# Basic For Qt® essentials

This document gives you some very important information about Qt® and Basic For Qt®. It comes with short paragraphs for each topic to help you to memorize it and not to overkill you with too much information.

## Contents

## Basic For Qt®

Basic For Qt® is the new language to write Qt based cross-platform software. If you have at least moderate knowledge in simple object-oriented concepts and the BASIC language, Basic For Qt® will not be difficult for you and you will write your own applications soon.

## Key Features

- Compile to C++ binary code with thin runtime
- Common BASIC-like functions are built-in
- Clean and easy syntax
- Qt Designer (use of ui files) supported
- Resource support
- Copy directory for application data files
- Use of native C++ classes and widgets
- Create custom widgets in Basic For Qt®, also usable with Qt Designer files (ui files)
- Direct Qt API access
- Qt classes maybe overriden and extended

## Compile to C++ code with thin runtime

Create modern Qt applications with a BASIC language and a garbage collector.

## Debugging your application within Qt Creator (GCC) or VS

Whenever your project gets compiled, a proper C++ project is created which can be opened in Qt Creator (GCC mode) or in VS (on Windows).

- For Qt Creator you need to open the ***.pro** file inside the build directory of your project directory.
- For VS you need to open the ***.vcproj** file inside the build directory of your project directory.

It is important to set the **Windows PATH** environment variable to the Qt DLL files or otherwise your application won't start.

- For Qt Creator it is the **mingw** directory within the Basic For Qt® installation directory
- For VS it is the **vs** directory within the Basic For Qt® installation directory

Another option is to copy all Qt DLL files in the **gcc/build/debug** directory in your project directory.

On non-Windows computers VS is not an option, of course.

## Common BASIC-like functions are built-in

Left, Mid, MsgBox, InStr...

A widget means a control in Qt terms.

## Clean and easy syntax

### *Variable*

```
Dim name As type [= expression]
Public Dim name As type [= expression]
Private Dim name As type [= expression]
```

### *Constant*

```
Const name [As type] = expression
Public Const name [As type] = expression
Private Const name [As type] = expression
```

### *Function Procedure- With returning a value*

```
Function name(arguments) As type
Public Function name(arguments) As type
Private Function name(arguments) As type
Return expression
End Function
```

### *Sub Procedure - Without return a value*

```
Sub name(arguments)
Public Sub name(arguments)
Private Sub name(arguments)
Return
End Sub
```

### *Event Procedure – Overriden super class*

```
Event name(arguments)
Return
End Event
```

### *Outlet – widget in qt designer file (ui files)*

```
Outlet name As type
Public Outlet name As type
```

```
Private Outlet name As type
```

**name = objectname:** in qt designer

### *Signal Procedure – event from widget in qt designer file (ui files)*

```
Signal name(arguments)
Return
End Signal
```

Follows the form "on**_objectname_eventname"**(arguments), e.g.

```
Signal on_pushButton_clicked(Checked As Boolean)
...
End Signal
```

**objectname:** in qt designer set

**eventname**: several widget events available

### *Slot Procedure*

Reserved for future releases.

### *Arguments*

Keep it empty (no arguments) or

```
name as type
name as type, name as type, ...
```

Arguments are passed by value by now. This may change in future versions. If you pass an array as argument and change its contents in the procedure, this changes are done in the originial array.

If a built-in function or subs comes without arguments you don't need ot use ( ) to call it.

# Using native C++ classes with declare statements

## Either used as super class in the IDE or used in the Qt Designer for widgets

- in the directory "/**cpp**"
- copy all needed C++ source files there
- no sub-directories are allowed yet (if you directly place C++ source files without a pro file) and no binary versions of C++ files yet
- all files there will be automatically compiled with the project
- You may also use sub-project with sub-directories. Therefore you need to create a pro file (NAME.pro) copy it to the /cpp directory and a sub directory (NAME) which is exactly named like the pro file (you need to create a header file NAME.h as well including all relavant internal header files of your sub project), e.g.
    - /cpp/NAME.pro
    - /cpp/NAME (the directory)
    - /cpp/NAME/NAME.h (including all relavant internal header files of your sub project)

You have to create this directory **cpp** in your Basic For Qt® app directory either by the by the proper command in the IDE or manually.

# Direct Qt API access

It is possible to gain access to Qt functionality, which is not yet implemented in Basic For Qt® by using declare statements.

# Declare statements

Whenever you use declare statements for a custom class, your C++ class needs to be coded in a *.h file and *.cpp file using the same file name as the class name, e.g.

Declare Class "Q7BCodeView" -> Q7BCodeView.h + Q7BCodeView.cpp
and there must be a C++ class named Q7BCodeView in that file declared

### *C++ data types conversion supported:*

| **C++** | **Basic For Qt®** |
|---|---|
| bool | Boolean |
| long long / int | Integer |
| single / double | Float |

| | |
|---|---|
| QString / const QString & / QString * | String |
| QStringList | Array |
| | |
| QWidget * | QWidget |

Enum values are converted to Integer by default.

## File encoding

All source files are treated as utf8, but all identifiers and names must be named with ascii character only.

# Variables

All variables points to objects and a new variable automatically points to a new object by declaration. This can be switched off by explicitely setting the variable to Null.

## Default values are

Boolean = False (bool)

Integer = 0 (qint64)

Float = 0.0 (qreal)

String = “” (QString)

else an object is assigned to the variable of its type, e.g.

```
Dim var As mytype ' var points to an object of type mytype
```

Type id  (alias for QObject) is always set to Null

## Explicitly set variable to Null

Prevents the automatically creation of an object of any type

```
Dim var as mytype = Null
```

## *Outlets*

An outlet is just a variable and gets its value when the related ui file is loading. It can be any object created in a ui file. So it is possible to set any object as an outlet, but it depends on what you try to accomplish.

## All tools are open source

All Basic For Qt® compiler and runtime tools are all completely open-source.

Basic For Qt® uses the GCC, Qt Creator / Qt SDK (and mingw on Windows). Additional, the compiler, runtime and IDE of Basic For Qt® are open-source as well. Despite this, there is an option to use the Microsoft Visual Studio C++ compiler on Windows.

## Resources

Having data files like texts or images loadable by your application. First option is the use of resources, which are directly compiled in your application binary.

All files and  and files of sub-directories in your project resource directory will be available as resources within your application.

All files in are included in your applications' executable and accessable with file name with the following form:

- ":/Resources/filename"
- ":/Resources/subdir/filename"

e.g.

```
MsgBox(ReadString(":/Resources/Readme.txt"))
```

You have to create this directory **Resources** in your Basic For Qt® app directory either by the by the proper command in the IDE or manually.

## Copy directory

Having data files like texts or images loadable by your application. Second option is the use of files, which are deployed within your application directory.

All files and directories of the project directory "Copy" will be copied to the directory of the executable.

You have to create this directory **Copy** in your Basic For Qt® app directory either by the by the proper command in the IDE or manually.

## Different kind of windows:

- window (normal qwidget)
- qmainwindow (menubar + toolbar) might be sdi or mdi
- dialog (modal)
- dockwidget
- toolwindow (non-modal)
- (sheet and drawser on Mac)

The mainwindow consists of one menubar, one or more toolbars (on Mac, you normally should use only one toolbar, keep that in mind), a statusbar on the bottom and no or several dockwidgets and one main area (in qt terms called centralwidget), which may be used as SDI or MDI.

Windows, toolwindows, dialogs may be used as well.

| | | |
|---|---|---|
| menubar | | |
| Toolbar (custom controls may be inserted) | | |
| dockwidgets | SDI or MDI | dockwidgets |
| statusbar | | |

QAction is used for menubar entries and toolbar entries, in fact there are shared among the two.

You should not dynamically change the entries of toolbars, because it is not common behaviour on Mac. But changing the menu entries at runtime for example for a list of windows is common practise on Mac as well.

## Cross-platform golden rules:

- Use case-sensitive filenames
- (don't use the mainwindow title for important information, because you don't have it on Mac)
- always use / as path separator (not \) on windows there are transformed automatically.

### *Calling Methods*

```
mySub(myVar)

myOtherSub()
```

```
Me.mySub(myVar)
```
if it is declared in the same code file

```
myObjectVar.mySub()
```

If a built-in function or sub comes without arguments you don't need ot use ( ) to call it.

### *Dot Syntax*

It is used to call subs and functions of another object or to access properties of another object.

```
Me.mySub(myVar)
```

```
myObjectVar.itsProperty = 23
```

### *Creating Objects in code*

Dim a As Array

creates an array object and the variable a will refer to the new created array object.

### *Memory Management*

All objects are automatically released and freed.

### *Designing a Class*

One code file contains only one class. The class name is determined by the file name of that code file. The file extensions determines the super class of the class.

### *Logging*

Use MsgBox(...) to show values during developement.

Additional, there are some debug functions: stdout(...) and stderr(...)

## Golden Rules

- confusing elements of C++ were omitted or hidden
- All variables in Basic For Qt® are objects; even the simple data types, like numeric and boolean values.
- Object variables always point to an object even no object has been assigned. But pointing to nothing (Null) can be forced in the declaration line of a variable.
- In Basic For Qt® memory management is automatically managed by a garbage collector.
- Basic For Qt® programs are compiled into C++ code, which makes them very fast.
- You do not need to use "new" for a variable declaration like in C++
- there is no separation of declaration and definition in code

- every class can be used as a module without the need of an object to call functions and subs of it (singletone feature) and without the need of extra coding for the developer
- To extend Basic For Qt® language features through direct call to Qt is possible
- Literals (format of numbers and string escape sequence) are the same as in C++
- Boolean literals are 'True' <> 0 and 'False' = 0
- Constants may be of any type
- Creating sub-classes from custom classes is not possible, but from built-in Qt classes
- The type Array and Dictionary as well String is very important and used in most applications

## Two tools for development

You write your code in the Basic For Qt® IDE (functions, subs and event code) and draw your GUI using Qt Designer and save them as ui files, which can be loaded by your application automatically. The KBasic form designer is planned to be integrated inside the IDE in some future release of Basic For Qt® (KBasic is the ancestor of Basic For Qt®).

## Where does my application start?

All GUI elements created with Qt Designer are stored as objects in a ui file, which will become alive again when they are loaded by your application at start up. This is the first step, before any code of you is executed.

## MainWindow.ui

It get automatically loaded during your application start up. Then the event 'Init' of the file MainWindow.QMainWindow gets called, when it has been declared by you.

## Init

Every class has an event sub called Init, which gets automatically called whenever an object is created from a class.

**The Event Init of Global.QObject is the first place to be called, after that Event Init of MainWindow.QMainWindow is called.**

## Finalize

**Every class has an event sub called Finalize, which gets automatically called whenever an object is going to be destroyed by the runtime, because it has no reference anymore.**

**If you use the singletone feature (usage of a class name instead of a variable name), your automatically created only class object of that specific file will not get finalized when the application quits.**

## SerialPort

**It is reported to be working using. (USB to Serial converter)**

## Team work: Basic For Qt® code and Qt Designer objects

Overview relationship:

| **Basic For Qt® IDE** (Code) | **Qt Designer** (GUI) |
|---|---|
| variables for GUI objects<br>*Outlet OBJECTNAME As type* | GUI objects |
| event procedures for GUI events<br>*Signal on_OBJECTNAME_SIGNALNAME(ARGUMENTS)* | GUI objects |

Which means if you want to use GUI objects you need to declare a variable in code so that you may use the GUI object in code and is defined for the rest of the code. It is the same for event procedures. If you would like to react to GUI signals, you have to define for each wanted signal a signal procedure in code.

## Classes and Objects

All custom classes are declared in the Basic For Qt® IDE. Every class is written in just one code file. The name of the code file determines the name of the class and the super class' name.
How to create objects from classes:

- Do it in code by manually instantiate from a known class. They exists as long no Null value is assigned to the origin variable and no other variable points to that object.

## List and Dictionary

May contain any variable of any type.
May also read and write xml files.
List: If you access beyound the bounds, you got a Null value returned.
Dictionary: If your key does not exists, you got a Null value returned.

## Generic type id and QObject

QObject is the base class of most Qt classes and therefore often used.

id is a variable type, which may contain any variable even non-QObject based classes like Dictionary. Variables declared with id may hold any object.

In order to call object procedures, you may change temporary the type by assigning the value to another variable with the needed type (at compilation).

## Named arguments

Named arguments may be used, therefore it is possible to write every call of subs and functions with named arguments.

Syntax:

```
mySub(namedArgument := 11)
```

## Signal and Event procedures

Signal, Event

These procedures must not be directly called by you in code. Instead there are automatically called.

## Object in use

### *Property call*

```
myobject.myproperty = 23
myvar = myobject.myproperty
```

### *Sub or Function called*

```
myobject.mysub(324)
myvar = myobject.myfunction()
```

You can declare custom properties for custom classes yet with a Property keyword.

Another way is that you can directly use the dynamic properties every QObject provides automatically. Use the operator ! to access them. If your class is not based on QObject an internal QMap of QVariants is used.

The property “myproperty” of the variable “myvar” is set and get with the following code:

```
myvar!myproperty = "hello"
MsgBox(myvar!myproperty)
```

Direct access to Qt's properties of each object by using !

e.g.

```
Outlet mycontrol As QPushButton  ' QAbstractButton is one of its
superclasses

...
```

```
mycontrol!text =  “changed”   ' qabstractbutton.html#text-prop
...
```

Property access: Besides String, Float, Integer, Boolean -> DateTime is supported as well. For Decimal and the other ones convert them to a string and back.

## Qt Designer (use of ui files)

The following list contains all supported Qt designer widgets (unsupported Qt designer widgets cannot be used in code):

Horizontal Layout: QHBoxLayout

Vertical Layout: QVBoxLayout

Grid Layout: QGridLayout

Form Layout: QFormLayout

Horizontal Spacer: Spacer

Vertical Spacer: Spacer

Push Button: QPushButton

Tool Button: QToolButton

Radio Button: QRadioButton

Check Box: QCheckBox

Command Link Button: QCommandLinkButton

Button Box: QDialogButtonBox

List Widget: QListWidget

Tree Widget: QTreeWidget

Table Widget: QTableWidget

Group Box: QGroupBox

Scroll Area: QScrollArea

Tool Box: QToolBox

Tab Widget: QTabWidget

Stacked Widget: QStackedWidget

Frame: QFrame

Widget: QWidget

MdiArea: QMdiArea

Dock Widget: QDockWidget

Combo Box: QComboBox

Font Combo Box: QFontComboBox

Line Edit: QLineEdit

Text Edit: QTextEdit
Plain Text Edit: QPlainTextEdit
Spin Box: QSpinBox
Double Spin Box: QDoubleSpinBox
Time Edit: QTimeEdit
Date Edit: QDateEdit
Date/Time Edit: QDateTimeEdit
Dial: QDial
Horizontal Scroll Bar: QScrollBar
Vertical Scroll Bar: QScrollBar
Horizontal Slider: QSlider
Vertical Slider: QSlider

Label: QLabel
Text Browser: QTextBrowser
Calender: QCalenderWidget
LCD Number: QLCDNumber
Progress Bar: QProgressBar
Horizontal Line: QFrame
Vertical Line: QFrame
QWebView: QWebView

PhononVideoPlayer: Phonon::VideoPlayer
PhononSeekSlider: Phonon::SeekSlider
PhononVolumeSlider: Phonon::VolumeSlider

## Classes which cannot be instantiated

QResizeEvent, QPaintEvent, QPainter, QCloseEvent

## Classes which cannot be sub-classed

PhononVideoPlayer, PhononSeekSlider, PhononVolumeSlider

### *ui files without a code file*

Adding new ui widgets as windows or dialogs to your application without the need of sub-classing (but no outlets and events are supported then). Useful only for static content, just as an about box in your application. Create a new ui file without a code file and use it with Qt designer and save it. In this case ui files are built-in as resources and interpreted at runtime and not included as C++ code.

Just open them with the following command.

```
Open("filename")
```

e.g. About.ui

```
Open("About")
```

## Using custom widgets in qt designer without the need to create a plugin for Qt designer

```
Outlet OBJECTNAME As DATATYPE Set
```

**Set** shows that the compiler should replace the stored data type (placeholder) in your ui

- use qwidget as a placeholder in Qt Designer, or otherwise make sure you use the proper super class
- works with every layout and parent (ui files are changed before compilation)
- your replacement classs directly or indirectly must inheriting QWidget, otherwise it will fail

## Preferences

are stored in the registry on windows
HKEY_CURRENT_USER\Software\"Projectname"\OrganizationDefaults

## Overview about KBasic vs. Basic For Qt® control names

| KBasic | Basic For Qt® |
|---|---|
| Control | QWidget |
| Form | QWidget or QDialog |
| CommandButton | QPushButton |
| CommandLinkButton | QCommandLinkButton |
| ToolButton | QToolButton |
| ImageButton | ??? |

| | |
|---|---|
| Label | QLabel |
| CheckBox | QCheckBox |
| RadioButton | QRadioButton |
| TextBox | QLineEdit |
| Frame | QGroupBox |
| ComboBox | QComboBox |
| ListBox | QListWidget |
| DateBox | QDateBox |
| TimeBox | QTimeBox |
| Sound | ??? |
| MovieBox | ??? |
| TabView | QTabWidget |
| ImageBox | ??? |
| TreeView | QTreeWidget or QTreeView |
| ListView | QTreeWidget or QTreeView |
| Box | QFrame |
| ProgressBar | QProgressBar |
| RichTextBox | QTextEdit |
| WebView | QWebView |
| HtmlView | QTextBrowser |
| ResizeBox | (not needed anymore) |
| SvgBox | ??? |
| Slider | QSlider |
| ScrollBar | QScrollBar |
| SpinBox | QSpinBox |
| MenuBarItem | QAction |
| ToolBarItem | QAction |
| Report | ??? |
| Header | ??? |
| Footer | ??? |
| SerialPort | ??? |
| FormView | ??? |
| FormsView | ??? |
| Line | Line |
| ToolBarView | ??? |

| | |
|---|---|
| UdpSocket | ??? |
| ??? | PhononVideoPlayer |
| ??? | PhononSeekSlider |
| ??? | PhononVolumeSlider |

**Usable in form designer, but no classes available yet, because normally you don't need their classes:**

Vertical Layout (QVBoxLayout), Horizontal Layout (QHBoxLayout), Grid Layout (QGridLayout), Form Layout (QFormLayout), Horizontal Spacer (Spacer), Vertical Spacer (Spacer)

**Fully supported:**

QDialogButtonBox, QListView, QTreeView, QTableView, QColumnView, QTableWidget, QGraphicsView, QCalenderWidget, QLCDNumber

QGroupBox, QScrollArea, QToolBox, QTabWidget, QStackedWidget, QFrame, QFontComboBox, QPlainTextEdit, QDial,

QDockWidget, QLineEdit, QTextEdit, QSpinBox, QDoubleSpinBox, QTimeEdit, QDateEdit, QDateTimeEdit, Horizontal ScrollBar (QScrollBar), Vertical ScrollBar (QScrollBar), Horizontal Slider (QSlider), Vertical Slider (QSlider), QTextBrowser, Horizontal Line (Line), Vertical Line (Line)

QLabel

## *auto-conversion between data types*

Boolean may be assigned an object of

- Integer
- Float

Integer may be assigned an object of

- Float
- Boolean

Float may be assigned an object of

- Integer

- Boolean

## *Create A New Project*

**Projects keep your work together.** When developing an appication in Basic For Qt®, you work mainly with projects. A project is a collection of files that make up your application. You create a project to manage and organize these files. Basic For Qt® provides an easy yet sophisticated system to manage the collection of files that make up a project. The project window shows each item in a project. Starting a new application with Basic For Qt® begins with the creation of a project. So before you can construct an application with Basic For Qt®, you need to create a new project. A project consists of many separate files collected in one project directory.

## *Order of Init on app startup*

1. Init of Application.QApplication
2. Init of Global.QObject
3. Init of MainWindow.QMainWindow

## *How to access the QMenuBar object of the mainwindow*

Place the following code in the MainWindow.QMainWindow file:

```
Outlet menubar As QMenuBar ' be sure that the object name in Qt
designer matches this name

Signal on_menubar_hovered(Action As QAction)
  MsgBox("on_menubar_hovered " & Action.Text)
End Signal

Signal on_menubar_triggered(Action As QAction)
  MsgBox("on_menubar_triggered " & Action.Text)
End Signal
```

## *How to access the QToolBar object of the mainwindow*

Be sure that you created a toolbar for the mainwindow in Qt Designer. Place the following code in the MainWindow.QMainWindow file:

```
Outlet toolbar As QToolBar ' be sure that the object name in Qt
designer matches this name
```

### How to access the QStatusBar object of the mainwindow

Place the following code in the MainWindow.QMainWindow file:

```
Outlet statusbar As QStatusBar ' be sure that the object name in
Qt designer matches this name
```

### How to get the MDI focus changes

Place the following code in the MainWindow.QMainWindow file:

```
Signal on_mdiArea_GotFocusMDI(theWidget As QWidget) ' be sure that
the object name in Qt designer matches this name
  MsgBox("on_mdiArea_GotFocusMDI", theWidget)
End Signal
```

### QButtonGroup

The Qt documentation says that you normally do not need QButtonGroup, because radio buttons with the same parent are automatically part of an exclusive group, so you normally don't need to do anything else.

To be continued...