МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МУРМАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ЦТМиЭ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ, САМОСТОЯТЕЛЬНЫХ, КОНТРОЛЬНЫХ И РАСЧЕТНО-ГРАФИЧЕСКИХ РАБОТ

По дисциплине: Технологии визуального программирования
название дисциплины

для направления (специальности) _____
код направления (специальности)

«Информатика и вычислительная техника»
наименование направления подготовки

Мурманск
2021

Составитель – Ершов Павел Сергеевич, старший преподаватель кафедры ЦТМиЭ

Методические указания к выполнению расчетно-графической работы рассмотрены и одобрены на заседании кафедры-разработчика:

Цифровых технологий, математики и экономики
название кафедры

21.06.2021 , протокол №12 .
дата

Рецензент – Шиманский С.А., доцент

ОГЛАВЛЕНИЕ

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4
Qt 4
Что такое GTK? 9
Внутри библиотек GTK 10
ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ 21
ЗАДАНИЕ ДЛЯ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ 21
ТРЕБОВАНИЯ К ОТЧЕТУ О ВЫПОЛНЕНИИ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ 21
СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ И ИНТЕРНЕТ-РЕСУРСОВ 22
Приложение 1. ОБРАЗЕЦ ТИТУЛЬНОГО ЛИСТА 23

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Qt

Qt (произносится [ˈkjuːt] (кьют) как «cute»[7] или неофициально Q-T (кью-ти)) — кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby[8]; Java — Qt Jambi[9]; PHP — PHP-Qt и другие.

Со времени своего появления в 1996 году библиотека легла в основу многих программных проектов. Кроме того, Qt является фундаментом популярной рабочей среды KDE, входящей в состав многих дистрибутивов Linux.
Qt – это не только элементы графического интерфейса. Этот фреймворк представляет собой взаимосвязанную систему. Родственность Qt-объектов осуществляется через наследование класса QObject. А связи между ними через сигнально-слотовую систему.В этой статье будут описаны основные классы и полезные особенности этой библиотеки.

QObject
Это базовый класс для всех объектов Qt. Его наследует любой класс использующий сигналы и слоты. Он обеспечивает возможность соединения объектов друг с другом. А также предоставляет к этому полезную функциональность. Во введение этого базового класса находиться:

Прослеживание за потомками и родителями и возвращение указателей на них (Наследственная информация).
Программная реализация таймера
Динамические свойства
Интернационализация приложения
QApplication
Это сердце любого приложения использующего графические элементы.Объект этого класса должен быть создан в единственном экземпляре перед основным кодом программы. В его сферу ответственности входит:

Инициализирует приложение в соответствии с пользовательскими настройками.
Выполняет обработку сообщений и передачу их соответствующим виджетам.
Анализирует аргументы командной строки и соответствующим образом устанавливает свой внутреннее состояние.
Определяет внешний вид приложения через установку стиля и изменение разрешенных цветов.
Предоставляет ссылки на глобальный буфер обмена(clipboard()) и другие полезные объекты.
Следит за открытыми окнами приложения и может выдать информацию о них.
Управляет видом иконки курсора.
И другие полезные функции.
Слоты и сигналы:
Для того чтобы объекты могли общаться, был введён механизм сигналов и слотов. С помощью этого объекты сообщают друг другу о произошедших событиях и отсылают нужные данные. Слот, как и функция-член, может быть публичной, приватной, защищённой или виртуальной. Он вызывается, если приходит подсоединённый к нему сигнал. Он в свою очередь генерируется, если в коде обработки события имеется вызов(макрос) emit signal().

Для себя можно представить определенную схему:

Событие ➞Генерация сигнала➞Передача параметров в слот.

Как выглядит в программе:

«Щелчок по кнопке» ➞ программная обработка ➞ signal(clicked())➞ slot(myslot()).

Основа для создания:
В классе, который должен будет иметь свои сигналы и слоты, нужно произвести наследование от QObject или другого класса библиотеки, наследующего его, и определить макрос Q_Object, с помощью этого объявления компилятор понимает, что необходимо сгенерировать из написанного qt-кода, который использует свои нововведения, в равнозначный шаблонный код стандартного C++. Начиная с Qt 5.0 появился новый способ соединения:

1)Ранняя версия:

```
1
QObject::connect(&Отправитель,SIGNAL(mysignal()),&Адресат,SLOT(myslot()));
```

В шаблонной версии выглядит так:

bool QObject::connect (const QObject * sender, const char * signal, const QObject * receiver, const char * method, type Qt::ConnectionType = Qt::AutoConnection)

Для создания связи необходимо прописать:

Отправителя и его сигнал.
Получателя и его слот.
А также тип соединения, который зависит от того используете вы соединение в потоке или нет. По умолчанию тип соединения определяется автоматически.
Этот старый способ соединения использует механизм обработки строк. Если не существует сигнала или слота с данными именами, то соединения не происходит, а в консоль не отсылается сообщение об ошибке.
Разъединение происходит при уничтожении объекта автоматически. В редких случаях используется такая форма для отключения сигналов:

```
1
QObject::disconnect(&Отправитель,SIGNAL(mysignal()),&Адресат,SLOT(myslot()));
```

Шаблон: bool QObject::disconnect (const QObject * sender, const char * signal, const QObject * receiver, const char * method)
2)connect(&Отправитель,&Class_1::mysignal,&Адресат,&Class_2::myslot);

Шаблон:connect(sender, &Sender::valueChanged, receiver, &Receiver::updateValue);
Достоинства:

Проверка существования сигналов и слота, типов, или если Q_OBJECT отсутствует(выдаёт ошибки).
Параметр может быть определением типа или с различными спецификаторами пространства имен.
Возможность автоматического приведения типов, если есть неявное преобразование (например, от QString до QVariant)
Возможно соединиться с любой функцией-членом класса(произведённого от QObject),а не только со слотами.
Недостатки:

- Более сложный синтаксис(Вы должны определить тип своего объекта)
- Очень сложный синтаксис в случаях перегрузок
- Параметры по умолчанию в слоте теперь не поддерживаются.

Нововведение: возможность соединения с простой функцией.

Благодаря новому синтаксису, появилась возможность соединения с функциями не являющимися слотами.
Пример:connect(sender, &Sender::valueChanged, saveFunction);

При создании такого вида соединения не происходит автоматического разъединения при уничтожении получателя! Это нужно делать вручную.
Такая форма используется для разъединения:

Шаблон:QObject::disconnect(sender, &Sender::valueChanged, receiver, &Receiver::updateValue);
Оба способа соединения разрешены для применения в программе. Так-как у них имеются различия нужно выбирать тот способ, который лучше подходит под ваши нужды.

Разберём на примере использование слотов и сигналов в программе.

Пример:
Добавьте в созданный на предыдущем уроке "sppstudio" новый подпроект lesson_2(Qt Widget).

Очистите его от посторонних файлов, оставив в нём main.cpp.Создайте в нем файлы myclass.h и myclass.cpp.
Давайте придумаем функциональность для нашей программы. Пусть она при нажатии на кнопку создаёт окна и если число созданных окон превысит 5, то она закончит выполнение.

Перед вами реализация её класса:

```
//myclass.h
#ifndef MYCLASS
#define MYCLASS
#include <QLabel>
#include <QPushButton>
#include <QDialog>
#include <QHBoxLayout>
#include <QDebug>

//отображение числа нажатий
class MyClass:public QObject
{
Q_OBJECT
private:
    QPushButton * butt;
    QHBoxLayout * hbox;
    QWidget *window;
    static int counter;//счетчик объектов
public:
    MyClass();
    ~MyClass()
    {
        qDebug()<<"delete object MyClass";
        delete butt;
        delete hbox;
        delete window;
    }

private slots:
    void incCounter();//добавляет единицу к counter при вызове
    void newWindow();//создаёт новое окно
signals:
    s_transmitter();//сигнал передатчик
};
#endif // MYCLASS
```

В заголовочном файле прописываем слоты, сигнал и члены класса.

Обратите внимания на макросы:

Q_Object — нужно прописывать в любом классе использующего свои слоты и сигналы.

private slots: и signals: — здесь собственно они прописываются.

Определения методов у нас находятся в файле .cpp

Перейдём к его детальному рассмотрению:

```
//myclass.cpp
#include "myclass.h"
#include <QString>
#include <QApplication>

MyClass::MyClass()
{
    butt = new QPushButton(" Создать ещё окно ");

    QString str;
    str = "Экземпляр №"+QString::number(counter);
```

```
    hbox = new QHBoxLayout;
    hbox->addWidget(butt);

    connect(butt,SIGNAL(clicked()),this,SLOT(incCounter()));
    connect(this,SIGNAL(s_transmitter()),this,SLOT(newWindow()));
    /*
    *Это старая версия соединения сигнала и слота
    *А ниже располагается новая:
    *connect(butt,&QPushButton::clicked,this,&MyClass::incCounter);
    *Вы можете выбрать эту версию
    *Для этой маленькой программы не будет большой разницы какой тип вы
используете
    */

    window = new QWidget;
    window->setWindowTitle(str);
    window->setLayout(hbox);
    window->resize(230,45);
    window->move(200,80*counter);
    window->show();
}
void MyClass::incCounter()
{
    ++counter;
    if(counter>5)
    {
        QCoreApplication::quit();
        return;
    }
    emit s_transmitter();
}

void MyClass::newWindow()
{
    MyClass * m = new MyClass();
    m->setParent(this);
}
int MyClass::counter = 1;
```

Со строки 6 начинается описание конструктора MyClass. В нём создаётся окно с кнопкой и соединяются слоты с сигналами. В 16-17 происходит соединение старого типа. За настройки вида приложения отвечают строки 27-31.

27: Установка названия окна

29: Размеры окна по x и y

30: Перемещает каждое новое окно вниз чтобы не нагромождать их на одном месте.

Последняя строка отвечает за появление окна на дисплее.

Функция incCounter отвечает за инкриминирование счётчика, вызов сигнала(строка 41) и выход из программы(36-40).

Функция newWindow создаёт новое окно и устанавливает ему родителя.Им становится объект у которого была нажата кнопка.

Установка родителя — это полезная возможность не следить за удалением объектов в динамической памяти, так как класс QObject берёт на себя все обязательства по удалению объектов-потомков при уничтожении родителя.

//main.cpp

#include <QApplication>

```
#include "myclass.h"
int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   MyClass myclass;
   a.exec();
   return 0;
}
```
И наконец main.cpp.

В нём прописывается автоматическая переменная-объект, которая удаляется при завершении программы.

Резюме:
Собственно в следующих статьях будет произведён упор не на описании базовых функций, а об использовании полезных особенностей библиотеки на практике. Здесь вы можете найти русский перевод документации по Qt 4.3 – 4.8. Описываются классы библиотеки, основные свойства и инструменты разработки.

Qt – это мощный инструментарий, имеющий множество способов для воплощения задумок программиста в жизнь. Давайте начнём его освоение с самой первой программы
Итак, у вас под рукой имеется среда разработки QtCreator с подключенным к ней компилятором(например MinGW). В среде разработки выбираем Другой проект-> Проект с поддиректориями.

Название: cppstudio. Здесь для удобства мы будем хранить все приложения взятые с сайта. Создадим здесь под проект QWidget с название lesson_1 и удаляем в нём файлы mainwindows.h, mainwindows.cpp, и форму.

Переписываем код программы печатающей на экран «Привет мир!» в main.cpp:

```
#include <QApplication>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication qapp(argc, argv);
    QHBoxLayout * hlayout = new QHBoxLayout;
    QLabel *label = new QLabel("Привет мир!");
    QPushButton *butt = new QPushButton("Exit");
    QWidget *mainWindow = new QWidget;
    QObject::connect(butt,SIGNAL(clicked()),&qapp,SLOT(quit()));
    hlayout->addWidget(label);
    hlayout->addWidget(butt);
    mainWindow->setLayout(hlayout);
    mainWindow->show();
    return qapp.exec();
}
```

Давайте разберём код программы:
В строчках 1-4 идёт подключение нужных заголовочных файлов.
В qt за графический интерфейс пользователя отвечает семейство классов <QWidget>. Так как нам понадобятся виджеты, то в первую очередь прописываем нужные библиотеки(QLabel, QPushButton).
#include <QApplication>
QApplication qapp(argc, argv);
Одноимённый класс, подключаемый через заголовочный файл QApplication нужен в любом Qt-проекте в единственном экземпляре. Он отвечает за отслеживание событий и сообщений в системе, изменения вида окон приложения и другие полезные функции.

QHBoxLayout * hlayout = new QHBoxLayout;
QLabel *label = new QLabel("Hello World");
QPushButton *butt = new QPushButton("Exit");
QWidget *mainWindow = new QWidget;

В этой части через динамическую память резервируется место под классы.
1
QObject::connect(butt,SIGNAL(clicked()),&qapp,SLOT(quit()));
Через метод класса QObjekt::connect(&Отправитель,SIGNAL(сигнал()),получатель,SLOT(слот()))
реализуется механизм сигналов и слотов. В данном случае при нажатии кнопки приложения генерируется сигнал clicked(), который активирует слотquit() и приводит к закрытию приложения.

layout->addWidget(label);
layout->addWidget(butt);
Для расположения и компоновки виджетов “в сетке”, горизонтально или вертикально используются классы QHBoxLayout, QGridLayout, QHBoxLayout. Метод-член addWidget добавляет их в менеджер компоновки.

mainWindow->setLayout(hlayout);
mainWindow->show();
Последние строки устанавливают окну класс компоновки с его виджетами и заставляют его стать видимым.

GTK

GTK (ранее GTK+[8]; сокращение от GIMP ToolKit) — кроссплатформенная библиотека элементов интерфейса (фреймворк), имеет простой в использовании API, наряду с Qt является одной из двух наиболее популярных на сегодняшний день библиотек для X Window System.

Будучи изначально частью графического редактора GIMP, она развилась в отдельный проект и приобрела заметную популярность. GTK — свободное ПО, распространяемое на условиях GNU LGPL, позволяющей создавать как свободное, так и проприетарное программное обеспечение с использованием библиотеки. GTK является официальной библиотекой для создания графического интерфейса проекта GNU.

Что такое GTK?

Проще говоря, gtk# - это обертка над gtk+, кроссплатформенным GUI фреймворком.

gtk+ - это мультиплатформенный тулкит для создания GUI. Предоставляя полный набор виждетов, gtk+ пригоден в различных проектах: от одноразовых игрушек до набора приложений уровня предприятия.

(from the gtk+ website (http://www.gtk.org/))

Сегодня gtk+ работает с любым X сервером, Direct Framebuffer'ом и производными в Microsoft Windows NT. Библиотека gtk+ известна от Linux, где она является базисом для построения виджетов рабочей среды GNOME. gtk+ включен практически во все дистрибутивы Linux, и стабильно работает под управлением Windows NT. (в 2000 работала нестабильно, если мне память не изменяет)

Портирование gtk+ на Mac OS X запланированно, но необходима заинтересованность других людей. Это зов к участию.

Одно из наиболее распространенных недоумений состоит в том, что GTK# требует Mono для работы. Это неверно. GTK# будет запускаться на любой .NET-совместимой среде. GTK# регулярно тестируется в MS .NET и Mono фреймворках, но также может быть запущена в любой полностью совместной среде. Это означает, что если Вы пишите приложение на GTK# и хотите запустить его в Windows, Вы можете развернуть проект только с GTK# с использованием среды MS, или развернуть его в среде Mono для Windows.

2. Установка

2.1 Скачивание/Установка

Первую вещь, которую необходимо сделать - это скачать GTK# и установить его.

Linux, MacOSX, FreeBSD и другие: Проверьте наличие требуемых пакетов gtk-sharp, mono в вашем дистрибутиве.

aptitude install mono gtk-sharp2

для Debian-based дистрибутивов.
В SuSE Linux выберите нужные пакеты в YaST. Если их нет в дистрибутиве, тогда ищите здесь(http://www.mono-project.com/Downloads) и загружайте. Если на mono-project.com нет пакетов для Вашей платформы, тогда Вам придется скомпилировать из исходников ;)

Windows: программируйте Windows.Forms, не хрен лезть в GTK =) [примечание автора]

Внутри библиотек GTK

Компоненты GTK#
GTK# состоит из следующих сборок, каждая соответствует подобной библиотеке:

gtk-sharp (http://www.mono-project.com/monodoc/N:Gtk)
Связи тулкита gtk+ 2.x для создания GUI

glib-sharp (http://www.mono-project.com/monodoc/N:Glib)
Связи тулкита glib 2.x, которые обеспечивают низкоуровневое библиотеки ядра для gtk+ (не-GUI)

pango-sharp (http://www.mono-project.com/monodoc/N:Pango)
Связи Pango, высокоуровневая библиотека компоновки и рендеринга международных текстов

atk-sharp (http://www.mono-project.com/monodoc/N:Atk)
связи к atk фреймворку

gdk-sharp (http://www.mono-project.com/monodoc/N:Gdk)
низкоуровневый инструментарий для "рисования", используемый gtk+

glade-sharp (http://www.mono-project.com/monodoc/N:Glade)
Glade# позволяет Вам загружать интерфейсы Glade в программу. Это наиболее простой путь создания GTK# GUI.

art-sharp (http://www.mono-project.com/monodoc/N:Art)
библиотека для работы с векторной графикой и отрисовки

rsvg-sharp (http://www.mono-project.com/monodoc/N:Rsvg)библиотека отрисовки SVG

gtk-dotnet (http://www.mono-project.com/monodoc/N:Gtk.DotNet)
интеграция пространства имен Gtk# с System.Drawing

gnome-sharp (http://www.mono-project.com/monodoc/N:Gnome)
Связи GNOME

gnomevfs-sharp (http://www.mono-project.com/monodoc/N:GnomeVfs)
связи файлов, их MIME типов, изображений к методу обращения приложений GNOME'а к файловой системе

vte-sharp (http://www.mono-project.com/monodoc/N:Vte)
связи к терминальному эмулятору VTE

gconf-sharp (http://www.mono-project.com/monodoc/N:GConf)
Связи к системе хранения конфигураций в GNOME

gtkhtml-sharp (http://www.mono-project.com/monodoc/T:Gtk.HTML)
Связи к легковесному HTML виджету

Другие компоненты не включены в основной дистрибутив GTK#, но заслуживают упоминания, потому что имеют отношение к GTK#: Gecko#, Gtksourceview#, Gsf#, Guile#, Gst# и dbus#.

2.3 GTK# или Glade#

Когда люди начинают изучение Mono, они могут стоять на перепутье - какую библиотеку использовать: gtk# или glade#. Для этого Вам нужно понять что такое glade# и что такое gtk#. gtk# - это ядро для построения оконных и виджет систем. Glade# наследует GTK#, таким образом является подмножеством GTK# и является совместимой, но glade# автоматически компонует виджеты с сохраняет их как XML файл. XML компоновка GUI может быть сгенерирована при помощи Glade утилиты, студии дизайна WYSIWYG.

Для большинства окон Glade# - лучший выбор. Он экономит время за счет отсутствия необходимости написания программного кода для GUI и делает простым изменение интерфейса в будущем. Но возникает проблема в том, что Вам нужно от интерфейса, Glade# не может выполнять некоторые специфичные вещи (скрывать элементы, динамически загружать новые части, наследовать виджеты). Только опыт может Вам понять, что в конкретном случае лучше использовать.

Для быстрого ознакомления с Glade# Вы возможно захотете взглянуть на этот скринкаст(http://nat.org/demos/gtksharp.html) Ната Фридмана (Nat Friedman)(http://nat.org), в котором он создает простое графическое приложение всего за несколько минут.

3. Первое GTK# приложение

Шаг 1. Присядьте удобнее

Мы должны чувствовать себя комфортно. Откройте Dr. Pepper и включите любимую музыку. Хорошо, теперь мы готовы.

Шаг 2. Создание папок и файлов

Для начала нам нужно создать новую директорию для маленького проекта. (Пользователи Windows: давайте не будем использовать пробелы в названии директории, чтобы избежать в дальнейшем головной боли).

Откройте shell (Если вы в Windows, откройте меню cПуск :) далее Программы -> Mono 1.x.x -> Mono Command Prompt. Она автоматически установит нужные пути к mono библиотекам.) Перейдите в только что созданную директорию. Мы часто будем использовать консоль, поэтому оставьте ее запущенной.

Вернемся к делу. Откройте свой любимый редактор (MonoDevelop, vi, emacs, notepad и т.д.) и создайте новый пустой проект (если это возможно) или создайте новый пустой файл. Сохраните файл под именем "helloworld.cs".

Шаг 3. Формирование кода

Я надеюсь, что Вы уже знакомы с C#, и код написанный ниже не вызовет никаких проблем в понимании. Мы должны создать новый класс, использовать Gtk# и указать точку входа в нашу программу. Это будет выглядеть так:

```
using System;
using Gtk;

public class GtkHelloWorld {

public static void Main() {
Console.WriteLine("HelloWorld");
}

}
```

Это должно выглядеть весьма знакомо для Вас. Только теперь мы можем воспользоваться компилятором. Сохраним исходный код, перейдем в консоль и построим проект:

```
mcs -pkg:gtk-sharp-2.0 helloword.cs
```

Для тех кто пользовался csc компилятором в Windows параметр "-pkg:" может показаться незнакомым. Этого параметра не было в csc, потому что Mono пришел из мира Linux. Этот параметр позволяет указать на необходимость подключения пакета gtk-sharp-2.0. Т.е. система ищет файл "gtk-sharp-2.0.pc", который содержит данные о местоположении библиотеки для этого пакета (среди другой информации). Т.е. мы не должны вводить "-r:gtk-sharp-2.0.dll -r:atk-sharp-2.0.dll -r:pango-sharp-2.0.dll ...." руками.

Шаг 4. Добавление графического интерфейса GUI

Теперь давайте вернемся обратно к нашему коду. Уберем оператор "Console.WriteLine". Первое, что мы сделаем - создадим новое окно. Проделаем это добавлением нового оператора new Window и блока приложения (для начала нити цикла main). Вот так:

```
using System;
using Gtk;

public class GtkHelloWorld {


public static void Main() {
Application.Init();

//Create the Window

Window myWin = new Window("My first GTK# Application! ");
myWin.Resize(200,200);

//Create a label and put some text in it.

Label myLabel = new Label();
myLabel.Text = "Hello World!!!!";

//Add the label to the form
myWin.Add(myLabel);

//Show Everything

myWin.ShowAll();

Application.Run();
}
}
```

Теперь скомпилируем исходный код так же как мы делали это раньше, и запустим программу

```
mono HelloWorld.exe
```

В итоге вы получите что-то вроде этого:

Не так уж и сложно, да?

Первая вещь, которую Вы могли заметить, это то, что в отличии от использования System.Windows.Forms мы не писали код для точной компоновки текста в окне. Например, мы не писали 'myLabel.Left = 100' или 'myLabel.Width = 200' или что-то подобное для размещения текстовой метки на форме, мы просто пишем 'myWin.Add(...)'. И все это потому, что 'Gtk.Window' - это виджет, который наследуется от Bin, или одиночного виджета который размещен в контейнере Container.

Другая часть кода, которая могла Вас заинтересовать, это использование выражений "Application.Init()" и "Application.Run()". Если вы когда-либо ранее использовали System.Windows.Forms это аналогично

использованию "Application.Run()" во многих случаях. Обычно, когда приложения заканчивает обработку любого кода в основном потоке, приложение останавливается. Команда "ShowAll()" не блокирует код и продолжает дальнейшее выполнение кода (вплоть до остановки). Команда "Application.Init()" говорит оболочке выполнения "слушать" сигналы поступающие от Gtk.Windows и в момент когда выполняется "Application.Run()" выполнение кода передается основному циклу сообщений. Это позволяет оставаться приложению запущенным до тех пор пока не будут закрыты все окна. Для большей информации смотрите информацию об объекте Application.

Шаг 5. Формирование окна

Возможно Вы захотите спросить себя "Как я смогу добавить новый виджет на окно, если оно может содержать только один виджет?" До этого мы говорили, что Window действительно может содержать в себе только один виджет, но виджет сам по себе может содержать в себе множество других виджетов. Некоторые из этих виджетов наследуются от контейнера Gtk.Box, а в некоторых случаях напрямую от контейнера. Контейнерный виджет Bin наследуется напрямую от виждета-контейнера, как и многие другие виджеты, но Bin может содержать в себе только один элемент управления.

Для того чтобы размещать большое количество виджетов в нашем окне, мы должны добавить на окно один из виджетов, который может содержать в себе другие виджеты. Существует множество виджетов, которые могут делать это, но мы затронем только некоторые простые: HBox(http://www.go-mono.com/docs/index.aspx?link=T%3aGtk.HBox), VBox(http://www.go-mono.com/docs/index.aspx?link=T%3aGtk.VBox) и возможно Table(http://www.go-mono.com/docs/index.aspx?link=T%3aGtk.Table).

Шаг 6. Добавление событий

Все классы производные от 'Widget (http://www.go-mono.com/docs/index.aspx?link=T%3aGtk.Box)' предоставляют следующие события:
ButtonPressEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ButtonPressEvent)
ButtonReleaseEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ButtonReleaseEvent)
ScrollEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ScrollEvent)
MotionNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.MotionNotifyEvent)
DeleteEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.DeleteEvent)
DestroyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.DestroyEvent)
ExposeEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ExposeEvent)
KeyPressEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.KeyPressEvent)
KeyReleaseEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.KeyReleaseEvent)
EnterNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.EnterNotifyEvent)
LeaveNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.LeaveNotifyEvent)
ConfigureEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ConfigureEvent)
FocusInEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.FocusInEvent)
FocusOutEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.FocusOutEvent)
MapEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.MapEvent)
UnmapEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.UnmapEvent)
PropertyNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.PropertyNotifyEvent)
SelectionClearEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.SelectionClearEvent)
SelectionRequestEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.SelectionRequestEvent)
SelectionNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.SelectionNotifyEvent)
ProximityInEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ProximityInEvent)
ProximityOutEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ProximityOutEvent)
VisibilityNotifyEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.VisibilityNotifyEvent)
ClientEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.ClientEvent)
NoExposeEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.NoExposeEvent)
WindowStateEvent (http://www.go-mono.com/docs/monodoc.ashx?link=F%3aGtk.Widget.WindowStateEvent)

Многие их этих событий могут быть обработаны стандартным обработчиком событий. Например:

public static void HandlerMethod(object obj, EventArgs args)

Пример обработки событий нажатия на кнопку:

public static void ButtonPressHandler(object obj, ButtonPressEventArgs args)

ButtonPressEventArgs - это класс производный от EventArgs. Класс ButtonPressEventArgs, как и многие другие в gtk#, добавляет свойство Gtk.Event (или другое от Gtk.Event) к EventArgs.

Типы Gdk.Event:

EventType.Nothing
EventType.Delete
EventType.Destroy
EventType.Expose
EventType.MotionNotify
EventType.ButtonPress
EventType.TwoButtonPress
EventType.ThreeButtonPress
EventType.ButtonRelease
EventType.KeyPress
EventType.KeyRelease
EventType.EnterNotify
EventType.LeaveNotify
EventType.FocusChange
EventType.Configure
EventType.Map
EventType.Unmap
EventType.PropertyNotify
EventType.SelectionClear
EventType.SelectionRequest
EventType.SelectionNotify
EventType.ProximityIn
EventType.ProximityOut
EventType.DragEnter
EventType.DragLeave
EventType.DragMotion
EventType.DragStatus
EventType.DropStart
EventType.DropFinished
EventType.ClientEvent
EventType.VisibilityNotify
EventType.NoExpose
EventType.Scroll
EventType.WindowState
EventType.Setting

Например, для использования события Gdk.Event мы можем использовать такой код:

```
using Gdk;
...
widget.ButtonPressEvent += new ButtonPressEventHandler(ButtonPressHandler);
...
private void ButtonPressHandler(object obj, ButtonPressEventArgs args) {

  // single click
  if (args.Event.Type == EventType.ButtonPress) {

     ...
  }
  // double click
```

```
  if (args.Event.Type == EventType.TwoButtonPress) {

    ...
  }

  // the left button was used
  if (args.Event.Button == 1) {

    ...
  }
}
```

В примере выше вы можете увидеть как обнаружить было ли одиночное нажатие мышкой или это был двойной клик.

4. Первое Glade# приложение

Шаг 1. Что такое Glade#

Glade# - это набор связей с libglade на языке C#. Позволяет легко создавать GUI приложения используя визуальные средства и сохранять их в формате, который приложение сможет использовать во время выполнения, чтобы создать интерфейс. На данный момент существует две среды для генерирования glade файлов: Glade(http://glade.gnome.org/) и конечно Stetic(http://mysterion.org/~danw/blog/2005/03/stetic).

4.1.1 Что такое glade файлы?

Файлы, записанные в XML формате, которые представляют собой GUI в GTK+, сохраняя сюда все аттрибуты и свойства.

4.1.2 На что похожи файлы .glade?

файл: gui.glade

```
<!--*- mode: xml -*-->

<glade-interface>

<widget class="GtkWindow" id="window1">

<property name="visible">True</property>
<property name="title" translatable="yes">Glade Window</property>

<property name="type">GTK_WINDOW_TOPLEVEL</property>
<property name="window_position">GTK_WIN_POS_CENTER</property>

<property name="modal">False</property>
<property name="default_width">256</property>

<property name="default_height">256</property>
<property name="resizable">True</property>

<property name="destroy_with_parent">False</property>
<property name="decorated">True</property>

<property name="skip_taskbar_hint">False</property>
<property name="skip_pager_hint">False</property>

<property name="type_hint">GDK_WINDOW_TYPE_HINT_NORMAL</property>
<property name="gravity">GDK_GRAVITY_NORTH_WEST</property>
```

```
<property name="focus_on_map">True</property>

<child>
<widget class="GtkScrolledWindow" id="scrolledwindow1">

 <property name="visible">True</property>
 <property name="can_focus">True</property>

 <property name="hscrollbar_policy">GTK_POLICY_ALWAYS</property>
 <property name="vscrollbar_policy">GTK_POLICY_ALWAYS</property>

 <property name="shadow_type">GTK_SHADOW_IN</property>
 <property name="window_placement">GTK_CORNER_TOP_LEFT</property>


 <child>
 <widget class="GtkLayout" id="layout1">
   <property name="visible">True</property>

   <property name="width">400</property>
   <property name="height">400</property>

   <property name="hadjustment">0 0 400 10 212.4 236</property>
   <property name="vadjustment">0 0 400 10 212.4 236</property>


   <child>
     <widget class="GtkLabel" id="label1">
       <property name="width_request">38</property>

       <property name="height_request">17</property>
       <property name="visible">True</property>

       <property name="label" translatable="yes">label1</property>
       <property name="use_underline">False</property>

       <property name="use_markup">False</property>
       <property name="justify">GTK_JUSTIFY_LEFT</property>

       <property name="wrap">False</property>
       <property name="selectable">False</property>

       <property name="xalign">0.5</property>
       <property name="yalign">0.5</property>
       <property name="xpad">0</property>
       <property name="ypad">0</property>

       <property name="ellipsize">PANGO_ELLIPSIZE_NONE</property>
       <property name="width_chars">-1</property>

       <property name="single_line_mode">False</property>
       <property name="angle">0</property>

     </widget>
     <packing>
       <property name="x">96</property>
       <property name="y">88</property>

     </packing>
   </child>

   <child>
```

```
      <widget class="GtkButton" id="button1">

        <property name="width_request">60</property>
        <property name="height_request">27</property>

        <property name="visible">True</property>
        <property name="can_focus">True</property>

        <property name="label" translatable="yes">button1</property>
        <property name="use_underline">True</property>

        <property name="relief">GTK_RELIEF_NORMAL</property>
        <property name="focus_on_click">True</property>

      </widget>
      <packing>
        <property name="x">88</property>
        <property name="y">168</property>

      </packing>
    </child>
  </widget>
  </child>
</widget>
</child>

</widget>

</glade-interface>
```

Файл .glade содержит в себе всю необходимую информацию для того, чтобы библиотека libglade могла воссоздать GUI.

Шаг 2. Интеграция glade файлов с нашей программой

Подразумевается что .glade файл уже создан, или при помощи Glade, или при помощи Stetic. Видео о Stetic можно посмотреть здесь (http://mysterion.org/~danw/blog/2005/03/steticzilla.html).

В намерениях нашего примера мы предпологаем, что GUI был сохранен в файл gui.glade, который содержит описание окна window1, кнопки button1 и метки label1.

Нам нужно будет создать новый указатель на Gtk# и Glade#, а затем создать новый класс и точку входа, с которой начинается наша программа.

```
// file: glade.cs
using System;
using Gtk;
using Glade;
public class GladeApp

{
   public static void Main (string[] args)

   {
           new GladeApp (args);
   }

   public GladeApp (string[] args)
   {

           Application.Init();
```

```
            Glade.XML gxml = new Glade.XML (null, "gui.glade", "window1",
null);
            gxml.Autoconnect (this);
            Application.Run();
   }

}
```

4.2.1 Как скомпилировать?

Теперь мы должны скомпилировать исходный файл glade.cs указывая пространство имен для glade, которое находится в библиотеке glade-sharp. Команда компиляции следующая:

$ mcs -pkg:glade-sharp -resource:gui.glade glade.cs

Коммандой mcs -pkg:glade-sharp мы создаем программу glade.exe, а опция -resource внедряет файл gui.glade в исполняемую программу.

Передавая null как первый параметр в конструктор Glade.XML, мы сообщаем библиотеке libglade загружать glade файл из ресурсов, как вариант использования конструктора, файл может быть загружен из файловой системы, что особенно полезно тогда, когда Вы не хотите перекомпилировать GUI приложение после каждого изменения .glade файла.

Если мы запускаем программу наш GUI может открыться, однако, нажимая на кнопки Вы не добьетесь эффекта, потому как мы не назначали событий виджету, определенному в gui.glade файле. Изучив следующую секцию Вы научитесь это делать.

Шаг 3. Как использовать Glade# в моем коде

4.4 Как обращаться к виджетам определенным в gui.glade

Для доступа к объектам, определенным в gui.glade файле, Вы должны знать имя объекта и его тип, и только тогда добавлять его в C# код. Делается это следующим образом (обратите внимание на оттрибут [Widget]):

[Widget]
Тип имя;

Применяем это определение к нашему примеру как следует ниже в коде:

```
using System;

using Gtk;
using Glade;
public class GladeApp
{
   public static void Main (string[] args)

   {
            new GladeApp (args);
   }

   public GladeApp (string[] args)
   {

            Application.Init();

            Glade.XML gxml = new Glade.XML (null, "gui.glade", "window1",
```

```
null);
            gxml.Autoconnect (this);
            Application.Run();
    }


   [Widget]
   Button button1;

   [Widget]
   Label label1;
}
```

4.5 Как добавить событие

Для добавления событий Вам необходимо следовать примеру кода ниже. Вы также можете добавлять события из Glade.

```
using System;
using Gtk;
using Glade;
public class GladeApp

{
   public static void Main (string[] args)

   {
            new GladeApp (args);
   }

   public GladeApp (string[] args)
   {

            Application.Init();

            Glade.XML gxml = new Glade.XML (null, "gui.glade",
"window1", null);
            gxml.Autoconnect (this);

            button1.Clicked += OnPressButtonEvent;

            Application.Run();
   }


  [Glade.Widget]
  Button button1;

  [Glade.Widget]
  Label label1;

  public void OnPressButtonEvent( object o, EventArgs e)

   {
      Console.WriteLine("Button press");
      label1.Text = "Mono";
```

```
    }

}
```

Впоследствии мы увидим готовый код использования событий для объектов, указанных в gui.glade файле.

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Самостоятельно изучить способы построения пользовательских интерфейсов с помощью HTML + CSS в приложениях Electron. Попытаться создать аналогичные приложения.

ЗАДАНИЕ ДЛЯ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ

Целью расчетно-графической работы является получение опыта в оценке качества графического пользовательского интерфейса программного средства.

Задание:

Написать приложение, использующее графический пользовательский интерфейс с использованием одной из технологий, описанных в теоретических материалах.

ТРЕБОВАНИЯ К ОТЧЕТУ О ВЫПОЛНЕНИИ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ

Отчет по выполнению расчетно-графической работы должен содержать:

1. Описание функционала рассматриваемого программного средства в виде списка функций.
2. Демонстрация работающего программного продукта

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ И ИНТЕРНЕТ-РЕСУРСОВ

1. Qt 5.10. Профессиональное программирование на C++ Шлее Макс 2018

Приложение 1. ОБРАЗЕЦ ТИТУЛЬНОГО ЛИСТА

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МУРМАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра
Математики, информационных систем и программного обеспечения

О Т Ч Е Т

о выполнении расчетно-графической работы

по дисциплине

«»

Выполнил:
студент группы _____
Фамилия И.О.

Проверил:
Старший преподаватель
кафедры МИС и ПО
Фамилия И.О.

Мурманск

20__