

# СЕМИНАР 9

---

Классы Qt для работы со временем

*Пожирает всё кругом:  
Зверя, птицу, лес и дом.  
Сталь сгрызёт, железо сгложет,  
Крепкий камень уничтожит,  
Власть его всего сильнее,  
Даже власти королей.  
(с) Хоббит или туда и обратно. Дж.Р.Р. Толкин*

# Особенности использования таймера в ПО ПРТС

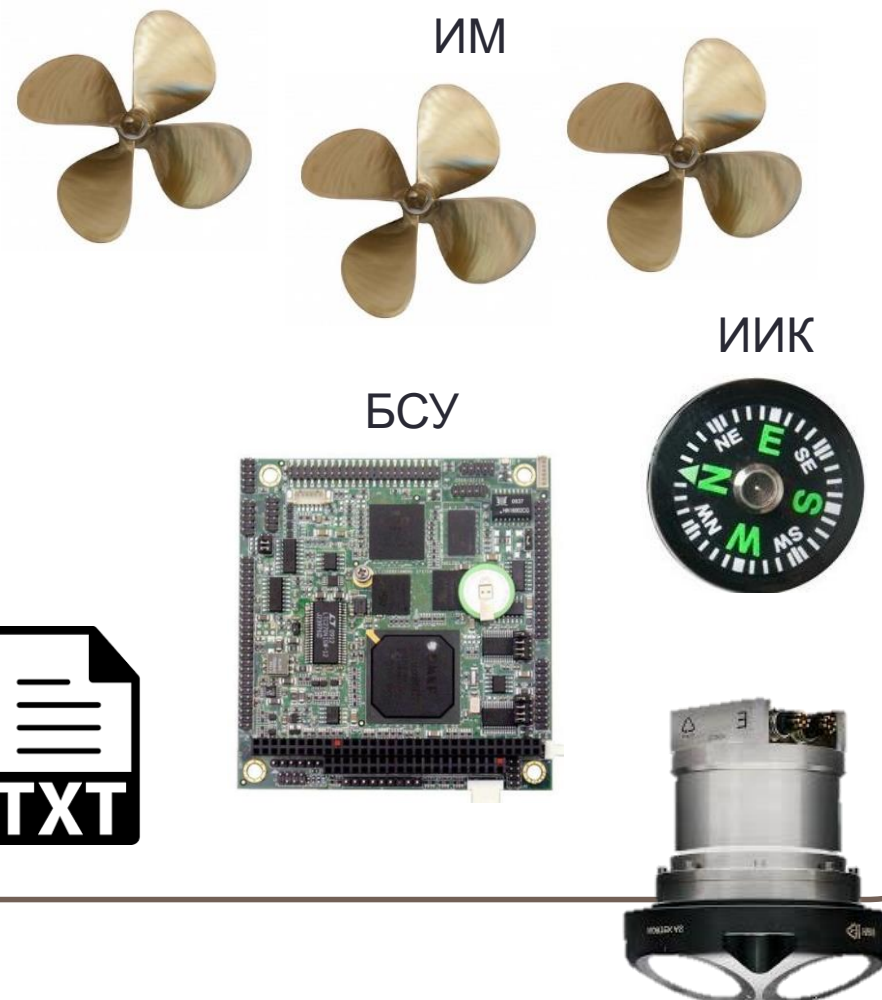
## Пульт управления



Eth II RS-485



## ТНПА



# Работа со временем в Qt

Классы для работы со временем:

1. QDate
2. QTime
3. QDateTime

Таймеры:

1. QObject
2. QTimer

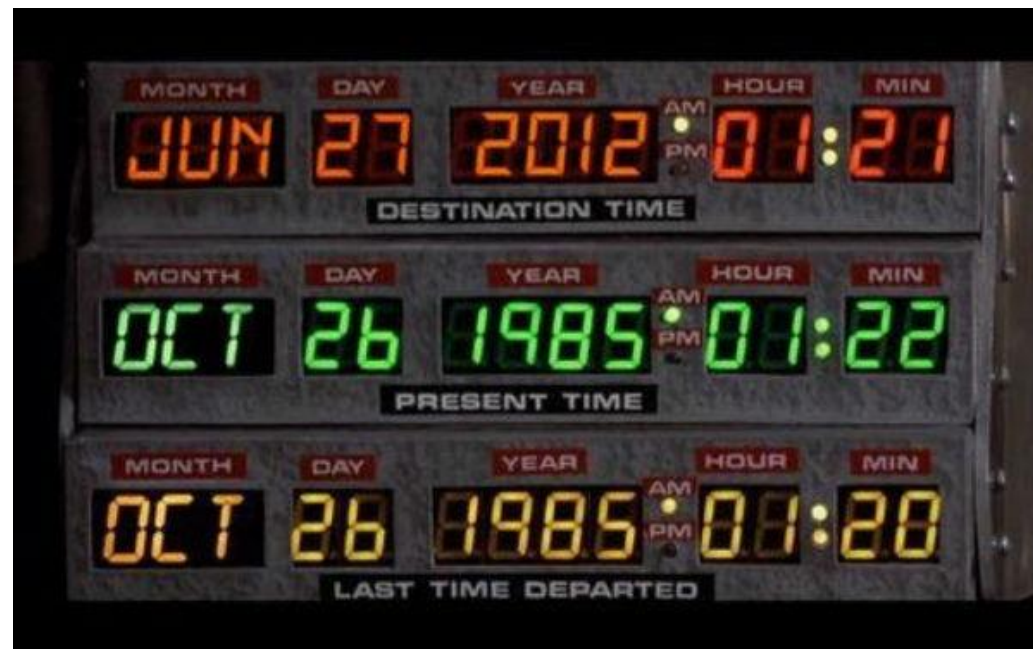


# QDate

**QDate** - структура для хранения дат и проведения с ними разного рода операций.

Особенности:

- Может оперировать днями недели;
- Поддерживает конвертацию между разными календарями (юлианским, григорианским и т. п.);
- Поддерживает преобразование даты из текста (QString) и в текст;
- Позволяет проводить арифметические действия с датами, проводить операции сравнения `==`, `!=`, `<`, `>`, `<=`, `>=`



# QDate.Полезные методы

| Метод                                                 | Описание                                                                                                      |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| setDate (int year, int month, int day)                | Позволяет установить дату                                                                                     |
| currentDate()                                         | Статический метод, который возвращает текущую дату                                                            |
| year(), month(), day(), number of Week(), dayOfWeek() | Возвращает хранящиеся в объекте значения года, месяца, дня, соответствующего номера и дня недели              |
| toString()                                            | Преобразует дату в строковый формат для вывода в соответствии с выбранным Qt::DateFormat или заданным вручную |
| fromString()                                          | Статический метод, который позволяет проводить обратное преобразование из строкового типа к типу QDate        |
| addDays(), addMonths(), addYears()                    | Позволяют проводить арифметические операции с датами                                                          |
| daysTo()                                              | Позволяет узнать разницу в днях между двумя датами                                                            |

# QTime

**QTime – класс для работы со временем. Способен хранить время с точностью до миллисекунд.**

Особенности:

- Поддерживает арифметические операции для объектов времени;
- Поддерживает преобразование времени из текста (QString) и в текст;
- Может работать как таймер;
- Ограничен 24 – часовым интервалом, по истечении которого отсчет начинается с нуля. Для решения этой проблемы можно воспользоваться классом QDateTime.

# QTime. Полезные методы.

| Метод                                                      | Описание                                                                                                                                               |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| setHMS (int hours, int minutes, int seconds, int mseconds) | Позволяет установить время                                                                                                                             |
| currentTime()                                              | Статический метод, который возвращает текущее время                                                                                                    |
| hour(), minute(), second(), msec()                         | Возвращает хранящиеся в объекте значения часов, минут, секунд, миллисекунд                                                                             |
| toString()                                                 | Преобразует время в строковый формат для вывода в соответствии с выбранным Qt::DateFormat или заданным вручную, например time.toString("hh:mm:ss:zzz") |
| fromString()                                               | Статический метод, который позволяет проводить обратное преобразование из строкового типа к типу QTime                                                 |
| addSecs(), addMSecs()                                      | Позволяют проводить арифметические операции со временем                                                                                                |
| secsTo() msecsTo()                                         | Позволяет узнать разницу во времени между двумя временными метками                                                                                     |
| start(), elapsed()                                         | Позволяет начать отсчет времени и узнать сколько времени прошло с начала отсчета (в миллисекундах)                                                     |



# QDateTime

**QDateTime** – класс для работы с датой и временем. Способен хранить время с точностью до миллисекунд.

## Полезные методы

| Метод      | Описание                                   |
|------------|--------------------------------------------|
| date()     | Вызов этого метода возвращает объект QDate |
| time()     | Вызов этого метода возвращает объект QTime |
| toString   |                                            |
| fromString |                                            |
| и т.п.     |                                            |



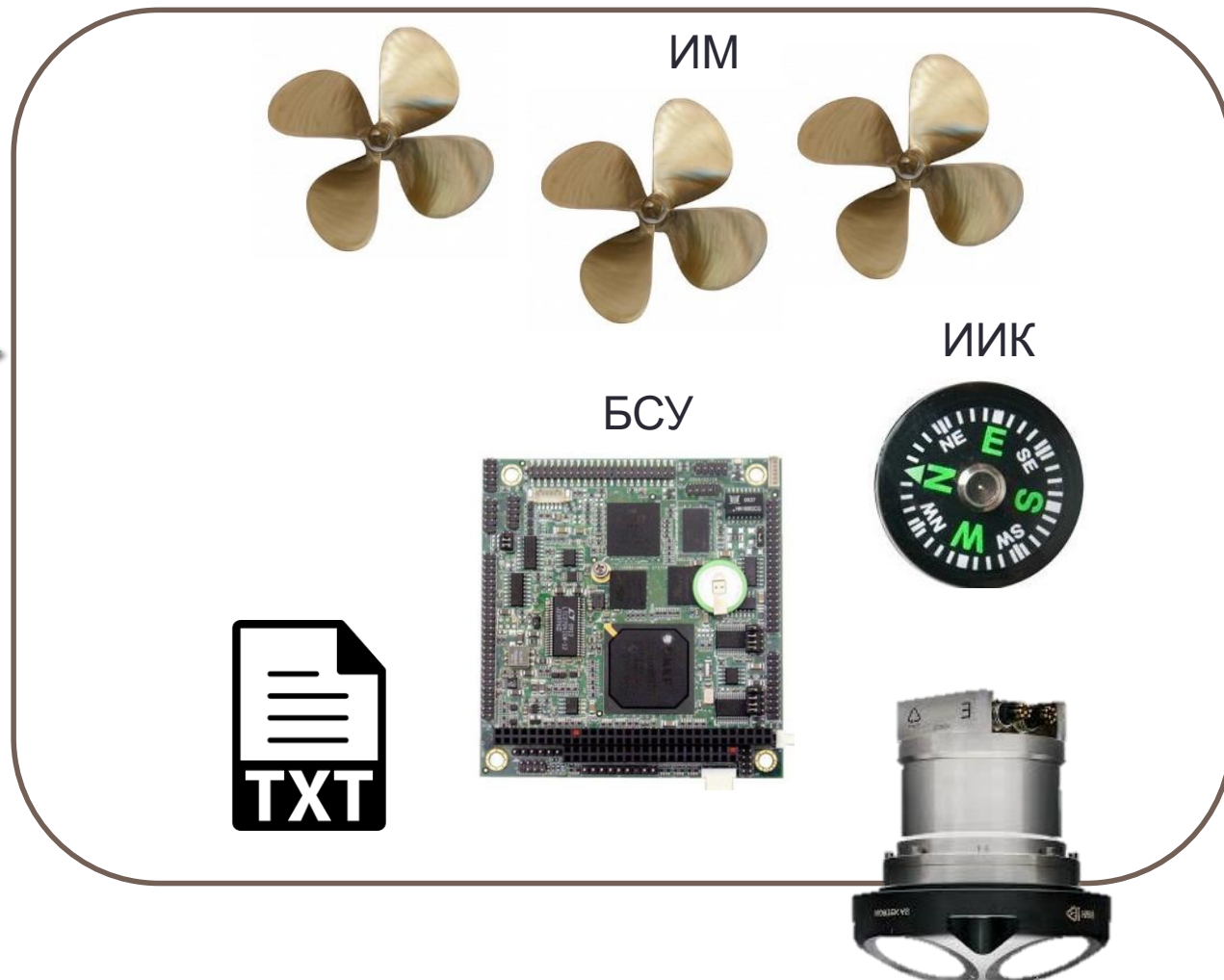
# Особенности работы со временем в ПО ПРТС

Пульт управления

ТНПА



Eth II RS-485



# Особенности работы со временем в ПО ПРТС

Примеры использования таймера:

1. Асинхронная обработка событий (обмен по RS-232/422/485)
2. Запуск алгоритмов обработки, СУ с определенной частотой
3. Запись в файл с определенной частотой (логирование данных)
4. Отправка данных по сети или последовательному интерфейсу
5. Отсчет времени (с момента запуска программы, включения ТНПА и т.п.)
6. И др.

# Таймеры. Qt

Таймер уведомляет программу об истечении заданного промежутка времени (интервала запуска).

Особенности:

- События таймера происходят асинхронно и не прерывают обработку других событий, выполняемых в том же потоке.
- Таймер переходит в сигнальное состояние по истечении интервала запуска, который указывается в миллисекундах.
- Максимальный интервал запуска - 24 дня.
- Можно использовать в многопоточном программировании.
- При этом если программа перегружена интенсивными вычислениями, то точность работы таймера низкая.

# Таймеры. Qt

Классы для работы с таймерами:

1. QObject;
2. QTimer;
3. QTimer.

# Таймеры. QObject.

Алгоритм работы таймера:



Каждый класс, унаследованный от QObject, содержит свои встроенные таймеры.

Особенности:

1. необходимость наследования класса от QObject, чтобы использовать таймер;
2. в унаследованном классе необходимо заместить виртуальную функцию `timerEvent(TimerEvent e);`
3. в функции `timerEvent()` необходимо различать таймеры по ID.

# Таймеры. QObject.

- Для запуска используется метод `startTimer(dt)`;
- `dt` – интервал запуска в мс;
- `startTimer` – возвращает ID таймера;
- В коде:
- `ID=startTimer(d)`;

Запуск таймера

Заместить виртуальный метод

- `void timerEvent (QTimerEvent *e);`

- `QTimerEvent::timerId ()` – возвращает идентификатор таймера, инициировавшего событие.
- `If (e->timerId==ID) { // наш код}`
- `else if (e->timerId==ID2) {//наш код по таймеру 2}`
- `else QWidget::timerEvent(e);`

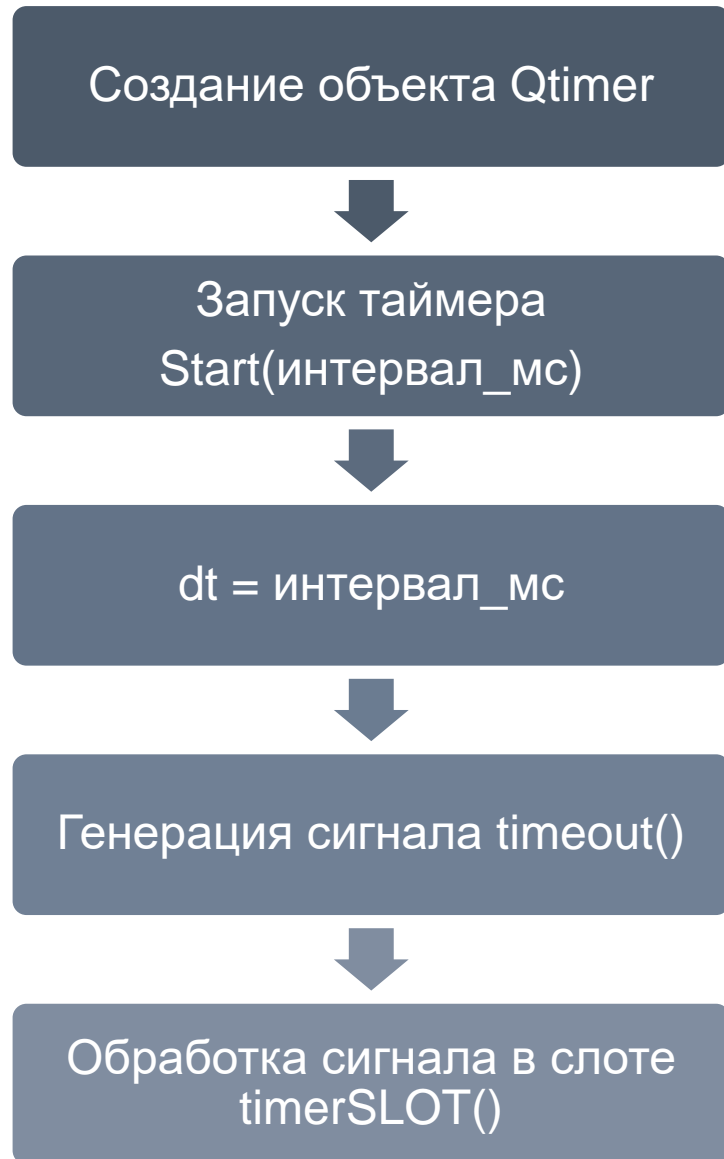
Проверка таймера по ID

# Таймеры. QObject. Полезные методы

| Метод              | Описание                                                                                                     |
|--------------------|--------------------------------------------------------------------------------------------------------------|
| startTimer()       | Вызов этого метода запускает таймер и возвращает его ID, необходимый для распознавания в методе timerEvent() |
| killTimer(timerID) | Уничтожает таймер, идентификатор которого передан в метод                                                    |



# Таймеры. QTimer.



Иерархия классов:

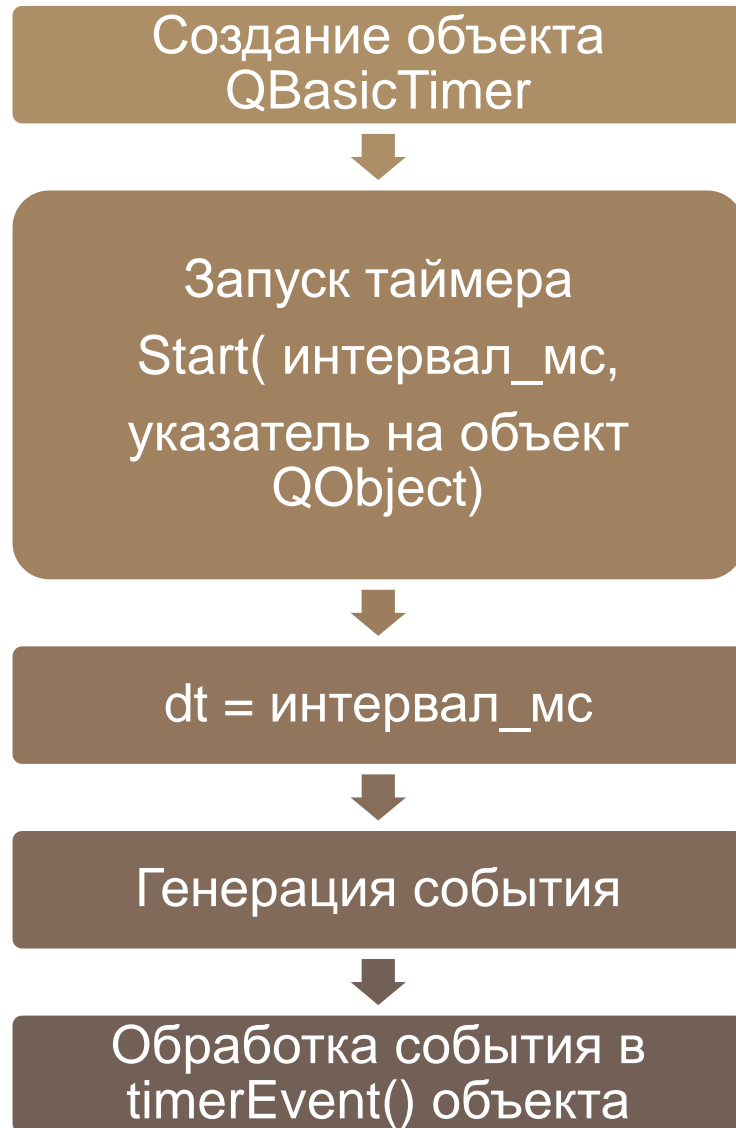


- По истечении интервала запуска отправляет сигнал timeout();
- Содержит статический метод singleShot() однократного срабатывания таймера, который позволяет не создавать объект таймера.

# Таймеры. QTimer. Полезные методы

| Метод                                      | Описание                                                                                                                                                                                                          |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>start()</code> , <code>stop()</code> | Вызов метода <code>start()</code> запускает таймер с интервалом запуска, указанным в миллисекундах, <code>stop()</code> – останавливает работу таймера                                                            |
| <code>singleShot()</code>                  | Статический метод для одноразового запуска таймера. В качестве параметров этой функции передаются интервал запуска, указатель на объект и слот, с которыми нужно соединить сигнал <code>timeout()</code> таймера. |
| <code>isActive()</code>                    | Проверка, находится ли таймер в активном состоянии                                                                                                                                                                |

# Таймеры. QTimer.



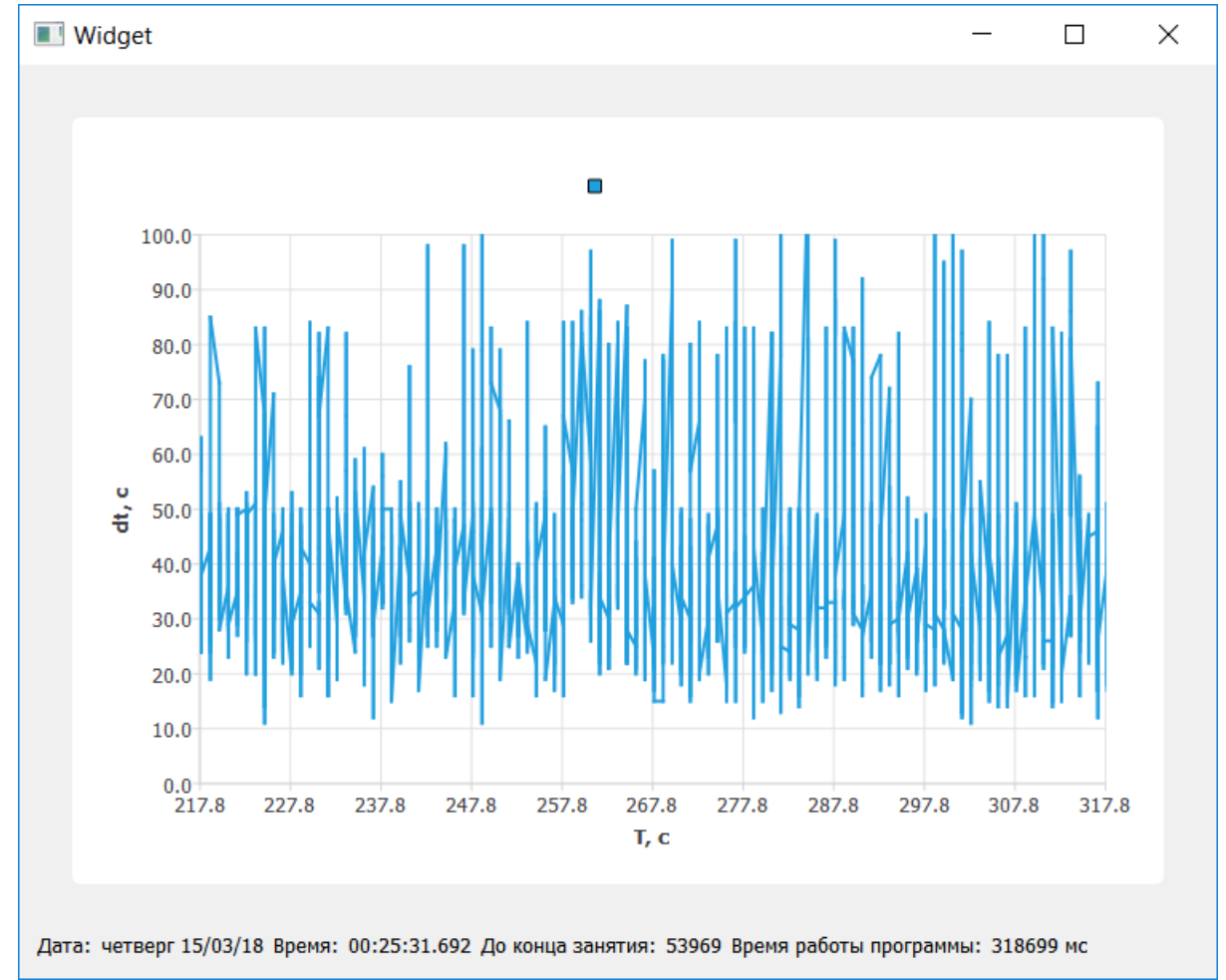
- Минималистичное решение для таймера.
- Предоставляет всего 4 метода:
- isActive()
- start()
- stop()
- timerId()
- Обработка событий таймера происходит в методе timerEvent() объекта, указатель на который был передан при запуске start() таймера.

# Практическая часть

Дополнить проект TimeProject таким образом, чтобы:

1. отображались текущая дата,
2. текущее время,
3. время до конца занятия,
4. время работы программы с момента запуска,
5. выводился график точности работы таймера QTimer. (за эталонное можно принять системное время или время, полученное с помощью QTime для оценки продолжительности работы программы).

\*Обновление данных о текущем времени реализовать с помощью таймера QObject.



# Практическая часть

widget.h

```
> widget.h tick(): void
1  #ifndef WIDGET_H
2  #define WIDGET_H
3
4  #include <QWidget>
5  #include "ui_widget.h"
6  #include <QDate>
7
8  class Widget : public QWidget, Ui::Widget {
9      ... Q_OBJECT
10 public:
11     ... explicit Widget(QWidget *parent = 0);
12     ... ~Widget();
13 private:
14     ... QDate *date; // сегодняшняя дата
15     ... QTime *time; // текущее время
16     ... QTime *operatingTime; // время работы приложения
17     ... QTimer *timer; // таймер, корректность которого будем проверять
18     ... double prevOperatingTime; // переменная, в которой будем хранить предыдущее время
19     ... // попадания в функцию обработки сигнала таймера
20
21     ... // мы рассмотрим работу таймера QTimer и таймера QObject
22     ... // для реализации таймера QObject нам понадобится переменная для
23     ... // сохранения id таймера:
24     ... int timerWidgetID;
25     ... // Так как таймер объекта QObject создает событие, а не сигнал, \
26     ... // то для его обработки необходимо заместить виртуальную
27     ... // функцию timerEvent(QTimerEvent *e) и прописать там наш код
28     ... // который должен выполняться при поступлении событий таймера
29     ... void timerEvent(QTimerEvent *e);
30
31     ... // таймер QTimer создает сигнал timeout(), который мы соединим со своим
32     ... // слотом, назовем его tick():
33 public slots:
34     ... void tick();
35 };
36 #endif // WIDGET_H
```

# Практическая часть

widget.cpp

```
editor
Инструменты Окно Справка
widget.cpp Widget::~Widget()
1 #include "widget.h"
2
3
4 Widget::Widget(QWidget *parent) :
5     QWidget(parent) {
6     setupUi(this);
7     //Выделим память под объект QTime, в котором будем хранить время
8     //прошедшее с момента запуска программы
9     operatingTime = new QTime();
10    //при помощи метода start() класса QTime можно начать отсчет времени
11    operatingTime->start();
12    //для того, чтобы затем узнать сколько времени прошло с момента
13    //начала отсчета, следует вызвать метод elapsed()
14    prevOperatingTime = operatingTime->elapsed();
15
16    //создадим объекты для хранения текущей даты
17    date = new QDate();
18    //и времени
19    time = new QTime();
20    //каждый класс, унаследованный от QObject, содержит свои встроенные таймеры
21    //вызов метода startTimer() -- запускает таймер и возвращает идентификатор таймера
22    //тик такого таймера -- это событие, а не сигнал и обрабатывается оно не в слоте,
23    //который вы с ним соедините, а в функции timerEvent класса. События ВСЕХ таймеров
24    //объекта, созданных таким образом будут обрабатываться в функции timerEvent()
25    //чтобы разобраться от какого таймера пришло событие необходимо знать его
26    //идентификатор -- id таймера. Сохраним его в переменной timerWidgetID.
27    //Кроме того, при этом методу передается величина интервала запуска таймера в миллисекундах:
28    timerWidgetID = this->startTimer(100); //интервал 100мс
29
30
```

# Практическая часть

widget.cpp

```
< > widget.cpp Widget::tick(): void
29
30
31 }
32
33 ~Widget() {
34
35 }
36
37 void Widget::timerEvent(QTimerEvent *e)
38 {
39     //проверяем ID таймера, если совпадает с тем, который создали, то
40     //обрабатываем событие сами
41     if(e->timerId()==timerWidgetID)
42     {
43
44     }
45     //если id таймера не совпадает с timerWidgetID, то отправляем событие в
46     //стандартную функцию нашего базового класса QWidget
47     else QWidget::timerEvent(e);
48 }
49
50
```



# Практическая часть

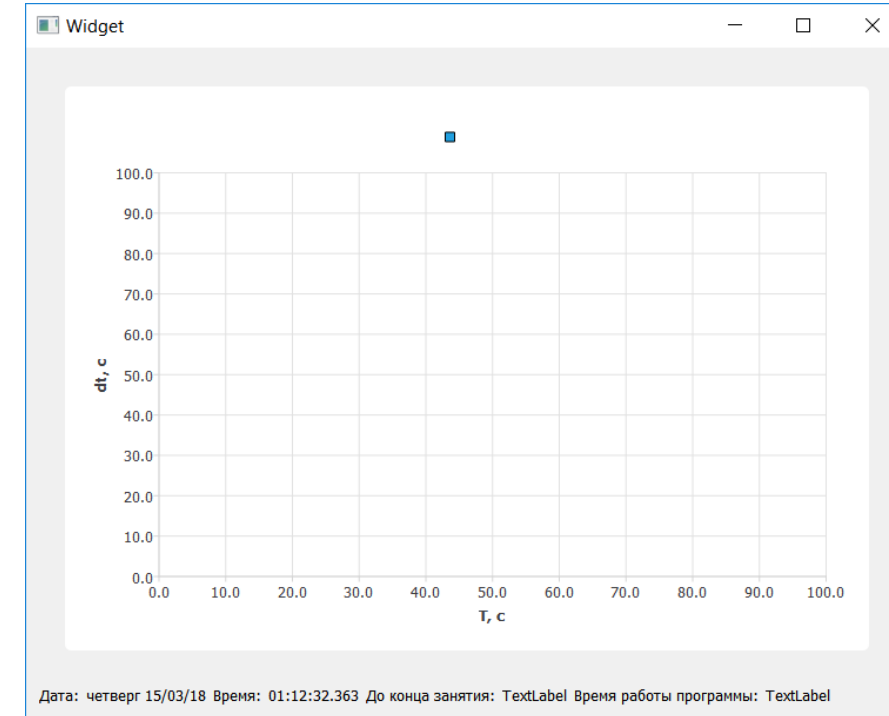
widget.cpp

```
widget.cpp*  Widget::timerEvent(QTimerEvent *) void

void Widget::timerEvent(QTimerEvent *e)
{
    //проверяем ID таймера, если совпадает с тем, который создали, то
    //обрабатываем событие сами
    if(e->timerId()==timerWidgetID){
        //1. Выведем текущую дату.
        //чтобы получить значение текущей даты можно вызвать статический метод
        //currentDate(), возвращающий объект класса QDate.
        *date = QDate::currentDate();
        //выведем значение текущей даты в метке lblData
        //чтобы преобразовать значение даты к типу QString используем метод toString()
        //которому можно передавать формат отображаемой даты, где d,M,y - переменные
        //условно обозначающие день, месяц, год. Количество переменных, т.е. dd, ddd и т.п
        //обозначает степень "подробности" выводимой информации
        lblData->setText(date->toString("dddd-dd/MM/yy"));

        //2. Выведем текущее время
        //для получения текущего времени в классе QTime имеется статический метод
        //currentTime()
        *time = QTime::currentTime();
        //QTime предоставляет метод toString() для передачи данных объекта времени
        //в виде строки. В этот метод можно передать формат Qt::DateFormat или
        //задать свой собственный формат по аналогии с отображением даты:
        lblTime->setText(time->toString("hh:mm:ss.zzz"));
    }
    //если id таймера не совпадает с timerWidgetID, то отправляем событие в
    //стандартную функцию нашего базового класса QWidget
    else QWidget::timerEvent(e);
}
```

Насладимся промежуточным  
результатом:



# Практическая часть

widget.cpp

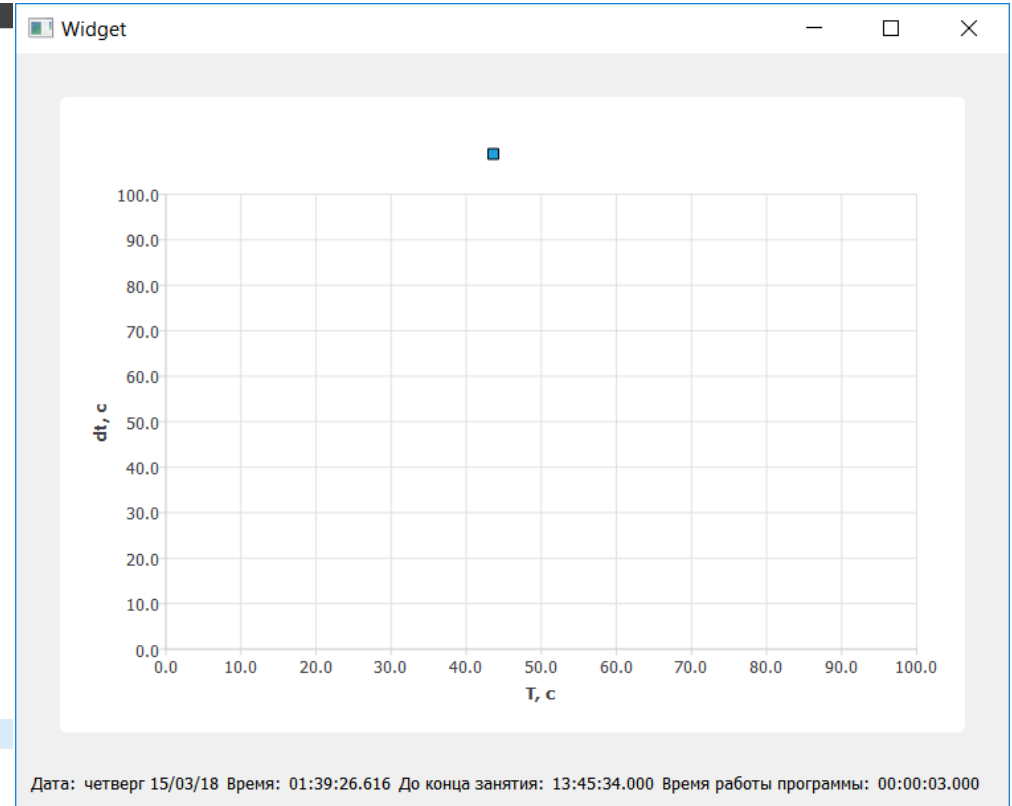
```
.....//в виде строки. В этот метод можно передать формат Qt::DateFormat или
.....//задать свой собственный формат по аналогии с отображением даты:
.....lblTime->setText(time->toString("hh:mm:ss.zzz"));

.....//3. Выведем время, оставшееся до конца занятия
.....//создавая объект QTime, в конструктор можно передать 4 числа
.....//они будут представлять собой часы, минуты, секунды и миллисекунды
.....//соответственно... окончание занятия у нас в 15:25, так что передадим
.....//в конструктор объекту timeToStop числа 15, 25, 0, 0:
.....QTime timeToStop(15,25,0,0);
.....//чтобы вычислить время, оставшееся до какого-либо события можно
.....//использовать метод secsTo(), в качестве параметра этому методу
.....//необходимо передать объект QTime, в котором хранится время события
.....//при этом метод возвращает количество секунд до события
.....//в переменной типа int
.....int t = QTime::currentTime().secsTo(timeToStop);
.....//данную переменную затем будет необходимо конвертировать в "читаемый"
.....//формат:
.....lblEnd->setText(QDateTime::fromTime_t(t).toUTC().toString("hh:mm:ss.zzz"));

.....//4. Выведем время работы приложения
.....lblTimer->setText(QDateTime::fromTime_t\
.....(operatingTime->elapsed()/1000).toUTC().toString("hh:mm:ss.zzz"));

.....}
.....//если id таймера не совпадает с timerWidgetID, то отправляем событие в
.....//стандартную функцию нашего базового класса QWidget
.....else QWidget::timerEvent(e);
.....}
```

Насладимся промежуточным результатом:



# Практическая часть

widget.cpp

В конструкторе класса Widget создадим таймер:

```
widget.cpp*  Widget::Widget(QWidget *)
... //и времени
... time = new QTimer();
... //каждый класс, унаследованный от QObject, содержит свои встроенные таймеры
... //вызов метода startTimer() -- запускает таймер и возвращает идентификатор таймера
... //тик такого таймера -- это событие, а не сигнал и обрабатывается оно не в слоте,
... //который вы с ним соедините, а в функции timerEvent класса. События ВСЕХ таймеров
... //объекта, созданных таким образом будут обрабатываться в функции timerEvent()
... //чтобы разобраться от какого таймера пришло событие необходимо знать его
... //идентификатор -- id -- таймера. Сохраним его в переменной timerWidgetID.
... //Кроме того, при этом методу передается величина интервала запуска таймера в миллисекундах:
... timerWidgetID = this->startTimer(100); //интервал 100мс
...
... //выделим память под объект таймера QTimer
... timer = new QTimer();
... //запускаем таймер методом start(), но в отличие от QTimer передаем
... //величину интервала запуска в миллисекундах:
... timer->start(50);
... //у таймера есть сигнал timeout(), который он генерирует по истечении интервала
... //запуска, соединим его с нашим слотом tick()
... connect(timer, SIGNAL(timeout()), SLOT(tick()));
```

# Практическая часть

widget.cpp

Опишем функцию tick() :

```
96
97
98 ✓ void Widget::tick(){
99
100     ....//в классе графика chartForm есть метод setYT, который выводит
101     ....//переданные значения Y и текущего времени на график, передадим в качестве Y
102     ....//для вывода разницу между текущим временем работы приложения
103     ....//operatingTime->elapsed() (в миллисекундах) и прошлым, сохраненным значением
104     ....//prevOperatingTime, а в качестве T текущее время operatingTime->elapsed()
105     ....chartForm->setYT(operatingTime->elapsed()-prevOperatingTime, operatingTime->elapsed());
106     ....//сохраним текущее время работы приложения для последующего сравнения
107     ....prevOperatingTime = operatingTime->elapsed();
108 }
109
110
111
112
```

# Практическая часть. Итог

