В Qt метакомпмилятор проделывает данную работу за нас.

Соответственно определив слот, мы можем вязать его стандартными средствами Qt обычным способом. Самый большой недостаток данного метода в том, что набор предопределеных слотов ограничен. Заранее определить все варианты параметров нет возможности. Пока я добавляю их, слоты, по мере необходимости. К огромному сожалению метакомпмилятор контролирует не только количество параметров, но и их типы.

В QAction QtE5 определен еще один интересный слот. Этот слот может вызвать обработчик передав ему параметр замомненный ранее, так называемую n.

Определение будет следующее:

```
QAction actNtest = new QAction(адрОбработчика, адрЭкземпляра, n);
```

Связывать сигнал теперь надо не с "slot()"а с "slotN()". Соответственно при вызове нашего обработчика ему будет передан этот параметр n. Это позволяет обработать одним обработчиком вызовы разных QAction, просто анализируя, что нам передано в качестве параметра. Это отлично работает, когда надо обработать множество одинаковых кнопок.

Имена слотов предопределены и могут быть использованы в аргументах connect(). Обратите внмание, что имена слотов в кавычках. Они передаются в аргументах connect именно как строки. На текущий момент есть слоты:

1) “Slot()” - Простой слот.

2) “SlotN()” - Слот имеет дополнительный параметр.

3) “Slot_Bool(bool)” - Слот перекидывает с сигнала bool параметр

4) “Slot_Int(int)” - Слот перекидывает int

Обратите внимание на определение нашего обработчика.

extern (C) void testLE() ... обязательны в определении!!!

Зачем так сделано? Дело в том, что QtE5 может быть использована с любого языка программирования поддерживающих формат C обратного вызова. Например из C, C++ и forth.

Хорошо. А как быть с тем, что сигналы с Qt могут иметь параметры?

Например события? Они имеют параметр (указатель) на экземпляр QEvent. Вот тут можно посмотреть вверх на имена предопределённых слотов.

Например, если мы в аргументах connect() напишем “Slot_int(int)” то будет задействован (вызван) слот который возьмет из сигнала параметр типа int и вызовет нашу функцию с этим параметром. Он как эстафету передаст параметр от сигнала к нашему обработчику. Конечно жаль, что нет

возможности не пользуясь C++ определить слот с другим набором аргументов. Но, как показала практика, набора в десяток различных вариантов покрывают почти все сигналы.