

Семинар 2 (часть 2)

Создание графических интерфейсов в Qt
Компоновка виджетов. Размерная политика Qt.

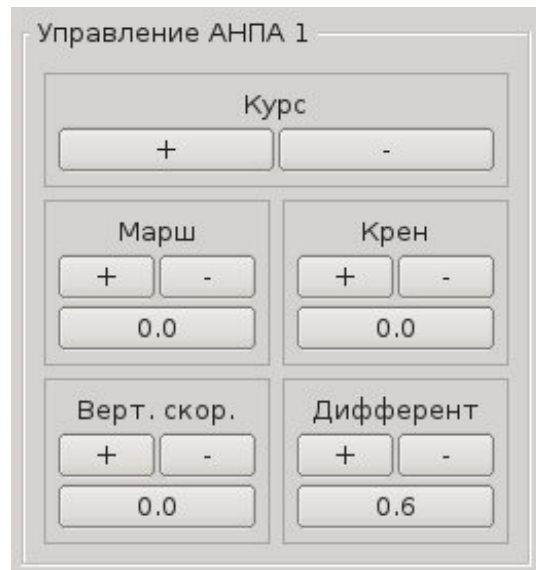
Qt Creator. Создание GUI.

Есть два подхода, которые можно использовать при построении графического пользовательского интерфейса, используя виджеты Qt:

1. Создать, настроить виджеты и разместить их на форме в соответствующих компоновках с помощью программного кода;
2. Воспользоваться визуальным редактором форм Qt Designer.

Создание GUI

Рассмотрим первый вариант. И создадим небольшую панель управления движением ПА по маршру.



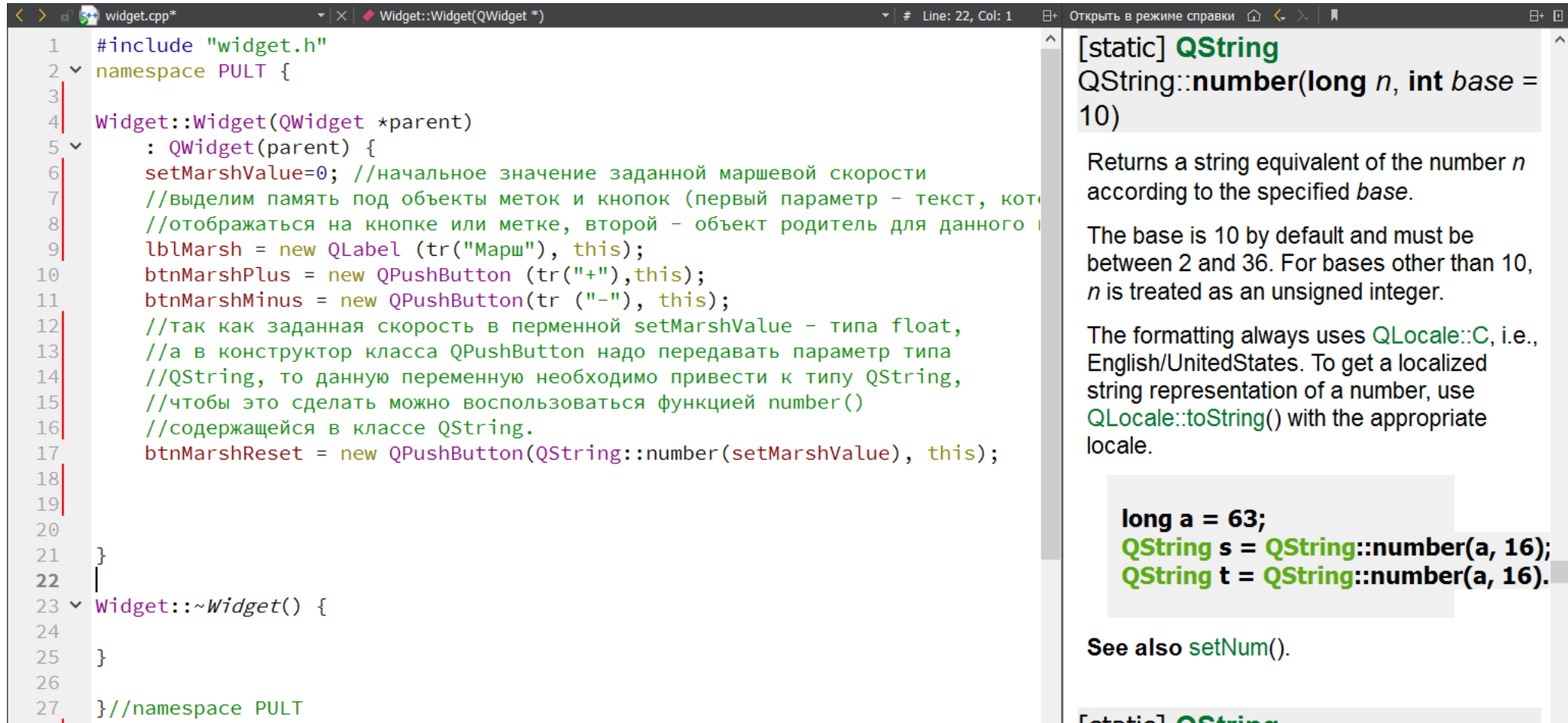
Создание GUI

widget.h

```
< > widget.h* setMarshValue: float
1  #ifndef WIDGET_H
2  #define WIDGET_H
3
4  #include <QWidget>
5  #include <QPushButton> //класс для создания кнопок
6  #include <QLabel> //класс для создания текстовых меток
7
8
9  namespace PULT {
10
11  class Widget : public QWidget {
12      Q_OBJECT
13  public:
14      Widget(QWidget *parent = 0);
15      virtual ~Widget();
16
17  private:
18      QPushButton *btnMarshPlus; //создаем указатели на
19      //объекты классов кнопок и текстовых меток, которые
20      //хотим создать
21      QPushButton *btnMarshMinus;
22      QPushButton *btnMarshReset;
23      QLabel *lblMarsh;
24      float setMarshValue; //переменная, в которой будет храниться
25      //заданное значение по маршруту
26  };
27
28  } //namespace PULT
29  #endif // WIDGET_H
30
```

Создание GUI

widget.cpp



```
1 #include "widget.h"
2 namespace PULT {
3
4 Widget::Widget(QWidget *parent)
5 : QWidget(parent) {
6     setMarshValue=0; //начальное значение заданной маршевой скорости
7     //выделим память под объекты меток и кнопок (первый параметр - текст, кото
8     //отображаться на кнопке или метке, второй - объект родитель для данного
9     lblMarsh = new QLabel (tr("Марш"), this);
10    btnMarshPlus = new QPushButton (tr("+"),this);
11    btnMarshMinus = new QPushButton(tr ("-"), this);
12    //так как заданная скорость в переменной setMarshValue - типа float,
13    //а в конструктор класса QPushButton надо передавать параметр типа
14    //QString, то данную переменную необходимо привести к типу QString,
15    //чтобы это сделать можно воспользоваться функцией number()
16    //содержащейся в классе QString.
17    btnMarshReset = new QPushButton(QString::number(setMarshValue), this);
18
19
20
21 }
22 |
23 ~Widget() {
24
25 }
26
27 } //namespace PULT
```

[static] **QString**
QString::number(long n, int base = 10)

Returns a string equivalent of the number *n* according to the specified *base*.

The base is 10 by default and must be between 2 and 36. For bases other than 10, *n* is treated as an unsigned integer.

The formatting always uses **QLocale::C**, i.e., English/UnitedStates. To get a localized string representation of a number, use **QLocale::toString()** with the appropriate locale.

```
long a = 63;
QString s = QString::number(a, 16);
QString t = QString::number(a, 16).
```

See also **setNum()**.

[static] **QString**

Компоновка элементов в Qt

Компоновщик не принадлежит к виджетам, не наследует от QWidget и является невидимым на форме. Он только управляет её размером и размещением её содержания.

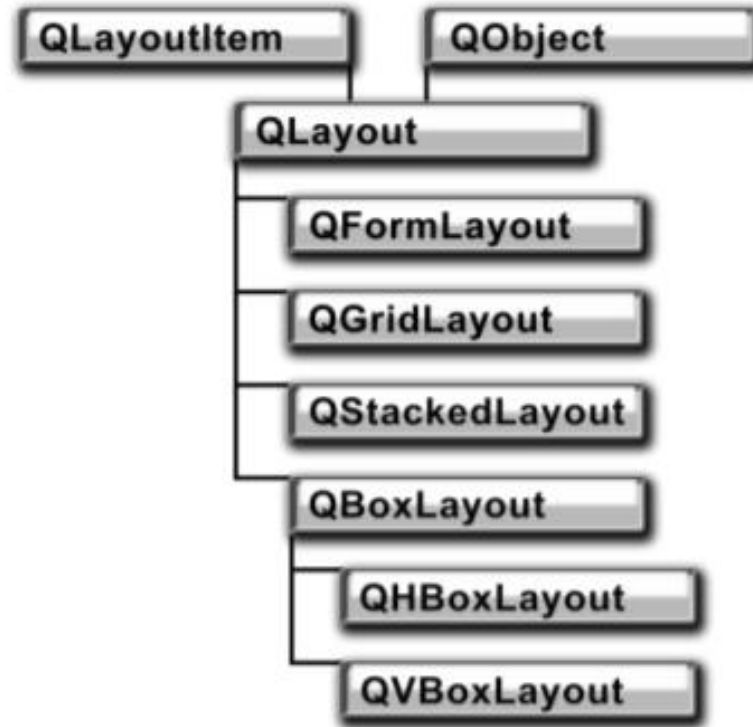


Рис. Иерархия классов менеджеров
компоновки

Компоновка элементов в Qt

Основные методы классов менеджеров компоновки:

`setSpacing()` – расстояние между виджетами компоновки (в пикселях);

`setMargin()` – отступ виджетов от границ компоновки;

`addWidget()` – добавление виджета в компоновку;

`addLayout()` – добавление встроенного менеджера компоновки;

`removeWidget()` – удаление виджета.

Для размещения компоновки на виджете используется функция [`QWidget::setLayout\(\)`](#).



Компоновка элементов в Qt. QHBoxLayout.

Объекты класса QHBoxLayout упорядочивают все виджеты только в горизонтальном порядке, слева направо.

В .h файле добавить:

```
#include <QHBoxLayout>
```

В классе в разделе public объявить

```
QHBoxLayout * hMarshLayout;
```

В .cpp

```
//Создадим объект класса QHBoxLayout для
```

```
горизонтального размещения дочерних виджетов
```

```
hMarshLayout = new QHBoxLayout();
```

```
//Метод addWidget добавляет виджеты (метки,  
кнопки и тп) в компоновку
```

```
hMarshLayout->addWidget(lblMarsh);
```

```
hMarshLayout->addWidget(btnMarshPlus);
```

```
hMarshLayout->addWidget(btnMarshMinus);
```

```
hMarshLayout->addWidget(btnMarshReset);
```

```
//чтобы "привязать" созданую компоновку к  
виджету предусмотрен метод
```

```
setLayout() this->setLayout(hMarshLayout);
```

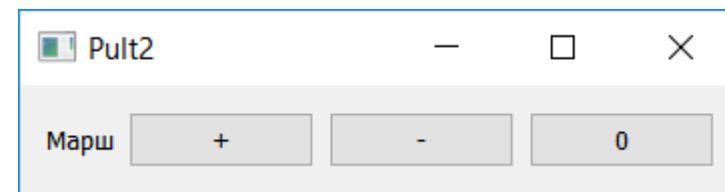


Рис. Горизонтальная
компоновка виджетов

Компоновка элементов в Qt. QVBoxLayout.

Объекты класса QVBoxLayout упорядочивают все виджеты только по-вертикали – сверху вниз.

Если .h файле добавить:

```
#include <QVBoxLayout>
```

А в классе предыдущего примера в разделе public объявить
QVBoxLayout * vMarshLayout;

А в .cpp – файле создать объект vMarshLayout класса
QVBoxLayout и заменить в коде hMarshLayout на
vMarshLayout, то полученный результат компоновки будет
представлять собой рис.

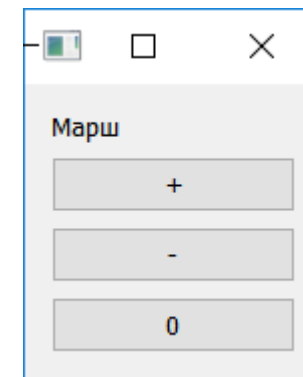


Рис. Вертикальная
компоновка виджетов

Компоновка элементов в Qt. Вложенные размещения

Размещая одну компоновку в другой, можно создавать размещения практически любой сложности (рис.).

Для организации вложенных размещений существует метод `addLayout()`.

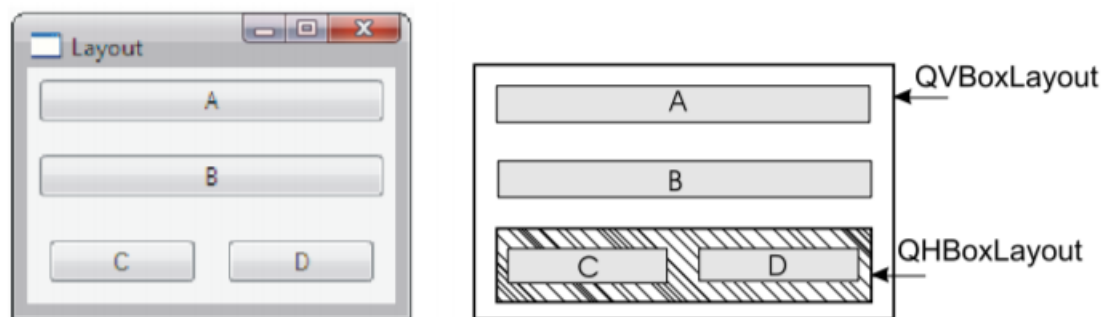


Рис. Вложенная компоновка

Компоновка элементов в Qt. Вложенные размещения.

widget.h

```
< > widget.h* vMarshLayout: QVBoxLayout *
1  #ifndef WIDGET_H
2  #define WIDGET_H
3
4  #include <QWidget>
5  #include <QPushButton> //класс для создания кнопок
6  #include <QLabel> //класс для создания текстовых меток
7  #include <QVBoxLayout>
8
9  namespace PULT {
10
11  class Widget : public QWidget {
12      Q_OBJECT
13  public:
14      Widget(QWidget *parent = 0);
15      virtual ~Widget();
16
17  private:
18      QPushButton *btnMarshPlus; //создаем указатели на
19      //объекты классов кнопок и текстовых меток, которые
20      //хотим создать
21      QPushButton *btnMarshMinus;
22      QPushButton *btnMarshReset;
23      QLabel *lblMarsh;
24      float setMarshValue; //переменная, в которой будет храниться
25      //заданное значение по маршруту
26
27      //объявим указатели на объекты менеджеров компоновки
28      QVBoxLayout *vMarshLayout;
29      QHBoxLayout *hMarshLayout;
30
```

Компоновка элементов в Qt. Вложенные размещения.

widget.cpp (часть 1)

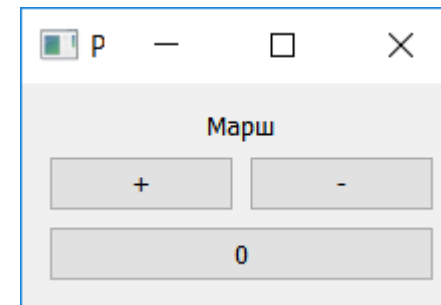
```
< > widget.cpp* Widget::Widget(QWidget *) # Line: 27, Col: 2
1  #include "widget.h"
2  namespace PULT {
3
4  Widget::Widget(QWidget *parent)
5  : QWidget(parent) {
6      setMarshValue=0; //начальное значение заданной маршевой скорости
7      //выделим память под объекты меток и кнопок (первый параметр – текст, который будет
8      //отображаться на кнопке или метке, второй – объект родитель для данного виджета)
9      lblMarsh = new QLabel (tr("Марш"));
10     btnMarshPlus = new QPushButton (tr("+"));
11     btnMarshMinus = new QPushButton(tr ("-"));
12     btnMarshReset = new QPushButton(QString::number(setMarshValue));
13     // Возможно самые внимательные из вас заметили, что теперь при создании
14     // элементов, мы не передаем в конструктор указатель на объект предок,
15     // а это значит, что некому будет позаботиться об освобождении памяти,
16     // выделенной для этих виджетов. Так что же все-таки происходит?
17     // Неужели программа и вправду содержит ошибку, которая
18     // может привести к утечке памяти (memory leak)?
19     // Ошибки в такой записи нет, так как далее мы будем размещать элементы с
20     // помощью менеджеров компоновки, которые в Qt отвечают не только за
21     // правильное размещение виджетов, но и присвоение им нужных виджетов-предков.
22     // Т.е. не нужно беспокоиться о том, чтобы присваивать объекты предков
23     // т.к. это будет сделано автоматически.
24
```

Компоновка элементов в Qt. Вложенные размещения.

widget.cpp (часть 2)

```
24  
25     vMarshLayout = new QVBoxLayout;  
26     hMarshLayout = new QHBoxLayout;  
27     vMarshLayout->addWidget(lblMarsh,0,Qt::AlignHCenter);  
28     //метод addWidget() можно вызвать с указанием stretch-фактора  
29     //и выравнивания (в нашем случае Qt::AlignHCenter - т.е. по центру)  
30     hMarshLayout->addWidget(btnMarshPlus);  
31     hMarshLayout->addWidget(btnMarshMinus);  
32     vMarshLayout->addLayout(hMarshLayout);  
33     vMarshLayout->addWidget(btnMarshReset);  
34     //устанавливаем на форму созданную компоновку  
35     this->setLayout(vMarshLayout);  
36 }  
37  
38 Widget::~Widget() {  
39  
40 }  
41  
42 } //namespace PULT  
43  
44
```

после сборки:



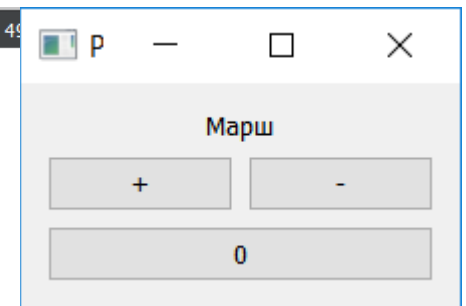
Компоновка элементов в Qt. QGridLayout.

QGridLayout – располагает виджеты в таблице. Таблица состоит из ячеек, позиции которых задаются строками и столбцами.

```
#include <QGridLayout>
```

В widget.h в объявлении класса создадим указатель QGridLayout *gMarshLayout;
В widget.cpp:

```
37 gMarshLayout = new QGridLayout;
38 // Добавить виджет в таблицу можно с помощью метода addWidget(), передав ему позицию
39 // ячейки, в которую помещается виджет. Иногда необходимо, чтобы виджет занимал сразу
40 // несколько позиций, чего можно достичь тем же методом addWidget(), указав в дополни-
41 // тельных параметрах количество строк и столбцов, которые будет занимать виджет (rowSpan, ColumnSpan).
42 //void QGridLayout::addWidget(QWidget *widget, int fromRow, int fromColumn, int rowSpan, int columnSpan,
43 //Qt::Alignment alignment = Qt::Alignment())
44 // В последнем параметре можно задать выравнивание (Qt::AlignCenter), например, по центру:
45 gMarshLayout->addWidget(lblMarsh,0,0,1,2, Qt::AlignCenter);
46 gMarshLayout->addWidget(btnMarshPlus,1,0);
47 gMarshLayout->addWidget(btnMarshMinus,1,1);
48 gMarshLayout->addWidget(btnMarshReset, 2,0,1,2);
49 this->setLayout(gMarshLayout);
50
51
52 }
```



Компоновка элементов в Qt. QFormLayout.

QFormLayout - вспомогательный класс компоновки, который размещает свои дочерние элементы в двух столбцах. Левый столбец содержит метки, а правый столбец содержит виджеты - поля ввода (однострочные редакторы (QLineEdit), счётчики и т.д.

Особенности использования:

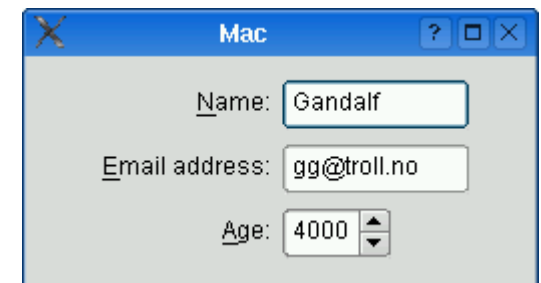
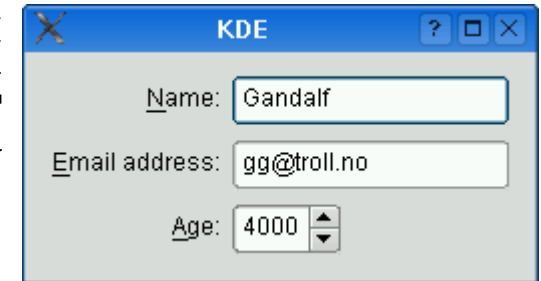
```
#include <QFormLayout>
```

Создание компоновщика с заданным родительским объектом (контейнером) *parent:

```
QFormLayout::QFormLayout ( QWidget * parent = 0 )
```

Добавление пары метка-поле с помощью перегруженной функции [addRow\(\)](#), которая принимает [QString](#) и [QWidget](#) *, самостоятельно создаёт [QLabel](#) и автоматически устанавливает её партнёра.

```
void QFormLayout::addRow ( QWidget * label, QWidget * field )
```



Управление размерами в Qt приложениях

Политики размера задают вызовом метода `setSizePolicy()` класса `QWidget`. Данный метод принимает значения для горизонтальной и вертикальной политики размера, определенные в классе `QSizePolicy`.

Значения	Описание
<code>QSizePolicy::Fixed</code>	Размеры элемента фиксированы и их нельзя изменить.
<code>QSizePolicy::Minimum</code>	Определяет минимально возможные размеры (растягивать элемент можно, а сжимать меньше определенного значения нельзя).
<code>QSizePolicy::Maximum</code>	Определяет максимально возможные размеры элемента.
<code>QSizePolicy::Preferred</code>	Определяет рекомендованные размеры для элемента (т.е. в случае необходимости элемент можно растягивать или сжимать).
<code>QSizePolicy::Expanding</code>	Политика аналогична <code>Preferred</code> , но с тенденцией к увеличению размера.
<code>QSizePolicy::MinimumExpanding</code>	Аналогично <code>Minimum</code> , но с тенденцией к увеличению размера.
<code>QSizePolicy::Ignored</code>	Элемент должен занимать столько места на форме, сколько возможно.

Управление размерами в Qt приложениях

Зададим размерную политику для виджетов панели управления маршевым движением ПА. Для этого необходимо методу `setSizePolicy()` виджета, для которого устанавливается размерная политика передать значения флагов `QSizePolicy::*` для горизонтального, а затем для вертикального направлений.

```
void setSizePolicy(QSizePolicy::Policy horizontal, QSizePolicy::Policy vertical)
```

В .cpp файле необходимо дописать (размерную политику выбирайте сами):

```
lblMarsh->setSizePolicy(QSizePolicy::Maximum, QSizePolicy::Maximum);  
btnMarshMinus->setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Minimum);  
btnMarshPlus->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);  
btnMarshReset->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
```

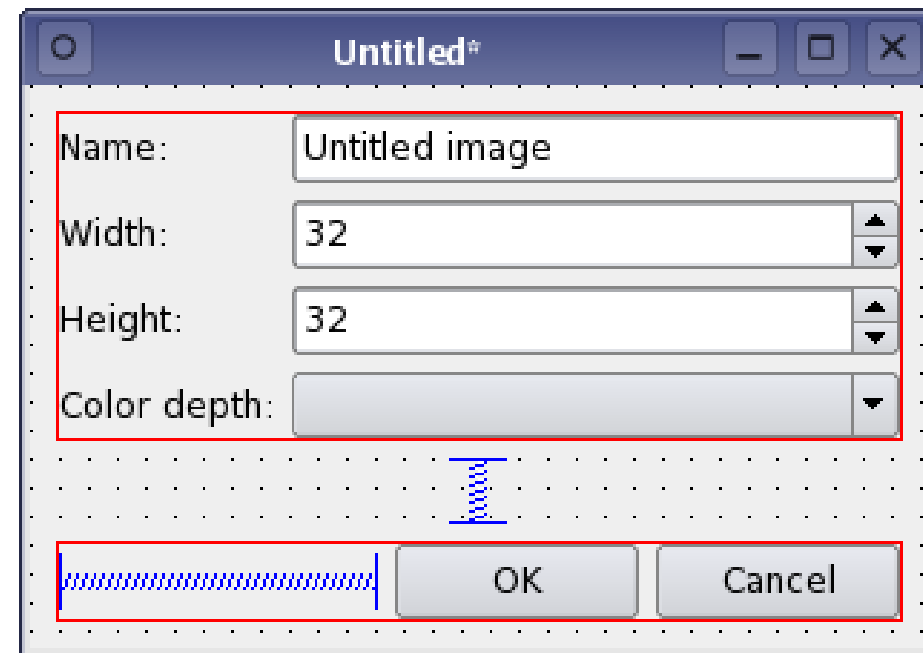
Управление размещением в Qt приложениях

QSpacerItem или фактор растяжения

Представляет собой пустое пространство в компоновке.

Как правило нет необходимости создавать объект данного класса, для этого в классах компоновки существуют специальные методы:

Класс компоновки	Методы добавления факторов растяжения в компоновку
QHBoxLayout QVBoxLayout	addSpacing(), addStretch(), insertSpacing(), insertStretch()
QGridLayout	setRowMinimumHeight(), setRowStretch(), setColumnMinimumWidth(), setColumnStretch()



Практическая часть

```
Если в widget.cpp файле дописать  
vMarshLayout->addWidget(lblMarsh);  
hMarshLayout->addWidget(btnMarshPlus);  
hMarshLayout->addWidget(btnMarshMinus);  
vMarshLayout->addLayout(hMarshLayout);  
vMarshLayout->addStretch(); //добавляет Spacer  
vMarshLayout->addWidget(btnMarshReset);
```

При растяжении формы Spacer будет занимать появившееся пространство (так как это пустое пространство имеет политику размера Expanding), а кнопки будут сохранять свои размеры и занимать место сверху и внизу формы.

