

PictureCrypt

1.4.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>PictureCrypt</b>	<b>1</b>
1.1	About . . . . .	1
1.2	Download . . . . .	1
1.3	Realisation . . . . .	1
1.4	How can someone use it? . . . . .	1
1.5	Structure of the project. . . . .	2
1.6	External use . . . . .	2
1.7	JPHS use . . . . .	3
1.8	License . . . . .	4
1.9	Contact us . . . . .	4
<b>2</b>	<b>Namespace Index</b>	<b>5</b>
2.1	Namespace List . . . . .	5
<b>3</b>	<b>Hierarchical Index</b>	<b>7</b>
3.1	Class Hierarchy . . . . .	7
<b>4</b>	<b>Class Index</b>	<b>9</b>
4.1	Class List . . . . .	9
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11

<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	ErrorsDictSetup Namespace Reference	13
6.1.1	Variable Documentation	13
6.1.1.1	data	13
6.1.1.2	f	13
6.1.1.3	filename	13
6.1.1.4	indent	13
6.1.1.5	input_data	14
6.1.1.6	key	14
6.1.1.7	raw	14
6.1.1.8	value	14
6.2	Ui Namespace Reference	14
<b>7</b>	<b>Class Documentation</b>	<b>15</b>
7.1	AboutPC Class Reference	15
7.1.1	Detailed Description	16
7.1.2	Constructor & Destructor Documentation	16
7.1.2.1	AboutPC(QWidget *parent=0)	16
7.1.2.2	~AboutPC()	16
7.1.3	Member Function Documentation	16
7.1.3.1	setVersion(QString version)	16
7.2	ControllerPC Class Reference	17
7.2.1	Detailed Description	18
7.2.2	Constructor & Destructor Documentation	18
7.2.2.1	ControllerPC()	18
7.2.3	Member Function Documentation	19
7.2.3.1	abortCircuit	19
7.2.3.2	setJPHSDir	19
7.2.4	Member Data Documentation	19
7.2.4.1	version	19
7.2.4.2	versionString	20

7.3	EncryptDialog Class Reference . . . . .	20
7.3.1	Detailed Description . . . . .	21
7.3.2	Constructor & Destructor Documentation . . . . .	22
7.3.2.1	EncryptDialog(QByteArray _data, QWidget *parent=0) . . . . .	22
7.3.2.2	~EncryptDialog() . . . . .	22
7.3.3	Member Function Documentation . . . . .	22
7.3.3.1	on_bitsSlider_valueChanged . . . . .	22
7.3.3.2	on_buttonBox_accepted . . . . .	22
7.3.3.3	on_buttonBox_rejected . . . . .	23
7.3.3.4	on_fileButton_clicked . . . . .	23
7.3.3.5	zip() . . . . .	23
7.3.4	Member Data Documentation . . . . .	24
7.3.4.1	bitsUsed . . . . .	24
7.3.4.2	compr_data . . . . .	24
7.3.4.3	data . . . . .	24
7.3.4.4	goodPercentage . . . . .	24
7.3.4.5	image . . . . .	24
7.3.4.6	inputFileName . . . . .	24
7.3.4.7	key . . . . .	24
7.3.4.8	size . . . . .	25
7.3.4.9	success . . . . .	25
7.3.4.10	val . . . . .	25
7.4	ModelPC Class Reference . . . . .	25
7.4.1	Detailed Description . . . . .	28
7.4.2	Member Enumeration Documentation . . . . .	28
7.4.2.1	CryptMode . . . . .	28
7.4.3	Constructor & Destructor Documentation . . . . .	28
7.4.3.1	ModelPC() . . . . .	28
7.4.4	Member Function Documentation . . . . .	29
7.4.4.1	alert . . . . .	29

7.4.4.2	alertView	30
7.4.4.3	circuit(QImage *image, QByteArray *data, long long int countBytes)	30
7.4.4.4	Decrypt(QImage *image, QString key, int _mode=0, QString *_error=nullptr)	31
7.4.4.5	decrypt	32
7.4.4.6	decryptv1_3(QImage *image, QString key)	33
7.4.4.7	decryptv1_4(QImage *image, QString key)	34
7.4.4.8	Encrypt(QByteArray data, QImage *image, int _mode, QString key="","", int _bitsUsed=8, QString *_error=nullptr)	35
7.4.4.9	encrypt	35
7.4.4.10	encryptv1_4(QImage *image, QByteArray data, QString key)	36
7.4.4.11	fail	37
7.4.4.12	Inject(QByteArray encr_data, QImage *image, int _mode, int _bitsUsed=8, QString *_error=nullptr)	37
7.4.4.13	inject	38
7.4.4.14	jphs(QImage *image, QByteArray *data)	39
7.4.4.15	processPixelsv1_4(QImage *image, QByteArray *data, QByteArray key, bool isEncrypt, QVector< QPair< QPoint, QPair< int, int > > > *were, long long size=-1)	39
7.4.4.16	processPixel(QPoint pos, QVector< QPoint > *were, bool isEncrypt)	40
7.4.4.17	saveData	41
7.4.4.18	savelImage	41
7.4.4.19	setProgress	42
7.4.4.20	unzip(QByteArray data, QByteArray key)	42
7.4.4.21	zip(QByteArray data, QByteArray key)	43
7.4.5	Member Data Documentation	44
7.4.5.1	defaultJPHSDir	44
7.4.5.2	error	44
7.4.5.3	success	44
7.4.5.4	version	44
7.4.5.5	versionString	44
7.5	QAESEncryption Class Reference	45
7.5.1	Detailed Description	46

7.5.2	Member Enumeration Documentation . . . . .	46
7.5.2.1	Aes . . . . .	46
7.5.2.2	Mode . . . . .	47
7.5.2.3	Padding . . . . .	47
7.5.3	Constructor & Destructor Documentation . . . . .	47
7.5.3.1	QAESEncryption(QAESEncryption::Aes level, QAESEncryption::Mode mode, QAESEncryption::Padding padding=QAESEncryption::ISO) . . . . .	47
7.5.4	Member Function Documentation . . . . .	47
7.5.4.1	Crypt(QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, QAESEncryption::Padding padding=QAESEncryption::ISO) . . . . .	47
7.5.4.2	decode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL) . . . . .	48
7.5.4.3	Decrypt(QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, QAESEncryption::Padding padding=QAESEncryption::ISO) . . . . .	49
7.5.4.4	encode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL) . . . . .	50
7.5.4.5	ExpandKey(QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &key) . . . . .	51
7.5.4.6	expandKey(const QByteArray &key) . . . . .	51
7.5.4.7	RemovePadding(const QByteArray &rawText, QAESEncryption::Padding padding) . . . . .	52
7.5.4.8	removePadding(const QByteArray &rawText) . . . . .	53
7.6	ViewPC Class Reference . . . . .	53
7.6.1	Detailed Description . . . . .	55
7.6.2	Constructor & Destructor Documentation . . . . .	55
7.6.2.1	ViewPC(QWidget *parent=nullptr) . . . . .	55
7.6.2.2	~ViewPC() . . . . .	56
7.6.3	Member Function Documentation . . . . .	56
7.6.3.1	abortCircuit . . . . .	56
7.6.3.2	abortModel . . . . .	56
7.6.3.3	alert . . . . .	56
7.6.3.4	decrypt . . . . .	57
7.6.3.5	encrypt . . . . .	58

7.6.3.6	<a href="#">inject</a>	58
7.6.3.7	<a href="#">on_actionAbout_triggered</a>	59
7.6.3.8	<a href="#">on_actionHelp_triggered</a>	59
7.6.3.9	<a href="#">on_fileButton_clicked</a>	59
7.6.3.10	<a href="#">on_startButton_clicked</a>	59
7.6.4	<a href="#">Encrypting</a>	59
7.6.5	<a href="#">Decrypting</a>	60
7.6.5.1	<a href="#">requestKey()</a>	60
7.6.5.2	<a href="#">saveData</a>	60
7.6.5.3	<a href="#">saveImage</a>	61
7.6.5.4	<a href="#">setEncryptMode</a>	61
7.6.5.5	<a href="#">setJPHSDir</a>	62
7.6.5.6	<a href="#">setProgress</a>	62
7.6.5.7	<a href="#">setVersion</a>	63
7.6.6	<a href="#">Member Data Documentation</a>	63
7.6.6.1	<a href="#">dialog</a>	63
7.6.6.2	<a href="#">errorsDict</a>	64
7.6.6.3	<a href="#">progressDialogClosed</a>	64
<b>8</b>	<b><a href="#">File Documentation</a></b>	<b>65</b>
8.1	<a href="#">aboutpc.cpp File Reference</a>	65
8.2	<a href="#">aboutpc.cpp</a>	65
8.3	<a href="#">aboutpc.h File Reference</a>	66
8.4	<a href="#">aboutpc.h</a>	67
8.5	<a href="#">controllerpc.cpp File Reference</a>	67
8.6	<a href="#">controllerpc.cpp</a>	67
8.7	<a href="#">controllerpc.h File Reference</a>	68
8.7.1	<a href="#">Detailed Description</a>	68
8.8	<a href="#">controllerpc.h</a>	69
8.9	<a href="#">encryptdialog.cpp File Reference</a>	69
8.10	<a href="#">encryptdialog.cpp</a>	69



8.11 encryptdialog.h File Reference . . . . .	70
8.12 encryptdialog.h . . . . .	72
8.13 ErrorsDict.json File Reference . . . . .	72
8.14 ErrorsDict.json . . . . .	72
8.15 ErrorsDictSetup.py File Reference . . . . .	72
8.16 ErrorsDictSetup.py . . . . .	73
8.17 main.cpp File Reference . . . . .	73
8.17.1 Function Documentation . . . . .	73
8.17.1.1 main(int argc, char *argv[]) . . . . .	73
8.18 main.cpp . . . . .	74
8.19 modelpc.cpp File Reference . . . . .	74
8.20 modelpc.cpp . . . . .	74
8.21 modelpc.h File Reference . . . . .	84
8.21.1 Detailed Description . . . . .	85
8.22 modelpc.h . . . . .	85
8.23 qaesencryption.cpp File Reference . . . . .	86
8.23.1 Function Documentation . . . . .	87
8.23.1.1 multiply(uint8 x, uint8 y) . . . . .	87
8.23.1.2 xTime(uint8 x) . . . . .	88
8.24 qaesencryption.cpp . . . . .	88
8.25 qaesencryption.h File Reference . . . . .	94
8.26 qaesencryption.h . . . . .	95
8.27 viewpc.cpp File Reference . . . . .	97
8.28 viewpc.cpp . . . . .	97
8.29 viewpc.h File Reference . . . . .	101
8.29.1 Detailed Description . . . . .	102
8.30 viewpc.h . . . . .	102



# Chapter 1

## PictureCrypt

Project made using QT Creator in C++

### 1.1 About

A simple steganography project which hides data in images This project is built using MVC pattern and features GUI. [Qt](#) and [QAESEncryption](#) by [bricke](#) were used.

### 1.2 Download

Get the binary files at [latest release page](#) Or download latest **UNSTABLE** binary file for linux [here](#)

### 1.3 Realisation

To create the encrypted image, you need to select any file for encryption, then using [EncryptDialog](#) you select the image to store the data. Then output image is generated.

#### Attention

Output image format available is .PNG, because .jpg isn't lossless, so the pixels containing data would be seriously simplified and the data damaged. .BMP isn't used, because noone really uses it and .PNG is just compressed .BMP (more or less)

#### Note

JPHS support is under development :D

### 1.4 How can someone use it?

Well... Anyone who wants to securely communicate. For example your boss watches your inbox, so you do the work and don't chat with your friends about the bar, they've just visited. Using this app you can send them a photo of your desk, saying it's my new working space, but inside the image there is secret message saying "Wanna get another beer tonight? xD". Boss sees this image, but doesn't spot anything. Great example...

## 1.5 Structure of the project.

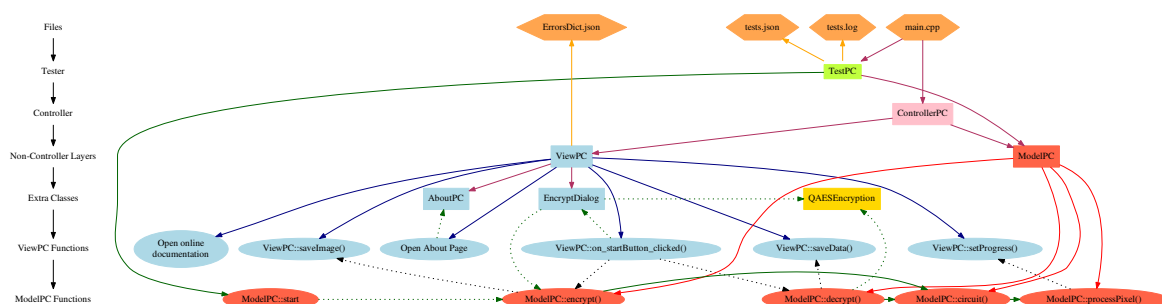
Project is done via MVC Pattern. View and Model layers are totally isolated and run on different threads.

Code from controller.cpp

```
view = new ViewPC();
model = new ModelPC(version);
QThread * modelThread = new QThread();
model->moveToThread(modelThread);
modelThread->start();
```

So when Model is hard-working, View layer is just fine.

Layers also have a ton of functions, so here is a scheme, that I was doing for about 10 hours, which demonstrates the most important functions and classes in the project. And everything is clickable here, so try it out!



Well... I think you didn't quite understand what is happening here... So hop into my "User-friendly" Documentation!

See source on <https://github.com/waleko/PictureCrypt>

Note

QAESEncryption class done by Bricke

## 1.6 External use

ModelPC class can be used externally (without UI)

**Note**

TestPC class was introduced recently, its use is advised.

```
#include <modelpc.h>
#include <testpc.h>
#include <QByteArray>
#include <QImage>

#include <QDebug> // Just for demonstration use

...

if(TestPC::Test())
    return;
ModelPC * model = new ModelPC();

// Embedding
QImage * resultImage = model->start(QByteArray data, // Data to be embedded
                                   QImage *image, // Image for embedding
                                   int mode = 0, // Mode of embedding
                                   QString key = "", // Key for extra-encryption (if empty, key will be
                                   generated automatically)
                                   int bitsUsed = 8, // Bits per Byte used (better explanation
                                   will be "ok"
                                   QString *error = nullptr); // Error output, if everything is ok, error
if(*error != "ok")
    return;
// Note *error is just a code of error (like "muchdata", dictionary of error codes is also available on
// github.

// De-embedding
QByteArray output = model->decrypt(QImage * image, // Image with hidden data
                                   QString *_error = nullptr); // Error output
if(data == output)
    qDebug() << "Great success!";
else
    qDebug() << "Fiasco :(";
```

**See also**

[ModelPC](#), [ModelPC::ModelPC](#), [ModelPC::saveData](#), [ModelPC::saveImage](#), [ModelPC::alertView](#), [ModelPC::setProgress](#)

## 1.7 JPBS use

The newer versions of the app have jpbs support, but they don't have jpbs built in as it is provided under GNU General Public License v3.0, is "for test purposes only" and is illegal in some countries, so...

**Attention**

We support JPBS, but we don't use any responsibility for it, we never used or downloaded it, we just used .exe output in the web, and it somehow works by chance. All responsibility for using jpbs is on you, that is why we use made only optionally. That means that to use jpbs with our app you will have to download the jpbs yourself and specify the jpbs directory. However we provide link to the site where you can download the supported version of the jpbs: <http://linux01.gwdg.de/~alatham/stego.html> As it's not our site publishing the dangerous zip file, we just put link to that site (Google does that too, so what? Sue Google?), This text is subject to United Nations' Universal Declaration of Human Rights, (see Article 19 <http://www.un.org/en/universal-declaration-human-rights>):

Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers.

And I typed this link randomly, and I'm scared...

## 1.8 License

This software is provided under the [UNLICENSE](#)

## 1.9 Contact us

Visit my site: <https://www.alexkovrigin.me>

Email me at [a.kovrigin0@gmail.com](mailto:a.kovrigin0@gmail.com)

### Author

Alex Kovrigin (waleko)

### Copyright

Alex Kovrigin 2018

## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">ErrorsDictSetup</a> . . . . .	<a href="#">13</a>
<a href="#">Ui</a> . . . . .	<a href="#">14</a>





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

QDialog	
AboutPC . . . . .	15
EncryptDialog . . . . .	20
QMainWindow	
ViewPC . . . . .	53
QObject	
ControllerPC . . . . .	17
ModelIPC . . . . .	25
QAESEncryption . . . . .	45



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">AboutPC</a>	The About Page dialog . . . . .	15
<a href="#">ControllerPC</a>	The <a href="#">ControllerPC</a> class Controller class, which controls View and Model layers . . . . .	17
<a href="#">EncryptDialog</a>	Class to get the image and key to store secret info . . . . .	20
<a href="#">ModelPC</a>	The <a href="#">ModelPC</a> class Model Layer of the app. Main class that does the work of PictureCrypt logic Controlled by <a href="#">ControllerPC</a> . . . . .	25
<a href="#">QAESEncryption</a>	Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: <a href="https://github.com/bricke/Qt-AES">https://github.com/bricke/Qt-AES</a> . . . . .	45
<a href="#">ViewPC</a>	View layer of the app. Controls <a href="#">EncryptDialog</a> and ProgressDialog . . . . .	53



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">aboutpc.cpp</a>	65
<a href="#">aboutpc.h</a>	66
<a href="#">controllerpc.cpp</a>	67
<a href="#">controllerpc.h</a>	68
<a href="#">encryptdialog.cpp</a>	69
<a href="#">encryptdialog.h</a>	70
<a href="#">ErrorsDict.json</a>	72
<a href="#">ErrorsDictSetup.py</a>	72
<a href="#">main.cpp</a>	73
<a href="#">modelpc.cpp</a>	74
<a href="#">modelpc.h</a>	84
<a href="#">qaesencryption.cpp</a>	86
<a href="#">qaesencryption.h</a>	94
<a href="#">viewpc.cpp</a>	97
<a href="#">viewpc.h</a>	101



## Chapter 6

# Namespace Documentation

### 6.1 ErrorsDictSetup Namespace Reference

#### Variables

- string `filename` = 'ErrorsDict.json'
- `raw` = open(`filename`, 'r')
- `data` = json.load(`raw`)
- `input_data` = input()
- `key`
- `value`
- `f`
- `indent`

#### 6.1.1 Variable Documentation

##### 6.1.1.1 ErrorsDictSetup.data = json.load(raw)

Definition at line 6 of file [ErrorsDictSetup.py](#).

##### 6.1.1.2 ErrorsDictSetup.f

Definition at line 22 of file [ErrorsDictSetup.py](#).

##### 6.1.1.3 string ErrorsDictSetup.filename = 'ErrorsDict.json'

Definition at line 2 of file [ErrorsDictSetup.py](#).

##### 6.1.1.4 ErrorsDictSetup.indent

Definition at line 22 of file [ErrorsDictSetup.py](#).

#### 6.1.1.5 ErrorsDictSetup.input\_data = input()

Definition at line 14 of file [ErrorsDictSetup.py](#).

#### 6.1.1.6 ErrorsDictSetup.key

Definition at line 17 of file [ErrorsDictSetup.py](#).

#### 6.1.1.7 ErrorsDictSetup.raw = open(filename, 'r')

Definition at line 4 of file [ErrorsDictSetup.py](#).

#### 6.1.1.8 ErrorsDictSetup.value

Definition at line 17 of file [ErrorsDictSetup.py](#).

## 6.2 Ui Namespace Reference



## Chapter 7

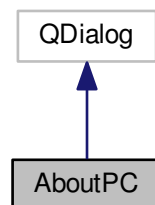
# Class Documentation

### 7.1 AboutPC Class Reference

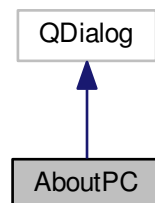
The [AboutPC](#) class The About Page dialog.

```
#include <aboutpc.h>
```

Inheritance diagram for AboutPC:



Collaboration diagram for AboutPC:



## Public Member Functions

- [AboutPC](#) (QWidget \*parent=0)
- [~AboutPC](#) ()
- void [setVersion](#) (QString version)  
[AboutPC::setVersion](#) Function to set the version display.

### 7.1.1 Detailed Description

The [AboutPC](#) class The About Page dialog.

Definition at line 12 of file [aboutpc.h](#).

### 7.1.2 Constructor & Destructor Documentation

7.1.2.1 [AboutPC::AboutPC](#) ( QWidget \* *parent* = 0 ) [explicit]

Definition at line 4 of file [aboutpc.cpp](#).

7.1.2.2 [AboutPC::~~AboutPC](#) ( )

Definition at line 11 of file [aboutpc.cpp](#).

### 7.1.3 Member Function Documentation

7.1.3.1 void [AboutPC::setVersion](#) ( QString *version* )

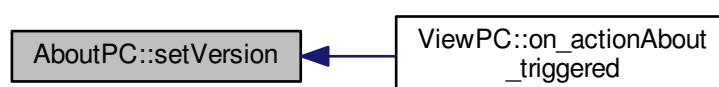
[AboutPC::setVersion](#) Function to set the version display.

Parameters

<i>version</i>	Version as QString
----------------	--------------------

Definition at line 19 of file [aboutpc.cpp](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

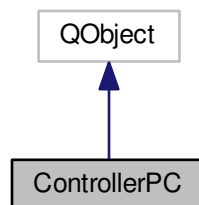
- [aboutpc.h](#)
- [aboutpc.cpp](#)

## 7.2 ControllerPC Class Reference

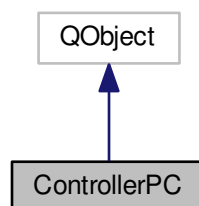
The [ControllerPC](#) class Controller class, which controls View and Model layers.

```
#include <controllerpc.h>
```

Inheritance diagram for ControllerPC:



Collaboration diagram for ControllerPC:



### Public Slots

- void [abortCircuit](#) ()  
*[ControllerPC::abortCircuit](#) Slot to be called when ProgressDialog in [ViewPC](#) is closed. It flags [ModelPC](#) to stop.*
- void [setJPHSDir](#) (QString dir)  
*[ControllerPC::setJPHSDir](#) Sets JPHS default dir.*

## Public Member Functions

- [ControllerPC](#) ( )

[ControllerPC::ControllerPC](#) Constructor of controller Constructor runs auto-test for [ModelPC](#), creates Model Class ([ModelPC](#)) and View Class ([ViewPC](#)). All signals and slots are connected here.

## Public Attributes

- long int [version](#)

*version* Version of the app

- QString [versionString](#)

*versionString* Version of the app as QString.

### 7.2.1 Detailed Description

The [ControllerPC](#) class Controller class, which controls View and Model layers.

See also

[ViewPC](#), [ModelPC](#)

Definition at line 20 of file [controllerpc.h](#).

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 ControllerPC::ControllerPC ( )

[ControllerPC::ControllerPC](#) Constructor of controller Constructor runs auto-test for [ModelPC](#), creates Model Class ([ModelPC](#)) and View Class ([ViewPC](#)). All signals and slots are connected here.

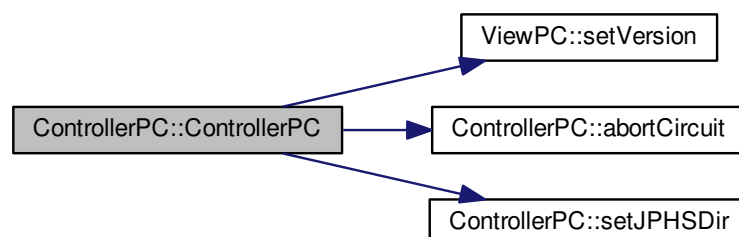
Controller class

Note

Version of the app is specified here.

Definition at line 9 of file [controllerpc.cpp](#).

Here is the call graph for this function:



### 7.2.3 Member Function Documentation

#### 7.2.3.1 void ControllerPC::abortCircuit ( ) [slot]

[ControllerPC::abortCircuit](#) Slot to be called when ProgressDialog in [ViewPC](#) is closed. It flags [ModelPC](#) to stop.

Definition at line 37 of file [controllerpc.cpp](#).

Here is the caller graph for this function:



#### 7.2.3.2 void ControllerPC::setJPHSDir ( QString dir ) [slot]

[ControllerPC::setJPHSDir](#) Sets JPHS default dir.

##### Parameters

<i>dir</i>	Directory
------------	-----------

Definition at line 45 of file [controllerpc.cpp](#).

Here is the caller graph for this function:



### 7.2.4 Member Data Documentation

#### 7.2.4.1 long int ControllerPC::version

version Version of the app

Definition at line 28 of file [controllerpc.h](#).

#### 7.2.4.2 QString ControllerPC::versionString

versionString Version of the app as QString.

Definition at line 32 of file [controllerpc.h](#).

The documentation for this class was generated from the following files:

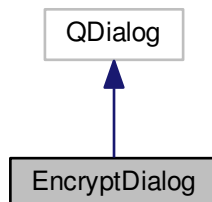
- [controllerpc.h](#)
- [controllerpc.cpp](#)

### 7.3 EncryptDialog Class Reference

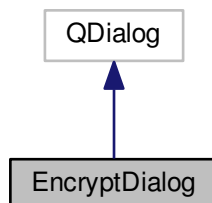
The [EncryptDialog](#) class Class to get the image and key to store secret info.

```
#include <encryptdialog.h>
```

Inheritance diagram for EncryptDialog:



Collaboration diagram for EncryptDialog:



## Public Slots

- void [on\\_fileButton\\_clicked](#) ()  
*EncryptDialog::on\_fileButton\_clicked* Slot to select the image.
- void [on\\_buttonBox\\_accepted](#) ()  
*EncryptDialog::on\_buttonBox\_accepted* Slot to start the encryption. Successful closing of the app.
- void [on\\_buttonBox\\_rejected](#) ()  
*EncryptDialog::on\_buttonBox\_rejected* Slot to reject the encryption.
- void [on\\_bitsSlider\\_valueChanged](#) (int value)  
*EncryptDialog::on\_bitsSlider\_valueChanged* Slot if value of the bits slider is changed.

## Public Member Functions

- [EncryptDialog](#) (QByteArray \_data, QWidget \*parent=0)  
*EncryptDialog::EncryptDialog* Constructor of the class. Input data is saved here and some variables are set here.
- [~EncryptDialog](#) ()
- QByteArray [zip](#) ()  
*EncryptDialog::zip* Zipping algorithm It copresses the data and then compresses it using qCompress()

## Public Attributes

- QByteArray [data](#)  
*data* Input data
- bool [success](#)  
*success* Flag, if image was successfully selected and data was encrypted.
- QByteArray [compr\\_data](#)  
*compr\_data* Compressed data, aka Output data.
- QString [inputFileName](#)  
*inputFileName* Filename of the image.
- long long int [size](#)  
*size* Size of the image in square pixels
- QString [key](#)  
*key* Key to be used for encryption in *EncryptDialog::zip*
- bool [goodPercentage](#)  
*goodPercentage* Flag if area of the used data via encryption is less than 70% of the area of the image.
- int [val](#)  
*val* Value of the slider
- int [bitsUsed](#)  
*bitsUsed* Bits used per byte of pixel.
- QImage [image](#)  
*image* Inputted image

### 7.3.1 Detailed Description

The [EncryptDialog](#) class Class to get the image and key to store secret info.

#### Note

Not the most important and well written class.

#### See also

[ViewPC](#)

Definition at line 21 of file [encryptdialog.h](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 `EncryptDialog::EncryptDialog ( QByteArray _data, QWidget * parent = 0 ) [explicit]`

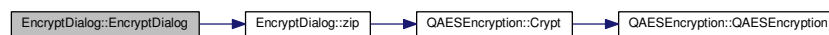
[EncryptDialog::EncryptDialog](#) Constructor of the class. Input data is saved here and some variables are set here.

Parameters

<code>_data</code>	Input data.
<code>parent</code>	Parent (not in use)

Definition at line 9 of file [encryptdialog.cpp](#).

Here is the call graph for this function:



#### 7.3.2.2 `EncryptDialog::~~EncryptDialog ( )`

Definition at line 26 of file [encryptdialog.cpp](#).

### 7.3.3 Member Function Documentation

#### 7.3.3.1 `void EncryptDialog::on_bitsSlider_valueChanged ( int value ) [slot]`

[EncryptDialog::on\\_bitsSlider\\_valueChanged](#) Slot if value of the bits slider is changed.

Parameters

<code>value</code>	Well, value
--------------------	-------------

Definition at line 107 of file [encryptdialog.cpp](#).

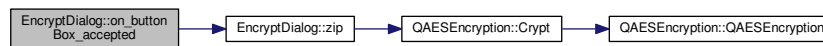
#### 7.3.3.2 `void EncryptDialog::on_buttonBox_accepted ( ) [slot]`

[EncryptDialog::on\\_buttonBox\\_accepted](#) Slot to start the encryption. Successful closing of the app.

Definition at line 82 of file [encryptdialog.cpp](#).



Here is the call graph for this function:



#### 7.3.3.3 void EncryptDialog::on\_buttonBox\_rejected ( ) [slot]

[EncryptDialog::on\\_buttonBox\\_rejected](#) Slot to reject the encryption.

Definition at line 98 of file [encryptdialog.cpp](#).

#### 7.3.3.4 void EncryptDialog::on\_fileButton\_clicked ( ) [slot]

[EncryptDialog::on\\_fileButton\\_clicked](#) Slot to select the image.

Definition at line 57 of file [encryptdialog.cpp](#).

#### 7.3.3.5 QByteArray EncryptDialog::zip ( )

[EncryptDialog::zip](#) Zipping algorithm It copresses the data and then compresses it using qCompress()

Returns

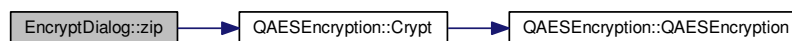
Returns Compressed data.

See also

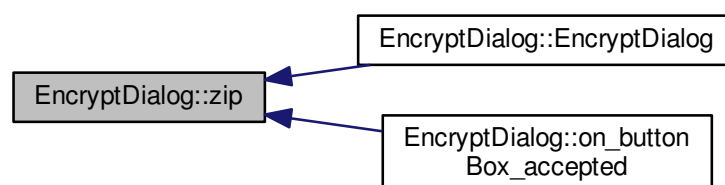
[ModelPC::unzip](#)

Definition at line 46 of file [encryptdialog.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.3.4 Member Data Documentation

#### 7.3.4.1 int EncryptDialog::bitsUsed

bitsUsed Bits used per byte of pixel.

See also

[ModelPC::circuit](#)

Definition at line 75 of file [encryptdialog.h](#).

#### 7.3.4.2 QByteArray EncryptDialog::compr\_data

compr\_data Compressed data, aka Output data.

Definition at line 50 of file [encryptdialog.h](#).

#### 7.3.4.3 QByteArray EncryptDialog::data

data Input data

Definition at line 42 of file [encryptdialog.h](#).

#### 7.3.4.4 bool EncryptDialog::goodPercentage

goodPercentage Flag if area of the used data via encryption is less than 70% of the area of the image.

Definition at line 66 of file [encryptdialog.h](#).

#### 7.3.4.5 QImage EncryptDialog::image

image Inputted image

Definition at line 79 of file [encryptdialog.h](#).

#### 7.3.4.6 QString EncryptDialog::inputFileName

inputFileName Filename of the image.

Definition at line 54 of file [encryptdialog.h](#).

#### 7.3.4.7 QString EncryptDialog::key

key Key to be used for encryption in EncryptDialog::zip

Definition at line 62 of file [encryptdialog.h](#).

#### 7.3.4.8 long long int EncryptDialog::size

size Size of the image in square pixels

Definition at line 58 of file [encryptdialog.h](#).

#### 7.3.4.9 bool EncryptDialog::success

success Flag, if image was successfully selected and data was encrypted.

Definition at line 46 of file [encryptdialog.h](#).

#### 7.3.4.10 int EncryptDialog::val

val Value of the slider

Definition at line 70 of file [encryptdialog.h](#).

The documentation for this class was generated from the following files:

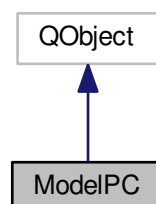
- [encryptdialog.h](#)
- [encryptdialog.cpp](#)

## 7.4 ModelPC Class Reference

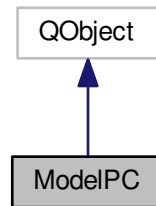
The [ModelPC](#) class Model Layer of the app. Main class that does the work of PictureCrypt logic Controlled by [ControllerPC](#).

```
#include <modelpc.h>
```

Inheritance diagram for ModelPC:



Collaboration diagram for ModelPC:



## Public Types

- enum `CryptMode` { `NotDefined`, `v1_3`, `v1_4`, `jphs_mode` }

## Public Slots

- QImage \* `encrypt` (QByteArray data, QImage \*image, int \_mode, QString key="", int \_bitsUsed=8, QString \*\_error=NULLPTR)  
*ModelPC::encrypt* Slot to zip and inject data and provide it with some extra stuff After completion start standard  
*ModelPC::inject* Isn't used in PictureCrypt, but used can be used in other - custom projects.
- QImage \* `inject` (QByteArray encr\_data, QImage \*image, int \_mode, int \_bitsUsed=8, QString \*\_error=NULLPTR)  
*ModelPC::inject* Slot to be called when encrypt mode in *ViewPC* is selected and started.
- QByteArray `decrypt` (QImage \*image, QString key, int \_mode=0, QString \*\_error=NULLPTR)  
*ModelPC::decrypt* Slot to be called when decrypt mode in *ViewPC* is selected and started.
- void `fail` (QString message)  
*ModelPC::fail* Slot to stop execution of crypton.
- void `alert` (QString message, bool isWarning=false)  
*ModelPC::alert* Function emits signal *ModelPC::alertView* and calls *ViewPC::alert*.

## Signals

- void `alertView` (QString messageCode, bool isWarning)  
*alertView* Signal to be called to create MessageBox.
- void `saveData` (QByteArray data)  
*saveData* Signal to be called to save data from *ModelPC::decrypt*.
- void `saveImage` (QImage \*image)  
*saveImage* Signal to be called to save image from *ModelPC::encrypt*.
- void `setProgress` (int val)  
*setProgress* Signal to be called to set progress of ProgressDialog.

## Public Member Functions

- [ModelPC](#) ()  
*ModelPC::ModelPC Constructor Unit tests are run here.*
- QByteArray [unzip](#) (QByteArray data, QByteArray key)  
*ModelPC::unzip Unzip data from [ModelPC::decrypt](#). Just mirrored [EncryptDialog::zip](#).*

## Static Public Member Functions

- static QImage \* [Encrypt](#) (QByteArray data, QImage \*image, int \_mode, QString key="", int \_bitsUsed=8, QString \*\_error=NULLPTR)
- static QImage \* [Inject](#) (QByteArray encr\_data, QImage \*image, int \_mode, int \_bitsUsed=8, QString \*\_error=NULLPTR)
- static QByteArray [Decrypt](#) (QImage \*image, QString key, int \_mode=0, QString \*\_error=NULLPTR)

## Public Attributes

- bool [success](#)  
*success Flag that true by default, but in case of error or cancelling of ProgressDialog it turns to false, which stops execution of [ModelPC::circuit](#)*
- long [version](#)  
*version Version of the class*
- QString [versionString](#)  
*versionString Version as string*
- QString [defaultJPHSDir](#)  
*defaultJPHSDir Default JPHS directory*

## Protected Member Functions

- void [circuit](#) (QImage \*image, QByteArray \*data, long long int countBytes)  
*ModelPC::circuit The brain of the app. Via special circuit stores data in image.*
- void [jphs](#) (QImage \*image, QByteArray \*data)  
*ModelPC::jphs JPHS function to use jphide and jpseek (currently under development)*
- void [processPixel](#) (QPoint pos, QVector< QPoint > \*were, bool isEncrypt)  
*ModelPC::processPixel Processes every pixel. Reads its contains or writes data.*
- void [encryptv1\\_4](#) (QImage \*image, QByteArray data, QString key)  
*ModelPC::encryptv1\_4 Encrypts and injects data to image used in v1.4+.*
- QByteArray [decryptv1\\_3](#) (QImage \*image, QString key)  
*ModelPC::decryptv1\_3 Decrypts data from image in v1.3.*
- QByteArray [decryptv1\\_4](#) (QImage \*image, QString key)  
*ModelPC::decryptv1\_4 Decrypts data from image in v1.4+.*
- void [proccessPixelsv1\\_4](#) (QImage \*image, QByteArray \*data, QByteArray key, bool isEncrypt, QVector< QPair< QPoint, QPair< int, int > > > \*were, long long size=-1)  
*ModelPC::proccessPixelsv1\_4 Hides (or retrieves) data to/from pixels.*
- QByteArray [zip](#) (QByteArray data, QByteArray key)  
*ModelPC::zip Zip function, copy of [EncryptDialog::zip](#) Used for [ModelPC](#) in custom projects, other than PictureCrypt.*

## Protected Attributes

- `QString * error`  
*error Current error*

### 7.4.1 Detailed Description

The [ModelPC](#) class Model Layer of the app. Main class that does the work of PictureCrypt logic Controlled by [ControllerPC](#).

See also

[ViewPC](#), [ControllerPC](#)

Author

Alex Kovrigin (waleko)

Definition at line 33 of file [modelpc.h](#).

### 7.4.2 Member Enumeration Documentation

#### 7.4.2.1 enum `ModelPC::CryptMode`

Enumerator

***NotDefined***  
***v1\_3***  
***v1\_4***  
***jphs\_mode***

Definition at line 38 of file [modelpc.h](#).

### 7.4.3 Constructor & Destructor Documentation

#### 7.4.3.1 `ModelPC::ModelPC ( )`

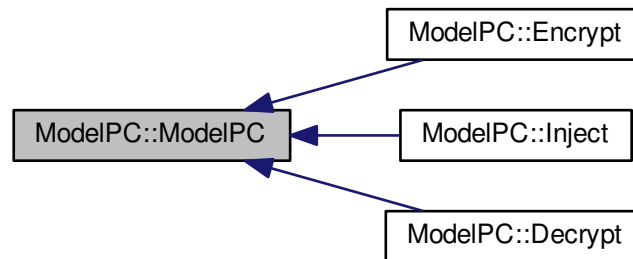
[ModelPC::ModelPC](#) Constructor Unit tests are run here.

See also

[ControllerPC](#), [ViewPC](#)

Definition at line 9 of file [modelpc.cpp](#).

Here is the caller graph for this function:



#### 7.4.4 Member Function Documentation

7.4.4.1 `void ModelPC::alert ( QString message, bool isWarning = false ) [slot]`

[ModelPC::alert](#) Function emits signal [ModelPC::alertView](#) and calls [ViewPC::alert](#).

Parameters

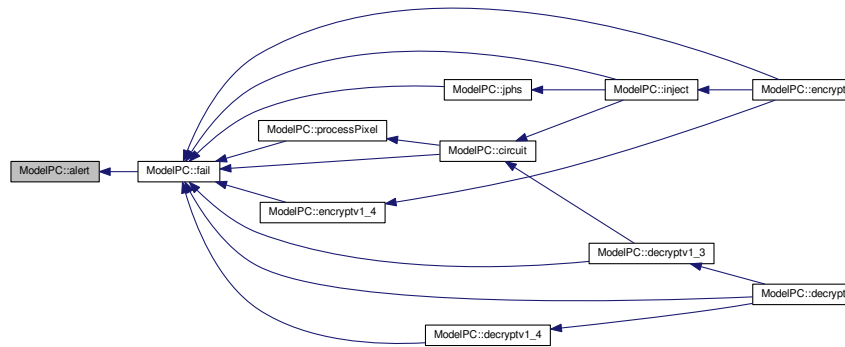
<i>message</i>	Message to be transmitted.
<i>isWarning</i>	Flag if message is critical.

See also

[ViewPC::alert](#)

Definition at line 940 of file [modelpc.cpp](#).

Here is the caller graph for this function:



#### 7.4.4.2 void ModelPC::alertView ( QString *messageCode*, bool *isWarning* ) [signal]

alertView Signal to be called to create MessageBox.

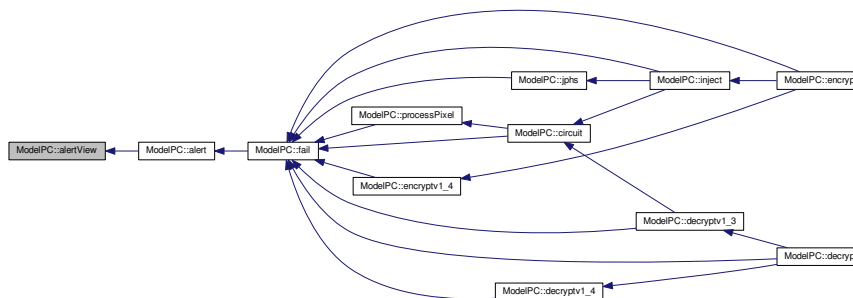
##### Parameters

<i>messageCode</i>	Message Code to be shown.
<i>isWarning</i>	Flag if message is critical.

See also

[ModelPC::alert](#), [ViewPC::alert](#)

Here is the caller graph for this function:



#### 7.4.4.3 void ModelPC::circuit ( QImage \* *image*, QByteArray \* *data*, long long int *countBytes* ) [protected]

[ModelPC::circuit](#) The brain of the app. Via special circuit stores data in image.

The circuit itself can be found in documentation or in commentaries in source.



## Parameters

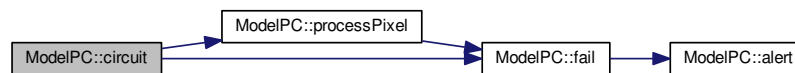
<i>image</i>	Image to be processed.
<i>data</i>	Data to be processed.
<i>countBytes</i>	Number of bytes to be read or written.

## See also

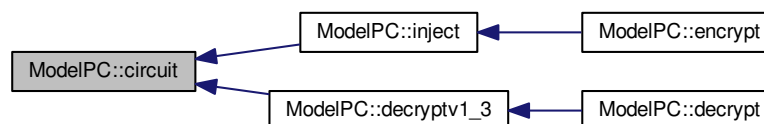
[ModelPC::processPixel](#)

Definition at line 359 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.4.4.4 QByteArray ModelPC::Decrypt ( QImage \* image, QString key, int \_mode = 0, QString \* \_error = nullptr ) [static]

Definition at line 34 of file [modelpc.cpp](#).

Here is the call graph for this function:



7.4.4.5 `QByteArray ModelPC::decrypt ( QImage * image, QString key, int _mode = 0, QString * _error = nullptr )`  
[slot]

[ModelPC::decrypt](#) Slot to be called when decrypt mode in [ViewPC](#) is selected and started.

## Parameters

<i>image</i>	Image to be decrypted.
<i>key</i>	Keyphrase with which the data is injected
<i>_mode</i>	Mode for decryption
<i>_error</i>	Error output

## Returns

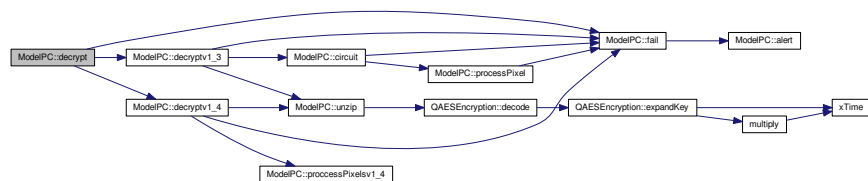
Returns decrypted data

## See also

[ViewPC::on\\_startButton\\_clicked](#), [ModelPC::inject](#), [ModelPC::circuit](#)

Definition at line 212 of file [modelpc.cpp](#).

Here is the call graph for this function:



#### 7.4.4.6 QByteArray ModelPC::decryptv1\_3 ( QImage \* *image*, QString *key* ) [protected]

[ModelPC::decryptv1\\_3](#) Decrypts data from image in v1.3.

## Parameters

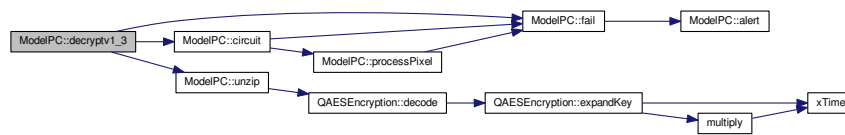
<i>image</i>	Image with data
<i>key</i>	Key

## Returns

Returns obtained data

Definition at line 777 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.4.4.7 QByteArray ModelPC::decryptv1\_4 ( QImage \* *image*, QString *key* ) [protected]

[ModelPC::decryptv1\\_4](#) Decrypts data from image in v1.4+.

##### Parameters

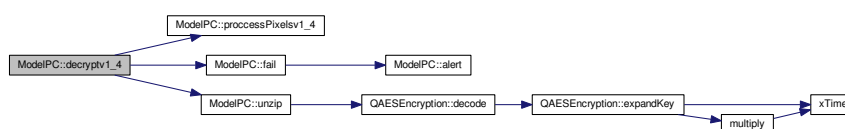
<i>image</i>	Image with data
<i>key</i>	Key

##### Returns

Returns obtained data

Definition at line 602 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**7.4.4.8** `QImage * ModelPC::Encrypt ( QByteArray data, QImage * image, int _mode, QString key = " ", int _bitsUsed = 8, QString * _error = nullptr ) [static]`

Definition at line 24 of file [modelpc.cpp](#).

Here is the call graph for this function:



**7.4.4.9** `QImage * ModelPC::encrypt ( QByteArray data, QImage * image, int _mode, QString key = " ", int _bitsUsed = 8, QString * _error = nullptr ) [slot]`

[ModelPC::encrypt](#) Slot to zip and inject data and provide it with some extra stuff After completion start standard [ModelPC::inject](#) Isn't used in PictureCrypt, but used can be used in other - custom projects.

#### Parameters

<i>data</i>	Data for embedding
<i>image</i>	Image for embedding
<i>mode</i>	Mode for embedding
<i>key</i>	Key for extra encryption
<i>_bitsUsed</i>	Bits per byte (see <code>ModelPC::bitsUsed</code> )
<i>_error</i>	Error output

#### Returns

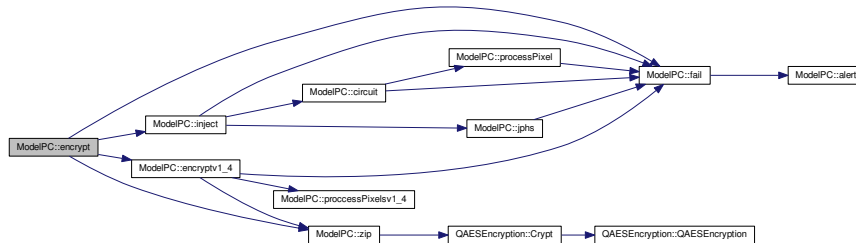
Returns image with embedded data

See also

[ModelPC::inject](#)

Definition at line 51 of file [modelpc.cpp](#).

Here is the call graph for this function:



7.4.4.10 `void ModelPC::encryptv1_4 ( QImage * image, QByteArray data, QString key )` [protected]

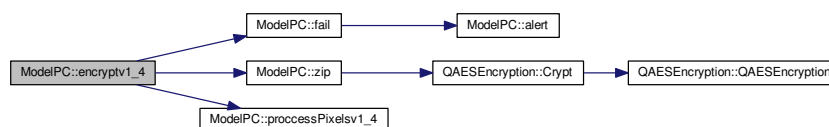
[ModelPC::encryptv1\\_4](#) Encrypts and injects data to image used in v1.4+.

Parameters

<i>image</i>	Image for injecting
<i>data</i>	Data for embedding

Definition at line 560 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.4.11 void ModelPC::fail ( QString *message* ) [slot]

[ModelPC::fail](#) Slot to stop execution of crypton.

## Parameters

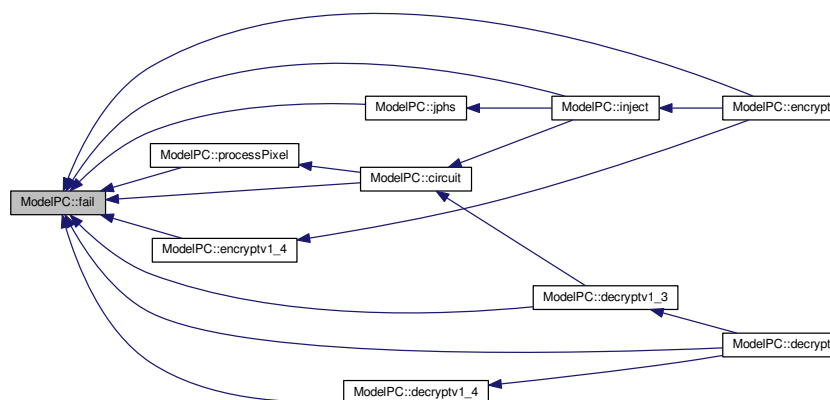
<i>message</i>	Message for user
----------------	------------------

Definition at line 283 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

7.4.4.12 QImage \* ModelPC::Inject ( QByteArray *encr\_data*, QImage \* *image*, int *\_mode*, int *\_bitsUsed* = 8, QString \* *\_error* = nullptr ) [static]

Definition at line 29 of file [modelpc.cpp](#).

Here is the call graph for this function:



**7.4.4.13** `QImage * ModelPC::inject ( QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString * _error = nullptr ) [slot]`

[ModelPC::inject](#) Slot to be called when encrypt mode in [ViewPC](#) is selected and started.

#### Parameters

<i>encr_data</i>	Data to be inserted to an image.
<i>image</i>	Image to be inserted in.
<i>mode</i>	Mode of encryption
<i>_bitsUsed</i>	Bits per byte used
<i>_error</i>	Error output

#### Returns

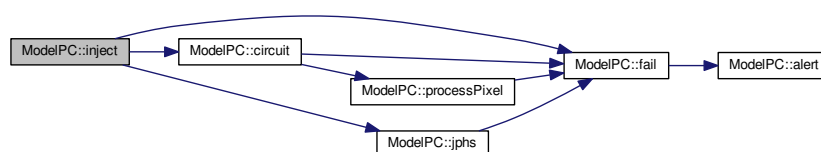
Returns image with embedded data.

#### See also

[ViewPC::on\\_startButton\\_clicked](#), [ModelPC::decrypt](#), [ModelPC::circuit](#), [ModelPC::start](#)

Definition at line 139 of file [modelpc.cpp](#).

Here is the call graph for this function:





Here is the caller graph for this function:



**7.4.4.14** `void ModelPC::jphs ( QImage * image, QByteArray * data )` `[protected]`

[ModelPC::jphs](#) JPHS function to use jpshide and jpseek (currently under development)

Parameters

<i>image</i>	Image for embedding
<i>data</i>	Data

Definition at line [298](#) of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**7.4.4.15** `void ModelPC::processPixelsv1_4 ( QImage * image, QByteArray * data, QByteArray key, bool isEncrypt, QVector< QPair< QPoint, QPair< int, int > > * were, long long size = -1 )` `[protected]`

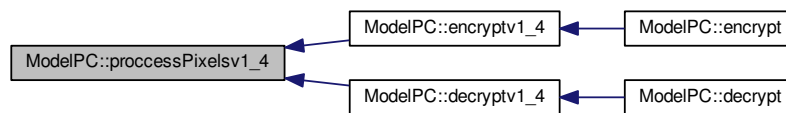
[ModelPC::processPixelsv1\\_4](#) Hides (or retrieves) data to/from pixels.

## Parameters

<i>image</i>	Original image
<i>data</i>	Data to write (Pointer to empty QByteArray if decrypting)
<i>key</i>	Key
<i>isEncrypt</i>	Mode of Crypton (true -> encryption, false -> decryption)
<i>were</i>	Were vector for visited pixels
<i>size</i>	Size of reading data, unneeded if writing

Definition at line 663 of file [modelpc.cpp](#).

Here is the caller graph for this function:



7.4.4.16 `void ModelPC::processPixel ( QPoint pos, QVector< QPoint > * were, bool isEncrypt )` [protected]

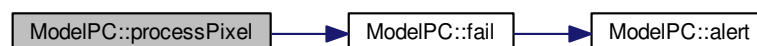
[ModelPC::processPixel](#) Processes every pixel. Reads its contains or writes data.

## Parameters

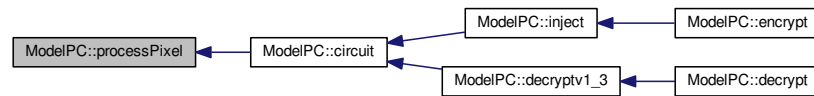
<i>pos</i>	Position of pixel
<i>were</i>	Vector array containing pixels, that were already processed.
<i>isEncrypt</i>	Mode of operation. If true encryption operations will continue, else the decryption ones.

Definition at line 500 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.4.4.17 void ModelPC::saveData ( QByteArray *data* ) [signal]

saveData Signal to be called to save data from [ModelPC::decrypt](#).

##### Parameters

<i>data</i>	Data to be saved.
-------------	-------------------

Here is the caller graph for this function:



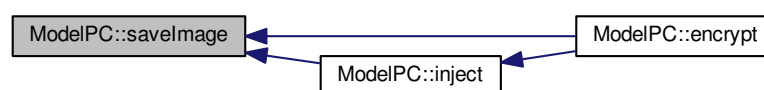
#### 7.4.4.18 void ModelPC::saveImage ( QImage \* *image* ) [signal]

saveImage Signal to be called to save image from [ModelPC::encrypt](#).

##### Parameters

<i>image</i>	Image to be saved.
--------------	--------------------

Here is the caller graph for this function:



#### 7.4.4.19 void ModelPC::setProgress ( int *val* ) [signal]

setProgress Signal to be called to set progress of ProgressDialog.

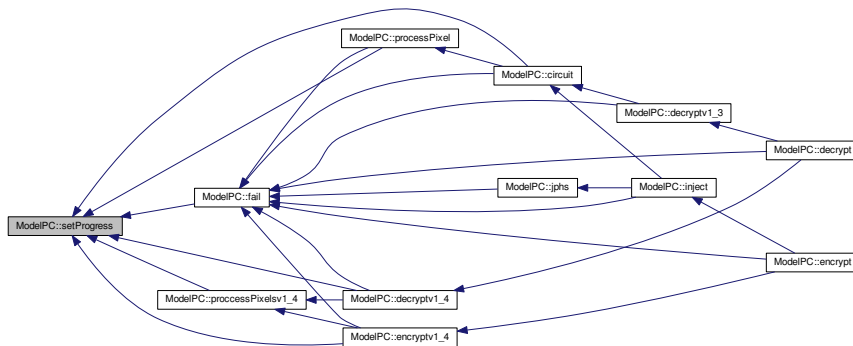
##### Parameters

<i>val</i>	Value to be set.
------------	------------------

##### See also

[ViewPC::setProgress](#)

Here is the caller graph for this function:



#### 7.4.4.20 QByteArray ModelPC::unzip ( QByteArray *data*, QByteArray *key* )

[ModelPC::unzip](#) Unzip data from [ModelPC::decrypt](#). Just mirrored [EncryptDialog::zip](#).

##### Parameters

<i>data</i>	Data to be decrypted.
<i>key</i>	Key to decrypt the data.

##### Returns

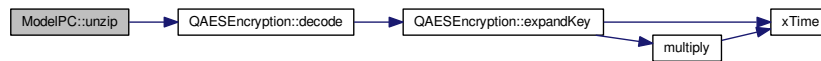
Returns data

##### See also

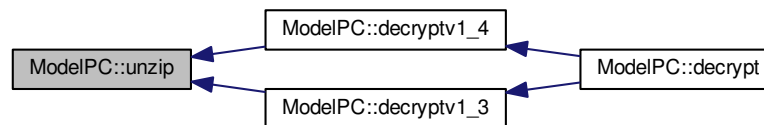
[EncryptDialog::zip](#), [ModelPC::decrypt](#), [ModelPC::zip](#)

Definition at line 879 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.4.4.21 QByteArray ModelPC::zip ( QByteArray data, QByteArray key ) [protected]

[ModelPC::zip](#) Zip function, copy of [EncryptDialog::zip](#) Used for [ModelPC](#) in custom projects, other than PictureCrypt.

##### Parameters

<i>data</i>	Data to be encrypted
<i>key</i>	Key for encryption

##### Returns

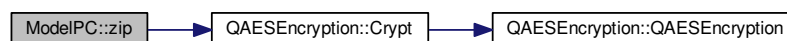
Returns decrypted data

##### See also

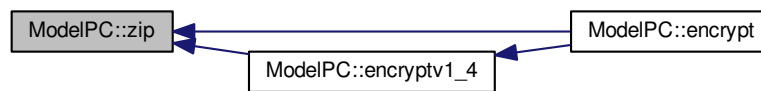
ModelPC::start, [ModelPC::inject](#), [ModelPC::unzip](#)

Definition at line 896 of file [modelpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.4.5 Member Data Documentation

### 7.4.5.1 QString ModelPC::defaultJPHSDir

defaultJPHSDir Default JPHS directory

Definition at line 94 of file [modelpc.h](#).

### 7.4.5.2 QString\* ModelPC::error [protected]

error Current error

Definition at line 108 of file [modelpc.h](#).

### 7.4.5.3 bool ModelPC::success

success Flag that true by default, but in case of error or cancelling of ProgressDialog it turns to false, which stops execution of [ModelPC::circuit](#)

Definition at line 82 of file [modelpc.h](#).

### 7.4.5.4 long ModelPC::version

version Version of the class

Definition at line 86 of file [modelpc.h](#).

### 7.4.5.5 QString ModelPC::versionString

versionString Version as string

Definition at line 90 of file [modelpc.h](#).

The documentation for this class was generated from the following files:

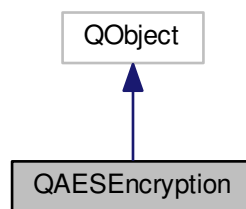
- [modelpc.h](#)
- [modelpc.cpp](#)

## 7.5 QAESEncryption Class Reference

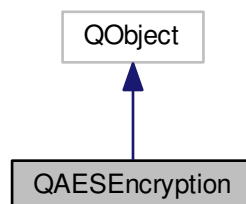
The [QAESEncryption](#) class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: <https://github.com/bricke/Qt-AES>.

```
#include <qaesencryption.h>
```

Inheritance diagram for QAESEncryption:



Collaboration diagram for QAESEncryption:



### Public Types

- enum [Aes](#) { [AES\\_128](#), [AES\\_192](#), [AES\\_256](#) }

*The Aes enum AES Level AES Levels The class supports all AES key lengths.*

- enum [Mode](#) { [ECB](#), [CBC](#), [CFB](#), [OFB](#) }

*The Mode enum AES Mode The class supports the following operating modes ECB CBC CFB OFB.*

- enum [Padding](#) { [ZERO](#), [PKCS7](#), [ISO](#) }

*The Padding enum Padding By default the padding method is ISO, however, the class supports:*

## Public Member Functions

- [QAESEncryption](#) ([QAESEncryption::Aes](#) level, [QAESEncryption::Mode](#) mode, [QAESEncryption::Padding](#) padding=[QAESEncryption::ISO](#))
- QByteArray [encode](#) (const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL)  
*encode Encodes data with AES*
- QByteArray [decode](#) (const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL)  
*decode Decodes data with AES*
- QByteArray [removePadding](#) (const QByteArray &rawText)  
*RemovePadding Removes padding.*
- QByteArray [expandKey](#) (const QByteArray &key)  
*ExpandKey Expands the key.*

## Static Public Member Functions

- static QByteArray [Crypt](#) ([QAESEncryption::Aes](#) level, [QAESEncryption::Mode](#) mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, [QAESEncryption::Padding](#) padding=[QAESEncryption::ISO](#))  
*Crypt Static encode function.*
- static QByteArray [Decrypt](#) ([QAESEncryption::Aes](#) level, [QAESEncryption::Mode](#) mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, [QAESEncryption::Padding](#) padding=[QAESEncryption::ISO](#))  
*Decrypt Static decode function.*
- static QByteArray [ExpandKey](#) ([QAESEncryption::Aes](#) level, [QAESEncryption::Mode](#) mode, const QByteArray &key)  
*ExpandKey Expands the key.*
- static QByteArray [RemovePadding](#) (const QByteArray &rawText, [QAESEncryption::Padding](#) padding)  
*RemovePadding Removes padding.*

### 7.5.1 Detailed Description

The [QAESEncryption](#) class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: <https://github.com/bricke/Qt-AES>.

#### Author

Bricke (Matteo B)

Definition at line 14 of file [qaesencryption.h](#).

### 7.5.2 Member Enumeration Documentation

#### 7.5.2.1 enum [QAESEncryption::Aes](#)

The Aes enum AES Level AES Levels The class supports all AES key lengths.

AES\_128 AES\_192 AES\_256

#### Enumerator

**AES\_128**

**AES\_192**

**AES\_256**

Definition at line 27 of file [qaesencryption.h](#).



## 7.5.2.2 enum QAESEncryption::Mode

The Mode enum AES Mode The class supports the following operating modes ECB CBC CFB OFB.

Enumerator

**ECB**  
**CBC**  
**CFB**  
**OFB**

Definition at line 40 of file [qaesencryption.h](#).

## 7.5.2.3 enum QAESEncryption::Padding

The Padding enum Padding By default the padding method is ISO, however, the class supports:

ZERO PKCS7 ISO

Enumerator

**ZERO**  
**PKCS7**  
**ISO**

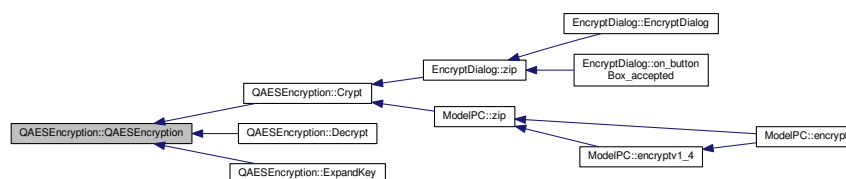
Definition at line 55 of file [qaesencryption.h](#).

## 7.5.3 Constructor &amp; Destructor Documentation

## 7.5.3.1 QAESEncryption::QAESEncryption ( QAESEncryption::Aes level, QAESEncryption::Mode mode, QAESEncryption::Padding padding = QAESEncryption::ISO )

Definition at line 67 of file [qaesencryption.cpp](#).

Here is the caller graph for this function:



## 7.5.4 Member Function Documentation

## 7.5.4.1 QByteArray QAESEncryption::Crypt ( QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &amp; rawText, const QByteArray &amp; key, const QByteArray &amp; iv = NULL, QAESEncryption::Padding padding = QAESEncryption::ISO ) [static]

Crypt Static encode function.

## Parameters

<i>level</i>	AES level of encryption
<i>mode</i>	AES mode
<i>rawText</i>	Input data
<i>key</i>	Key for encryption
<i>iv</i>	IV vector
<i>padding</i>	Padding

## Returns

Returns encrypted data

## See also

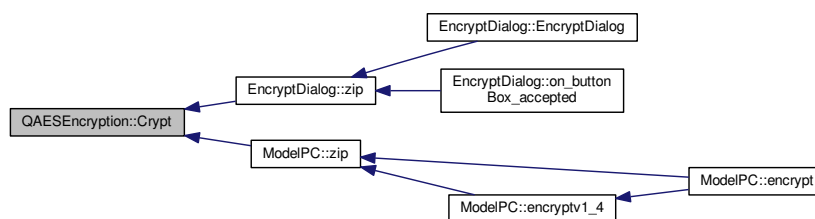
[QAESEncryption::encode](#), [QAESEncryption::Decrypt](#)

Definition at line 6 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**7.5.4.2** `QByteArray QAESEncryption::decode ( const QByteArray & rawText, const QByteArray & key, const QByteArray & iv = NULL )`

decode Decodes data with AES

## Note

Basically the non-static method of [QAESEncryption::Decrypt](#)

## Parameters

<i>rawText</i>	Input data
<i>key</i>	Key
<i>iv</i>	IV vector

## Returns

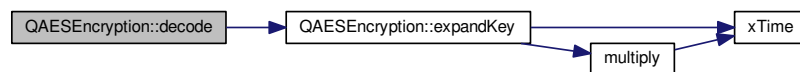
Returns decoded data

## See also

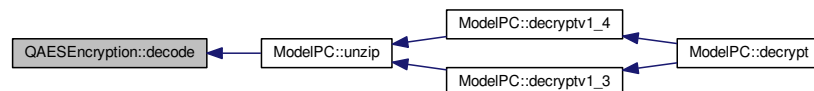
[QAESEncryption::Decrypt](#), [QAESEncryption::encode](#)

Definition at line 441 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**7.5.4.3** `QByteArray QAESEncryption::Decrypt ( QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray & rawText, const QByteArray & key, const QByteArray & iv = NULL, QAESEncryption::Padding padding = QAESEncryption::ISO ) [static]`

Decrypt Static decode function.

## Parameters

<i>level</i>	AES level of encryption
<i>mode</i>	AES mode
<i>rawText</i>	Encrypted data
<i>key</i>	Key for encrytion
<i>iv</i>	IV vector
<i>padding</i>	Padding

**Returns**

Returns Decrypted data

**See also**

[QAESEncryption::decode](#), [QAESEncryption::Crypt](#)

Definition at line 12 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



**7.5.4.4** `QByteArray QAESEncryption::encode ( const QByteArray & rawText, const QByteArray & key, const QByteArray & iv = NULL )`

encode Encodes data with AES

**Note**

Basically the non-static method of [QAESEncryption::Crypt](#)

**Parameters**

<i>rawText</i>	Input data
<i>key</i>	Key
<i>iv</i>	IV vector

**Returns**

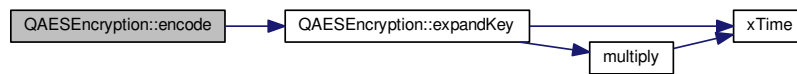
Returns encoded data

**See also**

[QAESEncryption::Crypt](#), [QAESEncryption::decode](#)

Definition at line 391 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



**7.5.4.5** `QByteArray QAESEncryption::ExpandKey ( QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray & key ) [static]`

ExpandKey Expands the key.

Parameters

<i>level</i>	AES level
<i>mode</i>	AES Mode
<i>key</i>	key

Returns

Returns expanded key (I guess)

See also

[QAESEncryption::expandKey](#)

Definition at line 18 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



**7.5.4.6** `QByteArray QAESEncryption::expandKey ( const QByteArray & key )`

ExpandKey Expands the key.

Note

Basically the non-static method of [QAESEncryption::ExpandKey](#)

## Parameters

<i>key</i>	key
------------	-----

## Returns

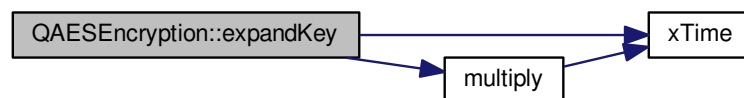
Returns expanded key (I guess)

## See also

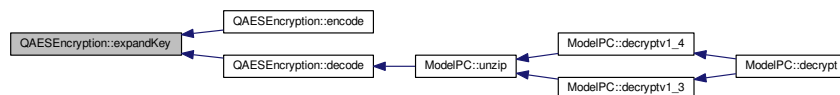
[QAESEncryption::ExpandKey](#)

Definition at line 132 of file [qaesencryption.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**7.5.4.7** `QByteArray QAESEncryption::RemovePadding ( const QByteArray & rawText, QAESEncryption::Padding padding ) [static]`

`RemovePadding` Removes padding.

## Parameters

<i>rawText</i>	Input data
<i>padding</i>	Padding

## Returns

Returns data with removed padding (I guess)

See also

[QAESEncryption::removePadding](#)

Definition at line 23 of file [qaesencryption.cpp](#).

#### 7.5.4.8 QByteArray QAESEncryption::removePadding ( const QByteArray & *rawText* )

RemovePadding Removes padding.

Note

Basically the non-static method of [QAESEncryption::RemovePadding](#)

Parameters

<i>rawText</i>	Input data
----------------	------------

Returns

Returns data with removed padding (I guess)

See also

[QAESEncryption::RemovePadding](#)

Definition at line 490 of file [qaesencryption.cpp](#).

The documentation for this class was generated from the following files:

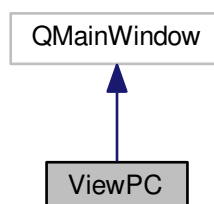
- [qaesencryption.h](#)
- [qaesencryption.cpp](#)

## 7.6 ViewPC Class Reference

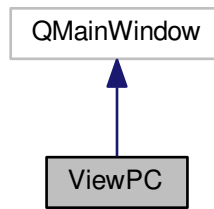
The [ViewPC](#) class View layer of the app. Controls [EncryptDialog](#) and ProgressDialog.

```
#include <viewpc.h>
```

Inheritance diagram for ViewPC:



Collaboration diagram for ViewPC:



## Public Slots

- void [alert](#) (QString message, bool isWarning=false)  
*ViewPC::alert* Slot to create *QMessageBox* with message.
- void [saveData](#) (QByteArray Edata)  
*ViewPC::saveData* Slot to be called to save data using *QFileDialog*.
- void [saveImage](#) (QImage \*image)  
*ViewPC::saveImage* Slot to be called to save image using *QFileDialog*.
- void [setProgress](#) (int val)  
*ViewPC::setProgress* Slot to set the value of the *ProgressDialog* (*ViewPC::dialog*).
- void [abortCircuit](#) ()  
*ViewPC::abortCircuit* Slot to close *ProgressDialog* (*ViewPC::dialog*)
- void [setEncryptMode](#) (bool encr)  
*ViewPC::setEncryptMode* Set the encrypt mode (*ViewPC::isEncrypt*)
- void [setVersion](#) (QString version)  
*ViewPC::setVersion* Set the version of the app from *ControllerPC*.

## Signals

- void [encrypt](#) (QByteArray data, QImage \*image, int mode, QString key)  
*encrypt* Signal calling *ModelPC::encrypt*
- void [inject](#) (QByteArray data, QImage \*image, int mode, int bitsUsed)  
*inject* Signal calling *ModelPC::inject*
- void [decrypt](#) (QImage \*\_image, QString key, int mode)  
*decrypt* Signal calling *ModelPC::decrypt*
- void [abortModel](#) ()  
*abortModel* Signal calling to stop *ModelPC::circuit*
- void [setJPHSDir](#) (QString dir)  
*setJPHSPath* Sets the default JPHS directory

## Public Member Functions

- [ViewPC](#) (QWidget \*parent=nullptr)
- [~ViewPC](#) ()  
*ViewPC::~~ViewPC* Simple destructor for this layer.



## Public Attributes

- QProgressDialog \* [dialog](#)  
*dialog ProgressDialog used.*
- bool [progressDialogClosed](#)  
*progressDialogClosed Flag, if dialog is closed.*
- QJsonObject [errorsDict](#)  
*errorsDict Json object for errors dictionary*

## Protected Slots

- void [on\\_fileButton\\_clicked](#) ()  
*ViewPC::on\_fileButton\_clicked Slot to be called, when according button is pressed.*
- void [on\\_startButton\\_clicked](#) ()  
*ViewPC::on\_startButton\_clicked Slot to be called, when Start Button is pressed.*
- void [on\\_actionAbout\\_triggered](#) ()  
*ViewPC::on\_actionAbout\_triggered Opens about page.*
- void [on\\_actionHelp\\_triggered](#) ()  
*ViewPC::on\_actionHelp\_triggered Opens online documentation.*

## Protected Member Functions

- QString [requestKey](#) ()  
*ViewPC::requestKey Request keyphrase from user using InputDialog.*

### 7.6.1 Detailed Description

The [ViewPC](#) class View layer of the app. Controls [EncryptDialog](#) and ProgressDialog.

See also

[ControllerPC](#), [ModelPC](#), [EncryptDialog](#)

Definition at line 35 of file [viewpc.h](#).

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 ViewPC::ViewPC ( QWidget \* *parent* = nullptr ) [explicit]

Definition at line 4 of file [viewpc.cpp](#).

Here is the call graph for this function:

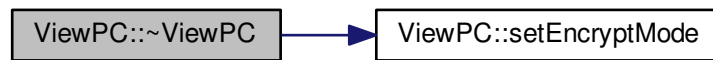


### 7.6.2.2 ViewPC::~~ViewPC ( )

[ViewPC::~~ViewPC](#) Simple destructor for this layer.

Definition at line 29 of file [viewpc.cpp](#).

Here is the call graph for this function:



## 7.6.3 Member Function Documentation

### 7.6.3.1 void ViewPC::abortCircuit ( ) [slot]

[ViewPC::abortCircuit](#) Slot to close ProgressDialog ([ViewPC::dialog](#))

Definition at line 228 of file [viewpc.cpp](#).

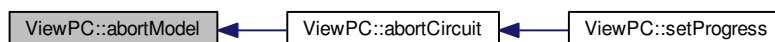
Here is the caller graph for this function:



### 7.6.3.2 void ViewPC::abortModel ( ) [signal]

`abortModel` Signal calling to stop [ModelPC::circuit](#)

Here is the caller graph for this function:



### 7.6.3.3 void ViewPC::alert ( QString message, bool isWarning = false ) [slot]

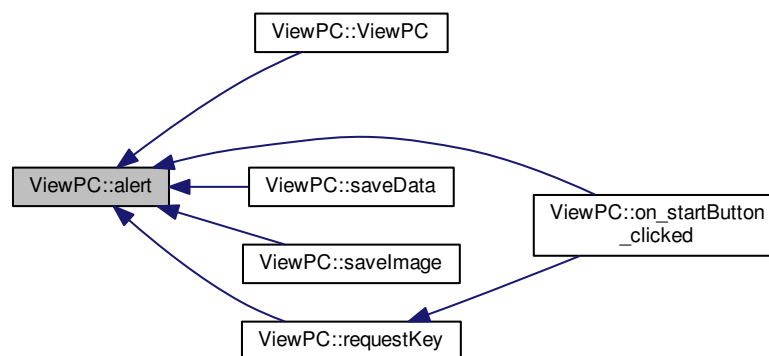
[ViewPC::alert](#) Slot to create QMessageBox with message.

## Parameters

<i>message</i>	Message to be shown
<i>isWarning</i>	Flag, if message is critical.

Definition at line 142 of file [viewpc.cpp](#).

Here is the caller graph for this function:



**7.6.3.4** `void ViewPC::decrypt ( QImage *_image, QString key, int mode )` [signal]

decrypt Signal calling [ModelPC::decrypt](#)

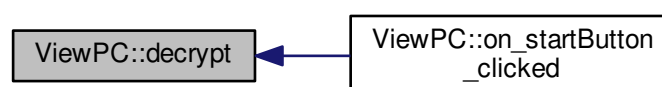
## Parameters

<i>_image</i>	Image for decryption
<i>key</i>	encryption key
<i>mode</i>	Mode of decryption

## See also

[ModelPC::decrypt](#), [ModelPC::CryptMode](#)

Here is the caller graph for this function:



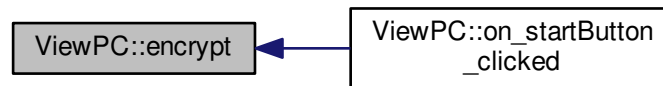
7.6.3.5 void ViewPC::encrypt ( QByteArray *data*, QImage \* *image*, int *mode*, QString *key* ) [signal]

encrypt Signal calling [ModelPC::encrypt](#)

#### Parameters

<i>data</i>	Data to write
<i>image</i>	Image to be encrypted into
<i>mode</i>	Mode of encryption
<i>key</i>	Key of encryption

Here is the caller graph for this function:



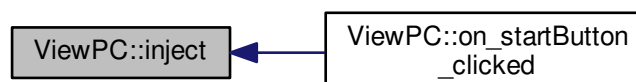
7.6.3.6 void ViewPC::inject ( QByteArray *data*, QImage \* *image*, int *mode*, int *bitsUsed* ) [signal]

inject Signal calling [ModelPC::inject](#)

#### Parameters

<i>data</i>	Data to write
<i>image</i>	Image to be encrypted into.
<i>mode</i>	Mode of encryption
<i>bitsUsed</i>	Bits used per byte

Here is the caller graph for this function:



7.6.3.7 `void ViewPC::on_actionAbout_triggered ( ) [protected],[slot]`

[ViewPC::on\\_actionAbout\\_triggered](#) Opens about page.

Definition at line 285 of file [viewpc.cpp](#).

Here is the call graph for this function:



7.6.3.8 `void ViewPC::on_actionHelp_triggered ( ) [protected],[slot]`

[ViewPC::on\\_actionHelp\\_triggered](#) Opens online documentation.

Definition at line 295 of file [viewpc.cpp](#).

7.6.3.9 `void ViewPC::on_fileButton_clicked ( ) [protected],[slot]`

[ViewPC::on\\_fileButton\\_clicked](#) Slot to be called, when according button is pressed.

Definition at line 48 of file [viewpc.cpp](#).

7.6.3.10 `void ViewPC::on_startButton_clicked ( ) [protected],[slot]`

[ViewPC::on\\_startButton\\_clicked](#) Slot to be called, when Start Button is pressed.

## 7.6.4 Encrypting

If Encrypting mode is active the data from text browser or from file from file selector will be opened and checked in size.

### Note

File size limit is 16MB

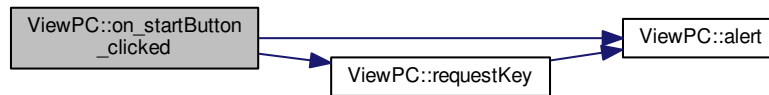
Then the [EncryptDialog](#) opens and image and key is selected. Then the [ViewPC::encrypt](#) signal is called to start [ModelPC::encrypt](#)

### 7.6.5 Decrypting

Else, the image from file selector is transmitted to [ModelPC::decrypt](#)

Definition at line 70 of file [viewpc.cpp](#).

Here is the call graph for this function:



#### 7.6.5.1 QString ViewPC::requestKey ( ) [protected]

[ViewPC::requestKey](#) Request keyphrase from user using InputDialog.

Returns

Returns keyphrase

Definition at line 265 of file [viewpc.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.6.5.2 void ViewPC::saveData ( QByteArray Edata ) [slot]

[ViewPC::saveData](#) Slot to be called to save data using QFileDialog.

## Parameters

<i>Edata</i>	Encrypted data to be saved.
--------------	-----------------------------

## See also

[ModelPC::encrypt](#)

Definition at line 163 of file [viewpc.cpp](#).

Here is the call graph for this function:



### 7.6.5.3 void ViewPC::saveImage ( QImage \* *image* ) [slot]

[ViewPC::saveImage](#) Slot to be called to save image using QFileDialog.

## Parameters

<i>image</i>	Image to be saved.
--------------	--------------------

## See also

[ModelPC::decrypt](#)

Definition at line 184 of file [viewpc.cpp](#).

Here is the call graph for this function:



### 7.6.5.4 void ViewPC::setEncryptMode ( bool *encr* ) [slot]

[ViewPC::setEncryptMode](#) Set the encrpt mode (`ViewPC::isEncrypt`)

## Parameters

<i>encr</i>	= isEncrypt, true if encrypting, false if decrypting
-------------	--

Definition at line 241 of file [viewpc.cpp](#).

Here is the caller graph for this function:



7.6.5.5 `void ViewPC::setJPHSDir ( QString dir ) [signal]`

setJPHSPath Sets the default JPHS directory

## Parameters

<i>dir</i>	Directory
------------	-----------

Here is the caller graph for this function:



7.6.5.6 `void ViewPC::setProgress ( int val ) [slot]`

[ViewPC::setProgress](#) Slot to set the value of the ProgressDialog ([ViewPC::dialog](#)).

## Parameters

<i>val</i>	New value of the dialog. If -1, creates ProgressDialog, if 101 closes the dialog.
------------	---



See also

[ViewPC::abortCircuit\(\)](#), [ModelPC::setProgress\(\)](#)

Definition at line 202 of file [viewpc.cpp](#).

Here is the call graph for this function:



**7.6.5.7** `void ViewPC::setVersion ( QString version ) [slot]`

[ViewPC::setVersion](#) Set the version of the app from [ControllerPC](#).

Parameters

<i>version</i>	Version as QString
----------------	--------------------

Definition at line 256 of file [viewpc.cpp](#).

Here is the caller graph for this function:



## 7.6.6 Member Data Documentation

**7.6.6.1** `QProgressDialog* ViewPC::dialog`

`dialog` ProgressDialog used.

See also

[ViewPC::setProgress](#), [ViewPC::cancel](#), [ModelPC::setProgress](#)

Definition at line 108 of file [viewpc.h](#).

#### 7.6.6.2 QJsonObject ViewPC::errorsDict

errorsDict Json object for errors dictionary

Definition at line 117 of file [viewpc.h](#).

#### 7.6.6.3 bool ViewPC::progressDialogClosed

progressDialogClosed Flag, if dialog is closed.

See also

[ViewPC::abortCircuit](#), [ViewPC::setProgress](#)

Definition at line 113 of file [viewpc.h](#).

The documentation for this class was generated from the following files:

- [viewpc.h](#)
- [viewpc.cpp](#)

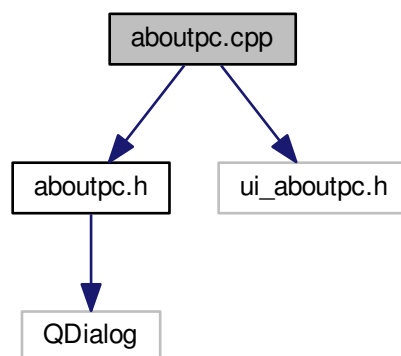
## Chapter 8

# File Documentation

### 8.1 aboutpc.cpp File Reference

```
#include "aboutpc.h"  
#include "ui_aboutpc.h"
```

Include dependency graph for aboutpc.cpp:



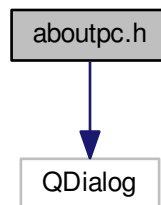
### 8.2 aboutpc.cpp

```
00001 #include "aboutpc.h"  
00002 #include "ui_aboutpc.h"  
00003  
00004 AboutPC::AboutPC(QWidget *parent) :  
00005     QDialog(parent),  
00006     ui(new Ui::AboutPC)  
00007 {  
00008     ui->setupUi(this);  
00009 }  
00010  
00011 AboutPC::~AboutPC()  
00012 {  
00013     delete ui;  
00014 }  
00019 void AboutPC::setVersion(QString version)  
00020 {  
00021     ui->versionLabel->setText("Version " + version);  
00022 }
```

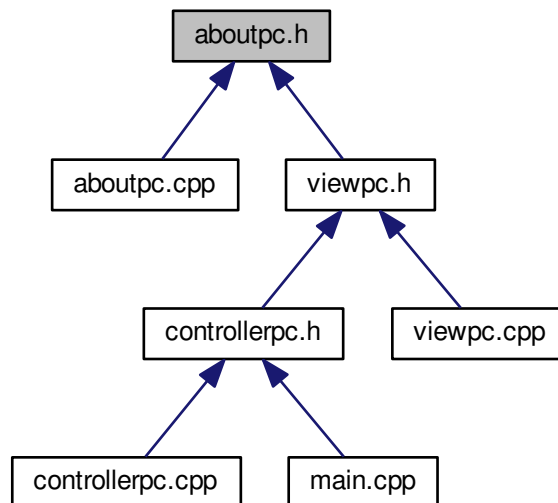
### 8.3 aboutpc.h File Reference

```
#include <QDialog>
```

Include dependency graph for aboutpc.h:



This graph shows which files directly or indirectly include this file:



#### Classes

- class [AboutPC](#)

The [AboutPC](#) class The About Page dialog.

#### Namespaces

- [Ui](#)

## 8.4 aboutpc.h

```

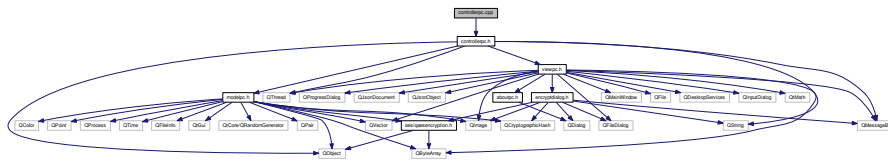
00001 #ifndef ABOUTPC_H
00002 #define ABOUTPC_H
00003
00004 #include <QDialog>
00005
00006 namespace Ui {
00007     class AboutPC;
00008 }
00012 class AboutPC : public QDialog
00013 {
00014     Q_OBJECT
00015
00016 public:
00017     explicit AboutPC(QWidget *parent = 0);
00018     ~AboutPC();
00019     void setVersion(QString version);
00020
00021 private:
00022     Ui::AboutPC *ui;
00023 };
00024
00025 #endif // ABOUTPC_H

```

## 8.5 controllerpc.cpp File Reference

```
#include "controllerpc.h"
```

Include dependency graph for controllerpc.cpp:



## 8.6 controllerpc.cpp

```

00001 #include "controllerpc.h"
00002
00009 ControllerPC::ControllerPC()
00010 {
00011     // Layer creation
00012     view = new ViewPC();
00013     model = new ModelPC();
00014     QThread * modelThread = new QThread();
00015     model->moveToThread(modelThread);
00016     modelThread->start();
00017
00018     view->setVersion(model->versionString);
00019     view->show();
00020
00021     // Layers Connection
00022     connect(view, SIGNAL(encrypt(QByteArray, QImage*, int, QString)), model, SLOT(encrypt(QByteArray,
00023 QImage*, int, QString)));
00024     connect(view, SIGNAL(inject(QByteArray,QImage*,int, int)), model, SLOT(inject(QByteArray,QImage*, int,
00025 int)));
00026     connect(view, SIGNAL(decrypt(QImage*,QString,int)), model, SLOT(decrypt(QImage*, QString, int)));
00027     connect(view, SIGNAL(abortModel()), this, SLOT(abortCircuit()));
00028     connect(view, SIGNAL(setJPHSDir(QString)), this, SLOT(setJPHSDir(QString)));
00029
00030     connect(model, SIGNAL(alertView(QString,bool)), view, SLOT(alert(QString,bool)));
00031     connect(model, SIGNAL(saveData(QByteArray)), view, SLOT(saveData(QByteArray)));
00032     connect(model, SIGNAL(saveImage(QImage*)), view, SLOT(saveImage(QImage*)));
00033     connect(model, SIGNAL(setProgress(int)), view, SLOT(setProgress(int)));
00034 }
00037 void ControllerPC::abortCircuit()

```



## 8.8 controllerpc.h

```

00001 #ifndef CONTROLLERPC_H
00002 #define CONTROLLERPC_H
00003
00004 #include <QObject>
00005 #include <QString>
00006 #include <QThread>
00007 #include <QMessageBox>
00008
00009 #include <modelpc.h>
00010 #include <viewpc.h>
00020 class ControllerPC : public QObject
00021 {
00022     Q_OBJECT
00023 public:
00024     ControllerPC();
00028     long int version;
00032     QString versionString;
00033 public slots:
00034     void abortCircuit();
00035     void setJPHSDir(QString dir);
00036 private:
00037     ViewPC * view;
00038     ModelPC * model;
00039 };
00040
00041 #endif // CONTROLLERPC_H

```

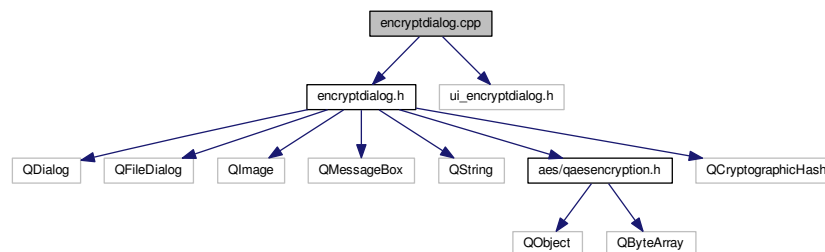
## 8.9 encryptdialog.cpp File Reference

```

#include "encryptdialog.h"
#include "ui_encryptdialog.h"

```

Include dependency graph for encryptdialog.cpp:



## 8.10 encryptdialog.cpp

```

00001 #include "encryptdialog.h"
00002 #include "ui_encryptdialog.h"
00009 EncryptDialog::EncryptDialog(QByteArray _data, QWidget *parent) :
00010     QDialog(parent),
00011     ui(new Ui::EncryptDialog)
00012 {
00013     ui->setupUi(this);
00014     data = _data;
00015     success = false;
00016     // UI setup
00017     ui->totalBytes->setText(QString::number(data.size()));
00018     key = "";
00019     compr_data = zip();
00020     long long int compr_data_size = compr_data.size();
00021     ui->zippedBytes->setText(QString::number(compr_data_size));
00022     goodPercentage = false;

```

```

00023     bitsUsed = 8;
00024 }
00025
00026 EncryptDialog::~EncryptDialog()
00027 {
00028     delete ui;
00029 }
00030
00031 void EncryptDialog::alert(QString text)
00032 {
00033     QMessageBox t;
00034     t.setWindowTitle("Message");
00035     t.setIcon(QMessageBox::Warning);
00036     t.setWindowIcon(QIcon(":/mail.png"));
00037     t.setText(text);
00038     t.exec();
00039 }
00040 QByteArray EncryptDialog::zip()
00041 {
00042     // Zip
00043     QByteArray c_data = qCompress(data, 9);
00044     // Encryption
00045     QByteArray hashKey = QCryptographicHash::hash(key.toUtf8(), QCryptographicHash::Sha256);
00046     return QAESEncryption::Crypt(QAESEncryption::AES_256,
00047     QAESEncryption::ECB, c_data, hashKey);
00048 }
00049 void EncryptDialog::on_fileButton_clicked()
00050 {
00051     // Selet file
00052     inputFileName = QFileDialog::getOpenFileName(this, tr("Open File"), "/", tr("Images (*.png
00053     *.xpm *.jpg *.jpeg)"));
00054     ui->fileLabel->setText(inputFileName);
00055     // Open image
00056     QImage img(inputFileName);
00057     image = img;
00058     // Get size
00059     size = img.width() * img.height();
00060     // UI setup
00061     long long int compr_data_size = compr_data.size();
00062     ui->zippedBytes->setText(QString::number(compr_data_size));
00063     if(inputFileName.isEmpty()) {
00064         ui->percentage->setText("");
00065         return;
00066     }
00067     double perc = (compr_data_size + 14) * 100 / (size * 3) * bitsUsed / 8;
00068     ui->percentage->setText(QString::number(perc) + "%");
00069     goodPercentage = perc < 70;
00070 }
00071 void EncryptDialog::on_buttonBox_accepted()
00072 {
00073     if(!goodPercentage) {
00074         alert("Your encoding percentage is over 70% which is a bit ambiguous :(");
00075         success = false;
00076         return;
00077     }
00078     // Final zip
00079     key = ui->keyLine->text();
00080     compr_data = zip();
00081     success = true;
00082     close();
00083 }
00084 void EncryptDialog::on_buttonBox_rejected()
00085 {
00086     success = false;
00087     close();
00088 }
00089 void EncryptDialog::on_bitsSlider_valueChanged(int
00090 value)
00091 {
00092     bitsUsed = value;
00093     ui->bitsUsedLbl->setText(QString::number(value));
00094     if(ui->percentage->text().isEmpty())
00095         return;
00096     double perc = (compr_data.size() + 14) * 100 / (size * 3) * 8 /
00097     bitsUsed;
00098     ui->percentage->setText(QString::number(perc) + "%");
00099 }

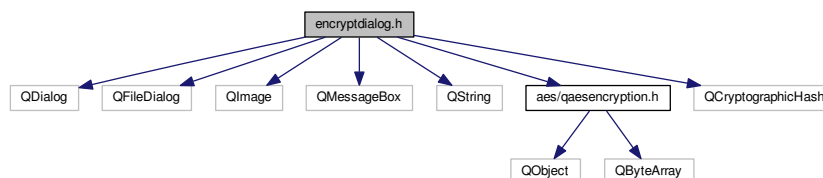
```

## 8.11 encryptdialog.h File Reference

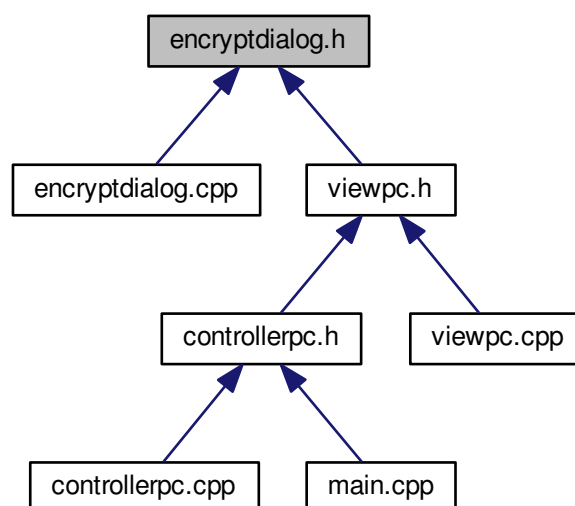
```
#include <QDialog>
```



```
#include <QFileDialog>
#include <QImage>
#include <QMessageBox>
#include <QString>
#include <aes/qaesencryption.h>
#include <QCryptographicHash>
Include dependency graph for encryptdialog.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [EncryptDialog](#)

The [EncryptDialog](#) class Class to get the image and key to store secret info.

## Namespaces

- [Ui](#)

## 8.12 encryptdialog.h

```

00001 #ifndef ENCRYPTDIALOG_H
00002 #define ENCRYPTDIALOG_H
00003
00004 #include <QDialog>
00005 #include <QFileDialog>
00006 #include <QImage>
00007 #include <QMessageBox>
00008 #include <QString>
00009
00010 #include <aes/qaesencryption.h>
00011 #include <QCryptographicHash>
00012
00013 namespace Ui {
00014 class EncryptDialog;
00015 }
00021 class EncryptDialog : public QDialog
00022 {
00023     Q_OBJECT
00024
00025 public:
00026     explicit EncryptDialog(QByteArray _data, QWidget *parent = 0);
00027     ~EncryptDialog();
00028
00029 public slots:
00030     void on_fileButton_clicked();
00031
00032     void on_buttonBox_accepted();
00033
00034     void on_buttonBox_rejected();
00035
00036     void on_bitsSlider_valueChanged(int value);
00037
00038 public:
00042     QByteArray data;
00046     bool success;
00050     QByteArray compr_data;
00054     QString inputFileName;
00058     long long int size;
00062     QString key;
00066     bool goodPercentage;
00070     int val;
00075     int bitsUsed;
00079     QImage image;
00080     QByteArray zip();
00081 private:
00082     Ui::EncryptDialog *ui;
00083     void alert(QString text);
00084 };
00085
00086 #endif // ENCRYPTDIALOG_H

```

## 8.13 ErrorsDict.json File Reference

## 8.14 ErrorsDict.json

```

00001 {
00002     "nodata": "No data given!",
00003     "nullimage": "Image not valid!",
00004     "bigkey": "Key is too big, max is 255 bytes!",
00005     "muchdata": "Too much data for this image",
00006     "wrongmode": "Incorrect mode selected",
00007     "wrongimage": "Image wasn't encrypted by this app or is damaged!",
00008     "noreaddata": "Read data is empty!",
00009     "savefilefail": "Cannot save the file!",
00010     "bitsBufferFail": "Something went very wrong! Error code: bitsBuffer",
00011     "nojphs": "JPBS not installed, installation required!\nSee Menu -> Configure -> JPBS directory",
00012     "fail_hash": "Invalid keyphrase"
00013 }

```

## 8.15 ErrorsDictSetup.py File Reference

### Namespaces

- [ErrorsDictSetup](#)



## 8.18 main.cpp

```

00001 #include "controllerpc.h"
00002 #include <QApplication>
00116 int main(int argc, char *argv[])
00117 {
00118     QApplication a(argc, argv);
00119     ControllerPC w;
00120
00121     return a.exec();
00122 }

```

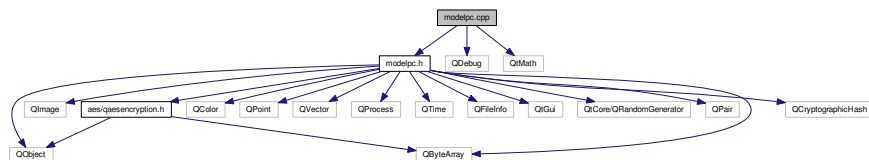
## 8.19 modelpc.cpp File Reference

```

#include "modelpc.h"
#include <QDebug>
#include <QtMath>

```

Include dependency graph for modelpc.cpp:



## 8.20 modelpc.cpp

```

00001 #include "modelpc.h"
00002 #include <QDebug>
00003 #include <QtMath>
00009 ModelPC::ModelPC()
00010 {
00011     // Version control
00012     versionString = "1.4.0.dev-alpha.4";
00013
00014     auto ver = versionString.split(".");
00015     version = ver[0].toInt() * qPow(2, 16) + ver[1].toInt() * qPow(2, 8) + ver[2].toInt();
00016
00017     ver_byte = bytes(ver[0].toInt()) +
00018               bytes(ver[1].toInt()) +
00019               bytes(ver[2].toInt());
00020     // Random seed
00021     qsrand(randSeed());
00022 }
00023
00024 QImage *ModelPC::Encrypt(QByteArray data, QImage *image, int _mode, QString
key, int _bitsUsed, QString *_error)
00025 {
00026     return ModelPC().encrypt(data, image, _mode, key, _bitsUsed, _error);
00027 }
00028
00029 QImage *ModelPC::Inject(QByteArray encr_data, QImage *image, int _mode, int _bitsUsed,
QString *_error)
00030 {
00031     return ModelPC().inject(encr_data, image, _mode, _bitsUsed, _error);
00032 }
00033
00034 QByteArray ModelPC::Decrypt(QImage *image, QString key, int _mode, QString *_error)
00035 {
00036     return ModelPC().decrypt(image, key, _mode, _error);
00037 }
00051 QImage * ModelPC::encrypt(QByteArray data, QImage * image, int _mode, QString
key, int _bitsUsed, QString *_error)
00052 {
00053     success = true;

```

```

00054     CryptMode mode = CryptMode(_mode);
00055     // Error management
00056     if(_error == nullptr)
00057         _error = new QString();
00058     *_error = "ok";
00059     error = _error;
00060
00061     if(data == nullptr || data.isEmpty()) {
00062         fail("nodata");
00063         return nullptr;
00064     }
00065     if(data.size() > pow(2, 24)) {
00066         fail("muchdata");
00067         return nullptr;
00068     }
00069     if(image == nullptr || image->isNull()) {
00070         fail("nullimage");
00071         return nullptr;
00072     }
00073     if(image->width() * image->height() > pow(10, 9)) {
00074         fail("bigimage");
00075         return nullptr;
00076     }
00077     if(_bitsUsed < 1 || _bitsUsed > 8) {
00078         fail("bitsWrong");
00079         return nullptr;
00080     }
00081     if(key == nullptr || key.isEmpty()) {
00082         fail("no_key");
00083         return nullptr;
00084     }
00085     else if(key.size() > 255) {
00086         fail("bigkey");
00087         return nullptr;
00088     }
00089     if(mode == CryptMode::NotDefined) {
00090         fail("undefined_mode");
00091         return nullptr;
00092     }
00093     long long usedBytes = data.size() + 14 + key.size();
00094     long long size = image->width() * image->height();
00095     if(usedBytes * 100 / (size * 3) * 8 / _bitsUsed > 70) {
00096         fail("bigdata");
00097         return nullptr;
00098     }
00099
00100     switch(mode)
00101     {
00102     case v1_3:
00103     {
00104         QByteArray zipped_data = zip(data, key.toUtf8());
00105         QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
00106         QByteArray encr_data = hash + zipped_data;
00107         if(*error == "ok")
00108             return inject(encr_data, image, _mode, _bitsUsed, error);
00109         else
00110             return nullptr;
00111         break;
00112     }
00113     case v1_4:
00114         bitsUsed = _bitsUsed;
00115         encryptv1_4(image, data, key);
00116         emit saveImage(image);
00117         return image;
00118         break;
00119     case jphs_mode:
00120         // TODO add jphs
00121         return nullptr;
00122         break;
00123     default:
00124         fail("wrongmode");
00125         return nullptr;
00126     }
00127 }
00128
00139 QImage * ModelPC::inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed,
00140     QString *_error)
00141 {
00142     success = true;
00143     CryptMode mode = CryptMode(_mode);
00144     // Error management
00145     if(_error == nullptr)
00146         _error = new QString();
00147     *_error = "ok";
00148     error = _error;
00149     bitsUsed = _bitsUsed;

```

```

00150
00151     if(encr_data == nullptr || encr_data.isEmpty()) {
00152         fail("nodata");
00153         return nullptr;
00154     }
00155     if(encr_data.size() > pow(2, 24)) {
00156         fail("muchdata");
00157         return nullptr;
00158     }
00159     if(image == nullptr || image->isNull()) {
00160         fail("nullimage");
00161         return nullptr;
00162     }
00163     if(image->width() * image->height() > pow(10, 9)) {
00164         fail("bigimage");
00165         return nullptr;
00166     }
00167     if(_bitsUsed < 1 || _bitsUsed > 8) {
00168         fail("bitsWrong");
00169         return nullptr;
00170     }
00171     if(mode == CryptMode::NotDefined) {
00172         fail("undefined_mode");
00173         return nullptr;
00174     }
00175
00176     encr_data = ver_byte + encr_data;
00177     long long int countBytes = encr_data.size();
00178     switch(mode)
00179     {
00180     case v1_3:
00181         circuit(image, &encr_data, countBytes);
00182         break;
00183     case jphs_mode:
00184         jphs(image, &encr_data);
00185         break;
00186     case v1_4:
00187         fail("inject-v1.4");
00188         return nullptr;
00189         break;
00190     default:
00191         fail("wrongmode");
00192         return nullptr;
00193     }
00194
00195     // Saving
00196     if(success) {
00197         emit saveImage(image);
00198         return image;
00199     }
00200     else
00201         return nullptr;
00202 }
00212 QByteArray ModelPC::decrypt(QImage * image, QString key, int _mode, QString *_error)
00213 {
00214     success = true;
00215     CryptMode mode = CryptMode(_mode);
00216     // Error management
00217     if(_error == nullptr)
00218         _error = new QString();
00219     *_error = "ok";
00220     error = _error;
00221     if(image == nullptr || image->isNull()) {
00222         fail("nullimage");
00223         return nullptr;
00224     }
00225     if(image->width() * image->height() > pow(10, 9)) {
00226         fail("bigimage");
00227         return nullptr;
00228     }
00229     if(key == nullptr || key.isEmpty()) {
00230         fail("no_key");
00231         return nullptr;
00232     }
00233     QByteArray result;
00234
00235     switch (mode) {
00236     case v1_3:
00237         result = decryptv1_3(image, key);
00238         break;
00239     case v1_4:
00240         result = decryptv1_4(image, key);
00241         break;
00242     case jphs_mode:
00243         // TODO add jphs support
00244         break;
00245     case NotDefined:

```

```

00246         isTry = true;
00247
00248         // v1_3
00249         result = decryptv1_3(new QImage(*image), key);
00250         if(success) {
00251             isTry = false;
00252             break;
00253         }
00254         success = true;
00255
00256         // v1_4
00257         result = decryptv1_4(image, key);
00258         if(success) {
00259             isTry = false;
00260             break;
00261         }
00262         success = true;
00263
00264         // TODO add jphs support
00265
00266         isTry = false;
00267         fail("all_modes_fail");
00268         return nullptr;
00269     break;
00270     default:
00271         // For invalid modes
00272         fail("wrongmode");
00273         return nullptr;
00274     }
00275     if(*error == "ok")
00276         emit saveData(result);
00277     return result;
00278 }
00283 void ModelPC::fail(QString message)
00284 {
00285     success = false;
00286     if(!isTry) {
00287         *error = message;
00288         alert(message, true);
00289         emit setProgress(101);
00290     }
00291     qDebug() << "[Debug] !!! fail() - " << message;
00292 }
00298 void ModelPC::jphs(QImage *image, QByteArray *data)
00299 {
00300     // Under Development
00301     return;
00302
00303     // Dead code
00304
00305     success = true;
00306     bool isEncrypt = !data->isEmpty();
00307     QString targetEXE = defaultJPHSDir + (isEncrypt ? "/jphide.exe" : "/jpseek.exe");
00308     if(!fileExists(targetEXE))
00309     {
00310         fail("nojphs");
00311         return;
00312     }
00313
00314     QString randomFileName = defaultJPHSDir + "/";
00315     qsrand(randSeed());
00316     for(int i = 0; i < 10; i++)
00317         randomFileName.append(97 + qrand() % 25);
00318     image->save(randomFileName + ".jpg");
00319     if(isEncrypt) {
00320         QFile file(randomFileName + ".pc");
00321         if(!file.open(QFile::WriteOnly)) {
00322             fail("save_file_fail");
00323             return;
00324         }
00325         file.write(*data);
00326         file.close();
00327
00328         QStringList args;
00329         args << (randomFileName + ".jpg") << (randomFileName + "_out.jpg") << (randomFileName + ".pc");
00330         QProcess prog(this);
00331         prog.start(targetEXE, args);
00332         prog.waitForStarted();
00333         prog.write("test\n");
00334         prog.waitForBytesWritten();
00335         prog.write("test\n");
00336         prog.waitForBytesWritten();
00337         prog.waitForReadyRead();
00338         QByteArray bytes = prog.readAll();
00339         prog.waitForFinished();
00340         //QByteArray readData = prog.readAll();
00341         prog.close();

```

```

00342         // Cleaning - Deleting temp files
00343     }
00344 }
00345 else {
00346     }
00347 }
00348 }
00349 }
00350
00359 void ModelPC::circuit(QImage *image, QByteArray *data, long long countBytes)
00360 {
00361     // Some flags and creation of the ProgressDialog
00362     success = true;
00363     emit setProgress(-1);
00364     bool isEncrypt = !data->isEmpty();
00365
00366     // Image setup
00367     int w = image->width();
00368     int h = image->height();
00369
00370     // Visited pixels array
00371     QVector<QPoint> were;
00372     were.push_back(QPoint(0, 0));
00373     were.push_back(QPoint(0, h - 1));
00374     were.push_back(QPoint(w - 1, 0));
00375     were.push_back(QPoint(w - 1, h - 1));
00376
00377     long long int offset = 0;
00378
00379     // Pre-start Cleaning
00380     circuitData = data;
00381     circuitImage = image;
00382     circuitCountBytes = countBytes;
00383     cur = 0;
00384     bitsBuffer.clear();
00385
00386     // Writing Top-Left to Bottom-Left
00387     for(int i = 1; i < h - 1 && mustGoOn(isEncrypt); i++) {
00388         QPoint pos(0, i);
00389         processPixel(pos, &were, isEncrypt);
00390     }
00391     // Writing Bottom-Right to Top-Right
00392     if(mustGoOn(isEncrypt))
00393     {
00394         for(int i = h - 2; i >= 1 && mustGoOn(isEncrypt); i--){
00395             QPoint pos(w - 1, i);
00396             processPixel(pos, &were, isEncrypt);
00397         }
00398     }
00399     // Main cycle
00400     // Strong is considered as actual corner pixel and weak as pixel near it like (1, 0) or (0, 1)
00401     while(mustGoOn(isEncrypt))
00402     {
00403         // Strong Top-Right to Strong Bottom-Right
00404         for(int i = offset; i < h - offset && mustGoOn(isEncrypt); i++){
00405             QPoint pos(w - offset - 2, i);
00406             processPixel(pos, &were, isEncrypt);
00407         }
00408         // Strong Top-Left to Weak Top-Right
00409         for(int i = offset + 1; i < w - offset - 2 && mustGoOn(isEncrypt); i++){
00410             QPoint pos(i, offset);
00411             processPixel(pos, &were, isEncrypt);
00412         }
00413         // Weak Bottom-Right to Weak Bottom-Left
00414         for(int i = w - 3 - offset; i >= offset + 2 && mustGoOn(isEncrypt); i--){
00415             QPoint pos(i, h - offset - 1);
00416             processPixel(pos, &were, isEncrypt);
00417         }
00418         // Weak Top-Left to Strong Bottom-Left
00419         for(int i = offset + 1; i < h - offset && mustGoOn(isEncrypt); i++){
00420             QPoint pos(offset + 1, i);
00421             processPixel(pos, &were, isEncrypt);
00422         }
00423         offset++;
00424     }
00425     // Extra writing
00426     if(!success)
00427         return;
00428     if(isEncrypt)
00429     {
00430         // Getting past colors
00431         QColor colUL = image->pixelColor(0, 0).toRgb();
00432         QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00433         QColor colDL = image->pixelColor(0, h - 1).toRgb();
00434         QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00435         int red = 0;
00436         int green = 0;

```



```

00437         int blue = 0;
00438
00439         // Writing Upper Left
00440         red = (colUL.red() & 224) + (countBytes >> 19);
00441         green = (colUL.green() & 224) + (countBytes >> 14) % 32;
00442         blue = (colUL.blue() & 224) + (countBytes >> 9) % 32;
00443         image->setPixelColor(0, 0, QColor(red, green, blue));
00444
00445         // Writing Upper Right
00446         red = (colUR.red() & 224) + (countBytes >> 4) % 32;
00447         green = (colUR.green() & 224) + ((countBytes % 16) << 1) + 1;
00448         blue = (colUR.blue() & 224) + 9;
00449         image->setPixelColor(w - 1, 0, QColor(red, green, blue));
00450
00451         // Getting extra bytes if left
00452         while(cur < countBytes)
00453             push(mod(circuitData->at(cur++), 8));
00454         if(bitsBuffer.size() > 20) {
00455             fail("bitsBufferFail");
00456             return;
00457         }
00458         // Getting extra data as long.
00459         long extraData = pop(-2);
00460
00461         // Writing Down Left
00462         red = (colDL.red() & 224) + (extraData >> 15);
00463         green = (colDL.green() & 224) + (extraData >> 10) % 32;
00464         blue = (colDL.blue() & 224) + (extraData >> 5) % 32;
00465         image->setPixelColor(0, h - 1, QColor(red, green, blue));
00466
00467         // Writing Down Right
00468         red = (colDR.red() & 224) + extraData % 32;
00469         green = (colDR.green() & 224);
00470         blue = (colDR.blue() & 224) + ((bitsUsed - 1) << 2) + 2;
00471         image->setPixelColor(w - 1, h - 1, QColor(red, green, blue));
00472     }
00473     else
00474     {
00475         // Read the past pixels
00476         QColor colDL = image->pixelColor(0, h - 1).toRgb();
00477         QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00478
00479         // Read extra data
00480         long extraData = ((colDL.red() % 32) << 15) + ((colDL.green() % 32) << 10);
00481         extraData += ((colDL.blue() % 32) << 5) + colDR.red() % 32;
00482
00483         // Add extra data to the bitsBuffer
00484         push(extraData, (countBytes - cur) * 8 - bitsBuffer.size());
00485
00486         // Move bits from bitsBuffer to the QByteArray
00487         while(!bitsBuffer.isEmpty())
00488             data->append(pop(8));
00489     }
00490     emit setProgress(101);
00491 }
00492
00500 void ModelPC::processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt)
00501 {
00502     if(!success)
00503         return;
00504     // Check if point was already visited
00505     if(were->contains(pos)) {
00506         fail("point_visited_twice");
00507         return;
00508     }
00509     else
00510         were->push_back(pos);
00511     if(isEncrypt)
00512     {
00513         // Make sure that there are enough bits in bitsBuffer to write
00514         while(bitsBuffer.size() < 3 * bitsUsed)
00515             push(mod(circuitData->at(cur++), 8));
00516         // Read past contains
00517         QColor pixelColor = circuitImage->pixelColor(pos);
00518         int red = pixelColor.red();
00519         int green = pixelColor.green();
00520         int blue = pixelColor.blue();
00521
00522         // Write new data in last bitsUsed pixels
00523         red += pop() - red % (int) qPow(2, bitsUsed);
00524         green += pop() - green % (int) qPow(2, bitsUsed);
00525         blue += pop() - blue % (int) qPow(2, bitsUsed);
00526
00527         circuitImage->setPixelColor(pos, QColor(red, green, blue));
00528     }
00529     else
00530     {

```

```

00531     QColor read_color = circuitImage->pixelColor(pos).toRgb();
00532     // Reading the pixel
00533     int red = read_color.red();
00534     int green = read_color.green();
00535     int blue = read_color.blue();
00536
00537     // Reading the last bitsUsed pixels
00538     red %= (int) qPow(2, bitsUsed);
00539     green %= (int) qPow(2, bitsUsed);
00540     blue %= (int) qPow(2, bitsUsed);
00541
00542     // Getting the data in the bitsBuffer.
00543     push(red);
00544     push(green);
00545     push(blue);
00546
00547     // Getting data to QByteArray
00548     while(bitsBuffer.size() >= 8) {
00549         circuitData->append(pop(8));
00550         cur++;
00551     }
00552 }
00553 emit setProgress(100 * cur / circuitCountBytes);
00554 }
00560 void ModelPC::encryptv1_4(QImage *image, QByteArray data, QString
key)
00561 {
00562     if(data.size() + 98 > image->height() * image->width() * 3) {
00563         fail("bigdata");
00564         return;
00565     }
00566     QTime st = QTime::currentTime();
00567     QByteArray rand_master = GetRandomBytes(32);
00568     QByteArray pass = QCryptographicHash::hash(key.toUtf8() + rand_master + QByteArray("hi"),
QCryptographicHash::Sha3_384);
00569     QByteArray noise = GetRandomBytes(data.size() / 10 + 32);
00570     QByteArray bytes_key = GetRandomBytes(32);
00571     QByteArray pass_rand = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_512);
00572     QByteArray zipped = zip(data, pass_rand);
00573     QByteArray heavy_data = zipped + noise;
00574
00575     QByteArray verification = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_256);
00576     QByteArray given_key = bytes_key.left(30);
00577     QByteArray heavy_data_size;
00578     // heavy_data_size is always 4 bytes as max for heavy_data is: 2^24 * 11/10 + 32 ~ 1.8 * 10^7 < 2^32
00579     long long raw_size = zipped.size();
00580     for(int i = 0; i < 4; i++) {
00581         int ch = raw_size % 256;
00582         raw_size >>= 8;
00583         heavy_data_size.push_front(ch);
00584     }
00585     QByteArray mid_data = verification + given_key + rand_master + heavy_data_size;
00586     // mid_data.size() = 32 + 30 + 32 + 4 = 98
00587     QVector <QPair<QPoint, QPair<int, int>>> *were = new QVector <QPair<QPoint, QPair<int, int>>>();
00588     emit setProgress(-1);
00589     processPixelsv1_4(image, &mid_data, key.toUtf8(), true, were);
00590     processPixelsv1_4(image, &heavy_data, pass_rand, true, were);
00591     emit setProgress(101);
00592     QTime final = QTime::currentTime();
00593     qDebug() << "[Debug] Finished encrypting in " << st.msecsTo(final) << " msecs.";
00594 }
00595
00602 QByteArray ModelPC::decryptv1_4(QImage *image, QString key)
00603 {
00604     QTime st = QTime::currentTime();
00605     QByteArray mid_data, heavy_data;
00606     QVector <QPair<QPoint, QPair<int, int>>> *were = new QVector <QPair<QPoint, QPair<int, int>>>();
00607     emit setProgress(-1);
00608     processPixelsv1_4(image, &mid_data, key.toUtf8(), false, were, 98);
00609     QByteArray verification = mid_data.left(32);
00610     QByteArray given_key = mid_data.mid(32, 32);
00611     QByteArray rand_master = mid_data.mid(62, 32);
00612     QByteArray heavy_data_size = mid_data.right(4);
00613
00614     QByteArray pass = QCryptographicHash::hash(key.toUtf8() + rand_master + QByteArray("hi"),
QCryptographicHash::Sha3_384);
00615
00616     // Guessing
00617     emit setProgress(0);
00618     QByteArray bytes_key;
00619     for(long long i = 0; i < pow(2, 16); i++) {
00620         QByteArray guess_part;
00621         long long g = i;
00622         for(int q = 0; q < 2; q++) {
00623             int ch = g % 256;
00624             g >>= 8;
00625             guess_part.push_front(ch);

```

```

00626     }
00627     emit setProgress(100 * i / pow(2, 16));
00628     QByteArray guess = given_key + guess_part;
00629     QByteArray check = QCryptographicHash::hash(pass + guess, QCryptographicHash::Sha3_256);
00630     if(check == verification) {
00631         bytes_key = guess;
00632         break;
00633     }
00634 }
00635 if(bytes_key.isEmpty()) {
00636     fail("veriffail");
00637     return nullptr;
00638 }
00639
00640 QByteArray pass_rand = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_512);
00641
00642 long long raw_size = mod(heavy_data_size[3]) +
00643     mod(heavy_data_size[2]) * pow(2, 8) +
00644     mod(heavy_data_size[1]) * pow(2, 16) +
00645     mod(heavy_data_size[0]) * pow(2, 24);
00646 emit setProgress(0);
00647 proccessPixelsvl_4(image, &heavy_data, pass_rand, false, were, raw_size);
00648 QByteArray unzipped = unzip(heavy_data, pass_rand);
00649 emit setProgress(101);
00650 QTime final = QTime::currentTime();
00651 qDebug() << "[Debug] Finished decrypting in " << st.msecsTo(final) << " msecs.";
00652 return unzipped;
00653 }
00663 void ModelPC::proccessPixelsvl_4(QImage *image, QByteArray*
    data, QByteArray key, bool isEncrypt, QVector<QPair<QPoint, QPair<int, int>>> *were, long long size
    )
00664 {
00665     long w = image->width();
00666     long h = image->height();
00667     auto seed_hex = QCryptographicHash::hash(key, QCryptographicHash::Sha3_256).toHex().left(8).toUpper();
00668     auto seed = seed_hex.toLongLong(nullptr, 16);
00669     QRandomGenerator foo(seed);
00670
00671     bitsBuffer.clear();
00672     long long left = (size == -1 ? data->size() : size) * 8;
00673     long long all = left;
00674     long cur = 0;
00675     if(isEncrypt) {
00676         while(left > 0 && success)
00677         {
00678             if(bitsBuffer.empty())
00679                 push(mod(data->at(cur++), 8);
00680             quint64 g = foo.generate64() % (w * h);
00681             long x = g % w;
00682             long y = g / w;
00683             int c = foo.generate64() % 3;
00684             int b = foo.generate64() % 24;
00685             int bit = -1;
00686             if(b < 16)
00687                 bit = 7;
00688             else if(bit < 20)
00689                 bit = 6;
00690             else if(bit < 22)
00691                 bit = 5;
00692             else if(bit < 23)
00693                 bit = 4;
00694             else if(bit < 24)
00695                 bit = 3;
00696             auto piece = qMakePair(QPoint(x, y), qMakePair(c, bit));
00697             if(were->contains(piece))
00698                 continue;
00699             were->append(piece);
00700             left--;
00701             emit setProgress(100 * (all - left) / all);
00702             int wr = pop(1);
00703             QColor pixel = image->pixelColor(piece.first);
00704             int red = pixel.red();
00705             int green = pixel.green();
00706             int blue = pixel.blue();
00707             int dif;
00708             if(c == 0)
00709                 dif = red;
00710             else if (c == 1)
00711                 dif = green;
00712             else
00713                 dif = blue;
00714             dif |= 1 << (7 - bit);
00715             dif ^= (wr ^ 1) << (7 - bit);
00716             if(c == 0)
00717                 red = dif;
00718             else if(c == 1)
00719                 green = dif;

```

```

00720         else
00721             blue = dif;
00722             image->setPixelColor(piece.first, QColor(red, green, blue));
00723     }
00724 } else {
00725     while(left > 0)
00726     {
00727         while (bitsBuffer.size() >= 8)
00728             data->push_back(pop(8));
00729         quint64 g = foo.generate64() % (w * h);
00730         long x = g % w;
00731         long y = g / w;
00732         int c = foo.generate64() % 3;
00733         int b = foo.generate64() % 24;
00734         int bit = -1;
00735         if(b < 16)
00736             bit = 7;
00737         else if(bit < 20)
00738             bit = 6;
00739         else if(bit < 22)
00740             bit = 5;
00741         else if(bit < 23)
00742             bit = 4;
00743         else if(bit < 24)
00744             bit = 3;
00745         auto piece = qMakePair(QPoint(x, y), qMakePair(c, bit));
00746         if(were->contains(piece))
00747             continue;
00748         were->append(piece);
00749         left--;
00750         emit setProgress(100 * (all - left) / all);
00751         QColor pixel = image->pixelColor(piece.first);
00752         int red = pixel.red();
00753         int green = pixel.green();
00754         int blue = pixel.blue();
00755         int dif;
00756         if(c == 0)
00757             dif = red;
00758         else if (c == 1)
00759             dif = green;
00760         else
00761             dif = blue;
00762         dif &= 1 << (7 - bit);
00763         int wr = dif != 0;
00764         push(wr, 1);
00765     }
00766     while (bitsBuffer.size() >= 8)
00767         data->push_back(pop(8));
00768 }
00769 }
00770
00771 QByteArray ModelPC::decryptv1_3(QImage *image, QString key)
00772 {
00773     // Image opening
00774     int w = image->width();
00775     int h = image->height();
00776
00777     // Getting corner pixels
00778     QColor colUL = image->pixelColor(0, 0).toRgb();
00779     QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00780     QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00781
00782     // Getting verification code
00783     int verifCode = (((colUR.green() % 2) << 5) + colUR.blue() % 32) << 2;
00784     verifCode += colDR.blue() % 4;
00785     if(verifCode != 166){
00786         fail("veriffail");
00787         return nullptr;
00788     }
00789     // Getting number of bytes
00790     long long int countBytes = (colUL.blue() % 32 + ((colUL.green() % 32) << 5) + ((colUL.red() % 32) << 10)
00791 ) << 9;
00792     countBytes += ((colUR.red() % 32) << 4) + (colUR.green() >> 1) % 16;
00793
00794     bitsUsed = (colDR.blue() >> 2) % 8 + 1;
00795     // curMode = colDR.green() % 32;
00796
00797     // Start of the circuit
00798     QByteArray data;
00799     circuit(image, &data, countBytes);
00800
00801     // Check if circuit was successful
00802     if(!success)
00803         return nullptr;
00804     if(data.isEmpty())
00805     {

```

```

00812         fail("noreaddata");
00813         return nullptr;
00814     }
00815     // Version check
00816     long long int _ver = mod(data.at(0)) * qPow(2, 16);
00817     _ver += mod(data.at(1)) * qPow(2, 8);
00818     _ver += mod(data.at(2));
00819     data.remove(0, 3);
00820     if(_ver > version) {
00821         fail("new_version");
00822         return nullptr;
00823     }
00824     else if(_ver < version) {
00825         fail("old_version");
00826         return nullptr;
00827     }
00828     // Get the hash
00829     QByteArray hash = data.left(32);
00830     data.remove(0, 32);
00831     // Unzip
00832     QByteArray unzipped_data = unzip(data, key.toUtf8());
00833     QByteArray our_hash = QCryptographicHash::hash(unzipped_data, QCryptographicHash::Sha256);
00834     if(our_hash != hash) {
00835         fail("veriffail");
00836         return QByteArray("");
00837     }
00838     return unzipped_data;
00839 }
00840 long ModelPC::pop(int bits)
00841 {
00842     // Hard to say
00843     long res = 0;
00844     int poppedBits = bits == -1 ? bitsUsed : bits;
00845     if(bits == -2)
00846         poppedBits = bitsBuffer.size();
00847     for(int i = 0; i < poppedBits; i++)
00848         res += bitsBuffer[i] * qPow(2, poppedBits - i - 1);
00849     bitsBuffer.remove(0, poppedBits);
00850     return res;
00851 }
00852 void ModelPC::push(int data, int bits)
00853 {
00854     // That's easier, but also hard
00855     int buf_size = bitsBuffer.size();
00856     int extraSize = bits == -1 ? bitsUsed : bits;
00857     bitsBuffer.resize(buf_size + extraSize);
00858     for(int i = bitsBuffer.size() - 1; i >= buf_size; i--, data >>= 1)
00859         bitsBuffer[i] = data % 2;
00860 }
00861 bool ModelPC::mustGoOn(bool isEncrypt)
00862 {
00863     return success && (isEncrypt ? (circuitCountBytes - cur) * 8 + bitsBuffer.size() >= bitsUsed * 3
00864 :
00865 circuitData->size() * 8 + bitsBuffer.size() <
00866 circuitCountBytes * 8 - (circuitCountBytes * 8) % (bitsUsed * 3));
00867 }
00868 QByteArray ModelPC::unzip(QByteArray data, QByteArray key)
00869 {
00870     // Decryption
00871     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00872     QAESEncryption encryption(QAESEncryption::AES_256,
00873 QAESEncryption::ECB);
00874     QByteArray new_data = encryption.decode(data, hashKey);
00875     // Decompressing
00876     return qUncompress(new_data);
00877 }
00878 QByteArray ModelPC::zip(QByteArray data, QByteArray key)
00879 {
00880     // Zip
00881     QByteArray c_data = qCompress(data, 9);
00882     // Encryption
00883     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00884     return QAESEncryption::Crypt(QAESEncryption::AES_256,
00885 QAESEncryption::ECB, c_data, hashKey);
00886 }
00887 bool ModelPC::fileExists(QString path)
00888 {
00889     QFileInfo check_file(path);
00890     return check_file.exists() && check_file.isFile();
00891 }
00892 QByteArray ModelPC::bytes(long long n)

```

```

00918 {
00919     return QByteArray::fromHex(QByteArray::number(n, 16));
00920 }
00927 unsigned int ModelPC::mod(int input)
00928 {
00929     if(input < 0)
00930         return (unsigned int) (256 + input);
00931     else
00932         return (unsigned int) input;
00933 }
00940 void ModelPC::alert(QString message, bool isWarning)
00941 {
00942     emit alertView(message, isWarning);
00943 }
00949 QColor ModelPC::RGBbytes(long long byte)
00950 {
00951     int blue = byte % 256;
00952     int green = (byte / 256) % 256;
00953     int red = byte / qPow(2, 16);
00954     return QColor(red, green, blue);
00955 }
00956
00957 QString ModelPC::generateVersionString(long ver)
00958 {
00959     return QString::number((int)( ver / qPow(2, 16))) + "." + QString::number(((int) (ver / 256)) % 256) +
00960     "." + QString::number(ver % 256);
00961 }
00962
00962 uint ModelPC::randSeed()
00963 {
00964     QTime time = QTime::currentTime();
00965     uint randSeed = time.msecsSinceStartOfDay() % 55363 + time.minute() * 21 + time.second() * 2 + 239;
00966     qsrand(randSeed);
00967     uint randSeed_2 = qrand() % 72341 + qrand() % 3 + qrand() % 2 + 566;
00968     return randSeed_2;
00969 }
00970 QByteArray ModelPC::GetRandomBytes(long long count)
00971 {
00972     QByteArray res;
00973     for(int i = 0; i < count; i++)
00974         res.append(qrand() % 256);
00975     return res;
00976 }

```

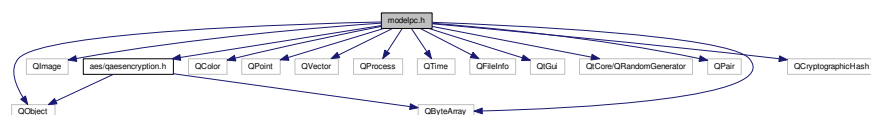
## 8.21 modelpc.h File Reference

```

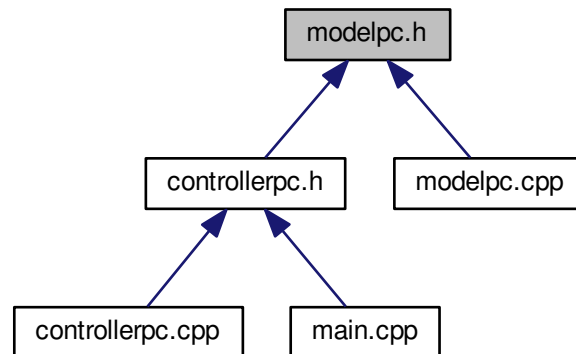
#include <QObject>
#include <QImage>
#include <QByteArray>
#include <QColor>
#include <QPoint>
#include <QVector>
#include <QProcess>
#include <QTime>
#include <QFileInfo>
#include <QtGui>
#include <QtCore/QRandomGenerator>
#include <QPair>
#include "aes/qaesencryption.h"
#include <QCryptographicHash>

```

Include dependency graph for modelpc.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ModelPC](#)

*The [ModelPC](#) class Model Layer of the app. Main class that does the work of PictureCrypt logic Controlled by [ControllerPC](#).*

### 8.21.1 Detailed Description

Header of [ModelPC](#) class

See also

[ControllerPC](#), [ModelPC](#), [ViewPC](#)

Definition in file [modelpc.h](#).

## 8.22 modelpc.h

```

00001 #ifndef MODELPC_H
00002 #define MODELPC_H
00003
00004 #include <QObject>
00005 #include <QImage>
00006 #include <QByteArray>
00007 #include <QColor>
00008 #include <QPoint>
00009 #include <QVector>
00010 #include <QProcess>
00011 #include <QTime>
00012 #include <QFileInfo>
00013 #include <QtGui>
00014 #include <QtCore/QRandomGenerator>
00015 #include <QPair>
00016
00017 #include "aes/gaesencryption.h"
00018 #include <QCryptographicHash>
  
```

```

00019
00020
00033 class ModelPC : public QObject
00034 {
00035     Q_OBJECT
00036 public:
00037     ModelPC();
00038     enum CryptMode {NotDefined, v1_3, v1_4, jphs_mode};
00039     static QImage *Encrypt(QByteArray data, QImage *image, int _mode, QString
key = "", int _bitsUsed = 8, QString *_error = nullptr);
00040     static QImage *Inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString
*_error = nullptr);
00041     static QByteArray Decrypt(QImage * image, QString key, int _mode = 0, QString *_error =
nullptr);
00042
00043 signals:
00050     void alertView(QString messageCode, bool isWarning);
00055     void saveData(QByteArray data);
00060     void saveImage(QImage *image);
00066     void setProgress(int val);
00067
00068 public slots:
00069     QImage *encrypt(QByteArray data, QImage *image, int _mode, QString key = "", int _bitsUsed = 8,
QString *_error = nullptr);
00070     QImage *inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString *
_error = nullptr);
00071     QByteArray decrypt(QImage * image, QString key, int _mode = 0, QString *_error = nullptr);
00072     void fail(QString message);
00073     void alert(QString message, bool isWarning = false);
00074
00075 public:
00076     QByteArray unzip(QByteArray data, QByteArray key);
00077
00082     bool success;
00086     long version;
00090     QString versionString;
00094     QString defaultJPHSDir;
00095 protected:
00096     void circuit(QImage * image, QByteArray * data, long long int countBytes);
00097     void jphs(QImage * image, QByteArray * data);
00098     void processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt);
00099     void encryptv1_4(QImage *image, QByteArray data, QString key);
00100     QByteArray decryptv1_3(QImage * image, QString key);
00101     QByteArray decryptv1_4(QImage * image, QString key);
00102     void proccessPixelsv1_4(QImage *image, QByteArray* data, QByteArray key, bool
isEncrypt, QVector<QPair<QPoint, QPair<int, int> > *were, long long size = -1);
00103     QByteArray zip(QByteArray data, QByteArray key);
00104
00108     QString * error;
00109 private:
00110     int bitsUsed;
00111     bool fileExists(QString path);
00112     QByteArray bytes(long long n);
00113     unsigned int mod(int input);
00114     QByteArray ver_byte;
00115     QColor RGBbytes(long long byte);
00116     QString generateVersionString(long ver);
00117     uint randSeed();
00118     bool isTry = false;
00119
00120     QByteArray * circuitData;
00121     QImage * circuitImage;
00122     long long circuitCountBytes;
00123     long cur;
00124     bool mustGoOn(bool isEncrypt);
00125
00126     QVector <bool> bitsBuffer;
00127     long pop(int bits = -1);
00128     void push(int data, int bits = -1);
00129
00130     void setError(QString word);
00131     QByteArray GetRandomBytes(long long count = 32);
00132 };
00133
00134 #endif // MODELPC_H

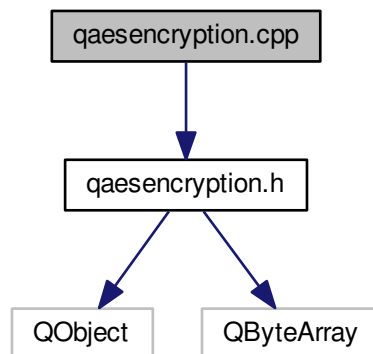
```

## 8.23 qaesencryption.cpp File Reference

```
#include "qaesencryption.h"
```



Include dependency graph for qaesencryption.cpp:



## Functions

- quint8 [xTime](#) (quint8 x)
- quint8 [multiply](#) (quint8 x, quint8 y)

### 8.23.1 Function Documentation

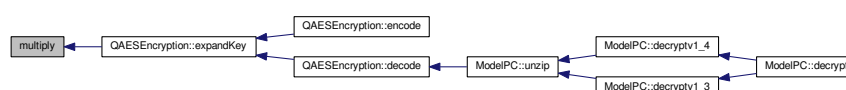
#### 8.23.1.1 quint8 multiply ( quint8 x, quint8 y ) `[inline]`

Definition at line [57](#) of file [qaesencryption.cpp](#).

Here is the call graph for this function:



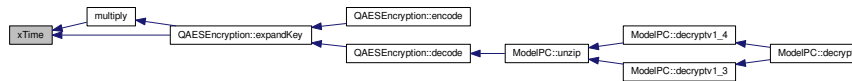
Here is the caller graph for this function:



### 8.23.1.2 quint8 xTime ( quint8 x ) [inline]

Definition at line 53 of file [qaesencryption.cpp](#).

Here is the caller graph for this function:



## 8.24 qaesencryption.cpp

```

00001 #include "qaesencryption.h"
00002
00003 /*
00004  * Static Functions
00005  */
00006 QByteArray QAESEncryption::Crypt(QAESEncryption::Aes level,
00007   QAESEncryption::Mode mode, const QByteArray &rawText,
00008   const QByteArray &key, const QByteArray &iv,
00009   QAESEncryption::Padding padding)
00010 {
00011     return QAESEncryption(level, mode, padding).encode(rawText, key, iv);
00012 }
00013 QByteArray QAESEncryption::Decrypt(QAESEncryption::Aes level,
00014   QAESEncryption::Mode mode, const QByteArray &rawText,
00015   const QByteArray &key, const QByteArray &iv,
00016   QAESEncryption::Padding padding)
00017 {
00018     return QAESEncryption(level, mode, padding).decode(rawText, key, iv);
00019 }
00020 QByteArray QAESEncryption::ExpandKey(
00021   QAESEncryption::Aes level, QAESEncryption::Mode mode, const
00022   QByteArray &key)
00023 {
00024     return QAESEncryption(level, mode).expandKey(key);
00025 }
00026 QByteArray QAESEncryption::RemovePadding(const QByteArray &rawText,
00027   QAESEncryption::Padding padding)
00028 {
00029     QByteArray ret(rawText);
00030     switch (padding)
00031     {
00032     case Padding::ZERO:
00033         //Works only if the last byte of the decoded array is not zero
00034         while (ret.at(ret.length()-1) == 0x00)
00035             ret.remove(ret.length()-1, 1);
00036         break;
00037     case Padding::PKCS7:
00038         ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00039         break;
00040     case Padding::ISO:
00041         ret.truncate(ret.lastIndexOf(0x80));
00042         break;
00043     default:
00044         //do nothing
00045         break;
00046     }
00047     return ret;
00048 }
00049 /*
00050  * End Static function declarations
00051  */
00052
00053 /*
00054  * Inline Functions
00055  */
00056
00057 inline quint8 xTime(quint8 x){
00058     return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
00059 }

```

```

00055 }
00056
00057 inline quint8 multiply(quint8 x, quint8 y){
00058     return ((y & 1) * x) ^ ((y>>1 & 1) * xTime(x)) ^ ((y>>2 & 1) * xTime(
xTime(x))) ^ ((y>>3 & 1)
00059         * xTime(xTime(xTime(x)))) ^ ((y>>4 & 1) * xTime(
xTime(xTime(xTime(x))))));
00060 }
00061
00062 /*
00063  * End Inline functions
00064  */
00065
00066
00067 QAESEncryption::QAESEncryption(Aes level, Mode mode,
                                Padding padding)
00068     : m_nb(4), m_blocklen(16), m_level(level), m_mode(mode), m_padding(padding)
00069 {
00070     m_state = NULL;
00071
00072     switch (level)
00073     {
00074     case AES_128: {
00075         AES128 aes;
00076         m_nk = aes.nk;
00077         m_keyLen = aes.keylen;
00078         m_nr = aes.nr;
00079         m_expandedKey = aes.expandedKey;
00080     }
00081     break;
00082     case AES_192: {
00083         AES192 aes;
00084         m_nk = aes.nk;
00085         m_keyLen = aes.keylen;
00086         m_nr = aes.nr;
00087         m_expandedKey = aes.expandedKey;
00088     }
00089     break;
00090     case AES_256: {
00091         AES256 aes;
00092         m_nk = aes.nk;
00093         m_keyLen = aes.keylen;
00094         m_nr = aes.nr;
00095         m_expandedKey = aes.expandedKey;
00096     }
00097     break;
00098     default: {
00099         AES128 aes;
00100         m_nk = aes.nk;
00101         m_keyLen = aes.keylen;
00102         m_nr = aes.nr;
00103         m_expandedKey = aes.expandedKey;
00104     }
00105     break;
00106     }
00107 }
00108
00109
00110 QByteArray QAESEncryption::getPadding(int currSize, int alignment)
00111 {
00112     int size = (alignment - currSize % alignment) % alignment;
00113     if (size == 0) return QByteArray();
00114     switch(m_padding)
00115     {
00116     case Padding::ZERO:
00117         return QByteArray(size, 0x00);
00118     break;
00119     case Padding::PKCS7:
00120         return QByteArray(size,size);
00121     break;
00122     case Padding::ISO:
00123         return QByteArray (size-1, 0x00).prepend(0x80);
00124     break;
00125     default:
00126         return QByteArray(size, 0x00);
00127     break;
00128     }
00129     return QByteArray(size, 0x00);
00130 }
00131
00132 QByteArray QAESEncryption::expandKey(const QByteArray &
key)
00133 {
00134     int i, k;
00135     quint8 tempa[4]; // Used for the column/row operations
00136     QByteArray roundKey(key);
00137
00138     // The first round key is the key itself.

```

```

00139 // ...
00140
00141 // All other round keys are found from the previous round keys.
00142 //i == Nk
00143 for(i = m_nk; i < m_nb * (m_nr + 1); i++)
00144 {
00145     tempa[0] = (quint8) roundKey.at((i-1) * 4 + 0);
00146     tempa[1] = (quint8) roundKey.at((i-1) * 4 + 1);
00147     tempa[2] = (quint8) roundKey.at((i-1) * 4 + 2);
00148     tempa[3] = (quint8) roundKey.at((i-1) * 4 + 3);
00149
00150     if (i % m_nk == 0)
00151     {
00152         // This function shifts the 4 bytes in a word to the left once.
00153         // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]
00154
00155         // Function RotWord()
00156         k = tempa[0];
00157         tempa[0] = tempa[1];
00158         tempa[1] = tempa[2];
00159         tempa[2] = tempa[3];
00160         tempa[3] = k;
00161
00162         // Function Subword()
00163         tempa[0] = getSBoxValue(tempa[0]);
00164         tempa[1] = getSBoxValue(tempa[1]);
00165         tempa[2] = getSBoxValue(tempa[2]);
00166         tempa[3] = getSBoxValue(tempa[3]);
00167
00168         tempa[0] = tempa[0] ^ Rcon[i/m_nk];
00169     }
00170     if (m_level == AES_256 && i % m_nk == 4)
00171     {
00172         // Function Subword()
00173         tempa[0] = getSBoxValue(tempa[0]);
00174         tempa[1] = getSBoxValue(tempa[1]);
00175         tempa[2] = getSBoxValue(tempa[2]);
00176         tempa[3] = getSBoxValue(tempa[3]);
00177     }
00178     roundKey.insert(i * 4 + 0, (quint8) roundKey.at((i - m_nk) * 4 + 0) ^ tempa[0]);
00179     roundKey.insert(i * 4 + 1, (quint8) roundKey.at((i - m_nk) * 4 + 1) ^ tempa[1]);
00180     roundKey.insert(i * 4 + 2, (quint8) roundKey.at((i - m_nk) * 4 + 2) ^ tempa[2]);
00181     roundKey.insert(i * 4 + 3, (quint8) roundKey.at((i - m_nk) * 4 + 3) ^ tempa[3]);
00182 }
00183 return roundKey;
00184 }
00185
00186 // This function adds the round key to state.
00187 // The round key is added to the state by an XOR function.
00188 void QAESEncryption::addRoundKey(const quint8 round, const QByteArray expKey)
00189 {
00190     QByteArray::iterator it = m_state->begin();
00191     for(int i=0; i < 16; ++i)
00192         it[i] = (quint8) it[i] ^ (quint8) expKey.at(round * m_nb * 4 + (i/4) * m_nb + (i%4));
00193 }
00194
00195 // The SubBytes Function Substitutes the values in the
00196 // state matrix with values in an S-box.
00197 void QAESEncryption::subBytes()
00198 {
00199     QByteArray::iterator it = m_state->begin();
00200     for(int i = 0; i < 16; i++)
00201         it[i] = getSBoxValue((quint8) it[i]);
00202 }
00203
00204 // The ShiftRows() function shifts the rows in the state to the left.
00205 // Each row is shifted with different offset.
00206 // Offset = Row number. So the first row is not shifted.
00207 void QAESEncryption::shiftRows()
00208 {
00209     QByteArray::iterator it = m_state->begin();
00210     quint8 temp;
00211     //Keep in mind that QByteArray is column-driven!!
00212
00213     //Shift 1 to left
00214     temp = (quint8)it[1];
00215     it[1] = (quint8)it[5];
00216     it[5] = (quint8)it[9];
00217     it[9] = (quint8)it[13];
00218     it[13] = (quint8)temp;
00219
00220     //Shift 2 to left
00221     temp = (quint8)it[2];
00222     it[2] = (quint8)it[10];
00223     it[10] = (quint8)temp;
00224     temp = (quint8)it[6];
00225     it[6] = (quint8)it[14];

```

```

00226     it[14] = (quint8)temp;
00227
00228     //Shift 3 to left
00229     temp = (quint8)it[3];
00230     it[3] = (quint8)it[15];
00231     it[15] = (quint8)it[11];
00232     it[11] = (quint8)it[7];
00233     it[7] = (quint8)temp;
00234 }
00235
00236 // MixColumns function mixes the columns of the state matrix
00237 //optimized!!
00238 void QAESEncryption::mixColumns()
00239 {
00240     QByteArray::iterator it = m_state->begin();
00241     quint8 tmp, tm, t;
00242
00243     for(int i = 0; i < 16; i += 4){
00244         t = (quint8)it[i];
00245         tmp = (quint8)it[i] ^ (quint8)it[i+1] ^ (quint8)it[i+2] ^ (quint8)it[i+3] ;
00246
00247         tm = xTime( (quint8)it[i] ^ (quint8)it[i+1] );
00248         it[i] = (quint8)it[i] ^ (quint8)tm ^ (quint8)tmp;
00249
00250         tm = xTime( (quint8)it[i+1] ^ (quint8)it[i+2] );
00251         it[i+1] = (quint8)it[i+1] ^ (quint8)tm ^ (quint8)tmp;
00252
00253         tm = xTime( (quint8)it[i+2] ^ (quint8)it[i+3] );
00254         it[i+2] = (quint8)it[i+2] ^ (quint8)tm ^ (quint8)tmp;
00255
00256         tm = xTime((quint8)it[i+3] ^ (quint8)t);
00257         it[i+3] = (quint8)it[i+3] ^ (quint8)tm ^ (quint8)tmp;
00258     }
00259 }
00260
00261 // MixColumns function mixes the columns of the state matrix.
00262 // The method used to multiply may be difficult to understand for the inexperienced.
00263 // Please use the references to gain more information.
00264 void QAESEncryption::invMixColumns()
00265 {
00266     QByteArray::iterator it = m_state->begin();
00267     quint8 a,b,c,d;
00268     for(int i = 0; i < 16; i+=4){
00269         a = (quint8) it[i];
00270         b = (quint8) it[i+1];
00271         c = (quint8) it[i+2];
00272         d = (quint8) it[i+3];
00273
00274         it[i] = (quint8) (multiply(a, 0x0e) ^ multiply(b, 0x0b) ^
00275 multiply(c, 0x0d) ^ multiply(d, 0x09));
00276         it[i+1] = (quint8) (multiply(a, 0x09) ^ multiply(b, 0x0e) ^
00277 multiply(c, 0x0b) ^ multiply(d, 0x0d));
00278         it[i+2] = (quint8) (multiply(a, 0x0d) ^ multiply(b, 0x09) ^
00279 multiply(c, 0x0e) ^ multiply(d, 0x0b));
00280         it[i+3] = (quint8) (multiply(a, 0x0b) ^ multiply(b, 0x0d) ^
00281 multiply(c, 0x09) ^ multiply(d, 0x0e));
00282     }
00283 }
00284
00285 // The SubBytes Function Substitutes the values in the
00286 // state matrix with values in an S-box.
00287 void QAESEncryption::invSubBytes()
00288 {
00289     QByteArray::iterator it = m_state->begin();
00290     for(int i = 0; i < 16; ++i)
00291         it[i] = getSBoxInvert((quint8) it[i]);
00292 }
00293
00294 void QAESEncryption::invShiftRows()
00295 {
00296     QByteArray::iterator it = m_state->begin();
00297     uint8_t temp;
00298
00299     //Keep in mind that QByteArray is column-driven!!
00300
00301     //Shift 1 to right
00302     temp = (quint8)it[13];
00303     it[13] = (quint8)it[9];
00304     it[9] = (quint8)it[5];
00305     it[5] = (quint8)it[1];
00306     it[1] = (quint8)temp;
00307
00308     //Shift 2
00309     temp = (quint8)it[10];
00310     it[10] = (quint8)it[2];
00311     it[2] = (quint8)temp;
00312
00313     temp = (quint8)it[14];
00314     it[14] = (quint8)it[6];
00315     it[6] = (quint8)it[3];
00316     it[3] = (quint8)it[7];
00317     it[7] = (quint8)temp;
00318 }

```

```

00309     it[14] = (quint8)it[6];
00310     it[6]  = (quint8)temp;
00311
00312     //Shift 3
00313     temp  = (quint8)it[15];
00314     it[15] = (quint8)it[3];
00315     it[3]  = (quint8)it[7];
00316     it[7]  = (quint8)it[11];
00317     it[11] = (quint8)temp;
00318 }
00319
00320 QByteArray QAESEncryption::byteXor(const QByteArray &a, const QByteArray &b)
00321 {
00322     QByteArray::const_iterator it_a = a.begin();
00323     QByteArray::const_iterator it_b = b.begin();
00324     QByteArray ret;
00325
00326     //for(int i = 0; i < m_blocklen; i++)
00327     for(int i = 0; i < std::min(a.size(), b.size()); i++)
00328         ret.insert(i, it_a[i] ^ it_b[i]);
00329
00330     return ret;
00331 }
00332
00333 // Cipher is the main function that encrypts the PlainText.
00334 QByteArray QAESEncryption::cipher(const QByteArray &expKey, const QByteArray &in)
00335 {
00336     //m_state is the input buffer...
00337     QByteArray output(in);
00338     m_state = &output;
00339
00340     // Add the First round key to the state before starting the rounds.
00341     addRoundKey(0, expKey);
00342
00343     // There will be Nr rounds.
00344     // The first Nr-1 rounds are identical.
00345     // These Nr-1 rounds are executed in the loop below.
00346     for(quint8 round = 1; round < m_nr; ++round){
00347         subBytes();
00348         shiftRows();
00349         mixColumns();
00350         addRoundKey(round, expKey);
00351     }
00352
00353     // The last round is given below.
00354     // The MixColumns function is not here in the last round.
00355     subBytes();
00356     shiftRows();
00357     addRoundKey(m_nr, expKey);
00358
00359     return output;
00360 }
00361
00362 QByteArray QAESEncryption::invCipher(const QByteArray &expKey, const QByteArray &in)
00363 {
00364     //m_state is the input buffer.... handle it!
00365     QByteArray output(in);
00366     m_state = &output;
00367
00368     // Add the First round key to the state before starting the rounds.
00369     addRoundKey(m_nr, expKey);
00370
00371     // There will be Nr rounds.
00372     // The first Nr-1 rounds are identical.
00373     // These Nr-1 rounds are executed in the loop below.
00374     for(quint8 round=m_nr-1; round>0 ; round--){
00375         invShiftRows();
00376         invSubBytes();
00377         addRoundKey(round, expKey);
00378         invMixColumns();
00379     }
00380
00381     // The last round is given below.
00382     // The MixColumns function is not here in the last round.
00383     invShiftRows();
00384     invSubBytes();
00385     addRoundKey(0, expKey);
00386
00387     return output;
00388 }
00389
00390
00391 QByteArray QAESEncryption::encode(const QByteArray &rawText, const QByteArray &
key, const QByteArray &iv)
00392 {
00393     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00394         return QByteArray();

```

```

00395
00396     QByteArray ret;
00397     QByteArray expandedKey = expandKey(key);
00398     QByteArray alignedText(rawText);
00399
00400     //Fill array with padding
00401     alignedText.append(getPadding(rawText.size(), m_blocklen));
00402
00403     switch(m_mode)
00404     {
00405     case ECB:
00406         for(int i=0; i < alignedText.size(); i+= m_blocklen)
00407             ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00408         break;
00409     case CBC: {
00410         QByteArray ivTemp(iv);
00411         for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00412             alignedText.replace(i, m_blocklen, byteXor(alignedText.mid(i, m_blocklen), ivTemp));
00413             ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00414             ivTemp = ret.mid(i, m_blocklen);
00415         }
00416     }
00417     break;
00418     case CFB: {
00419         ret.append(byteXor(alignedText.left(m_blocklen), cipher(expandedKey, iv)));
00420         for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00421             if (i+m_blocklen < alignedText.size())
00422                 ret.append(byteXor(alignedText.mid(i+m_blocklen, m_blocklen),
00423                                     cipher(expandedKey, ret.mid(i, m_blocklen))));
00424         }
00425     }
00426     break;
00427     case OFB: {
00428         QByteArray ofbTemp;
00429         ofbTemp.append(cipher(expandedKey, iv));
00430         for (int i=m_blocklen; i < alignedText.size(); i += m_blocklen){
00431             ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00432         }
00433         ret.append(byteXor(alignedText, ofbTemp));
00434     }
00435     break;
00436     default: break;
00437 }
00438     return ret;
00439 }
00440
00441 QByteArray QAESEncryption::decode(const QByteArray &rawText, const QByteArray &
key, const QByteArray &iv)
00442 {
00443     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00444         return QByteArray();
00445
00446     QByteArray ret;
00447     QByteArray expandedKey = expandKey(key);
00448
00449     switch(m_mode)
00450     {
00451     case ECB:
00452         for(int i=0; i < rawText.size(); i+= m_blocklen)
00453             ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00454         break;
00455     case CBC: {
00456         QByteArray ivTemp(iv);
00457         for(int i=0; i < rawText.size(); i+= m_blocklen){
00458             ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00459             ret.replace(i, m_blocklen, byteXor(ret.mid(i, m_blocklen), ivTemp));
00460             ivTemp = rawText.mid(i, m_blocklen);
00461         }
00462     }
00463     break;
00464     case CFB: {
00465         ret.append(byteXor(rawText.mid(0, m_blocklen), cipher(expandedKey, iv)));
00466         for(int i=0; i < rawText.size(); i+= m_blocklen){
00467             if (i+m_blocklen < rawText.size()) {
00468                 ret.append(byteXor(rawText.mid(i+m_blocklen, m_blocklen),
00469                                     cipher(expandedKey, rawText.mid(i, m_blocklen))));
00470             }
00471         }
00472     }
00473     break;
00474     case OFB: {
00475         QByteArray ofbTemp;
00476         ofbTemp.append(cipher(expandedKey, iv));
00477         for (int i=m_blocklen; i < rawText.size(); i += m_blocklen){
00478             ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00479         }
00480         ret.append(byteXor(rawText, ofbTemp));

```

```

00481     }
00482         break;
00483     default:
00484         //do nothing
00485         break;
00486     }
00487     return ret;
00488 }
00489
00490 QByteArray QAESEncryption::removePadding(const QByteArray &rawText)
00491 {
00492     QByteArray ret(rawText);
00493     switch (m_padding)
00494     {
00495     case Padding::ZERO:
00496         //Works only if the last byte of the decoded array is not zero
00497         while (ret.at(ret.length()-1) == 0x00)
00498             ret.remove(ret.length()-1, 1);
00499         break;
00500     case Padding::PKCS7:
00501         ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00502         break;
00503     case Padding::ISO:
00504         ret.truncate(ret.lastIndexOf(0x80));
00505         break;
00506     default:
00507         //do nothing
00508         break;
00509     }
00510     return ret;
00511 }

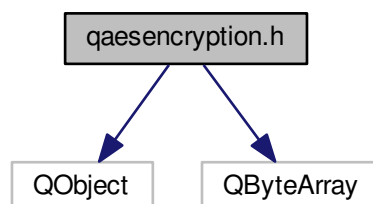
```

## 8.25 qaesencryption.h File Reference

```
#include <QObject>
```

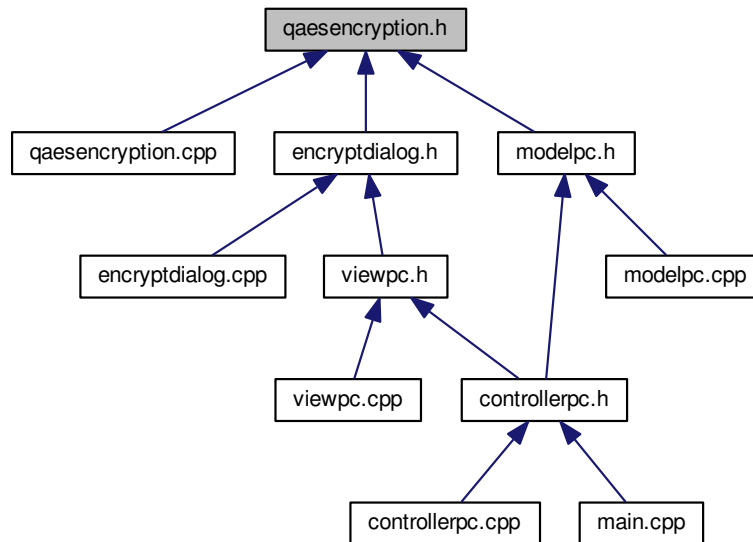
```
#include <QByteArray>
```

Include dependency graph for qaesencryption.h:





This graph shows which files directly or indirectly include this file:



## Classes

- class `QAESEncryption`

The `QAESEncryption` class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: <https://github.com/bricke/qt-AES>.

## 8.26 qaesencryption.h

```

00001 #ifndef QAESENCRYPTION_H
00002 #define QAESENCRYPTION_H
00003
00004 #include <QObject>
00005 #include <QByteArray>
00006
00014 class QAESEncryption : public QObject
00015 {
00016     Q_OBJECT
00017 public:
00027     enum Aes {
00028         AES_128,
00029         AES_192,
00030         AES_256
00031     };
00040     enum Mode {
00041         ECB,
00042         CBC,
00043         CFB,
00044         OFB
00045     };
00046
00055     enum Padding {
00056         ZERO,
00057         PKCS7,
00058         ISO
00059     };
00071     static QByteArray Crypt(QAESEncryption::Aes level,

```

```

    QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &
    key,
00072         const QByteArray &iv = NULL, QAESEncryption::Padding
padding = QAESEncryption::ISO);
00084     static QByteArray Decrypt(QAESEncryption::Aes level,
QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &
    key,
00085         const QByteArray &iv = NULL,
QAESEncryption::Padding padding = QAESEncryption::ISO);
00094     static QByteArray ExpandKey(QAESEncryption::Aes level,
QAESEncryption::Mode mode, const QByteArray &key);
00102     static QByteArray RemovePadding(const QByteArray &rawText,
QAESEncryption::Padding padding);
00103
00104     QAESEncryption(QAESEncryption::Aes level,
QAESEncryption::Mode mode,
00105         QAESEncryption::Padding padding =
QAESEncryption::ISO);
00116     QByteArray encode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv =
    NULL);
00127     QByteArray decode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv =
    NULL);
00136     QByteArray removePadding(const QByteArray &rawText);
00145     QByteArray expandKey(const QByteArray &key);
00146
00147     signals:
00148
00149     public slots:
00150
00151     private:
00152         int m_nb;
00153         int m_blocklen;
00154         int m_level;
00155         int m_mode;
00156         int m_nk;
00157         int m_keyLen;
00158         int m_nr;
00159         int m_expandedKey;
00160         int m_padding;
00161         QByteArray* m_state;
00162
00163         struct AES256{
00164             int nk = 8;
00165             int keylen = 32;
00166             int nr = 14;
00167             int expandedKey = 240;
00168         };
00169
00170         struct AES192{
00171             int nk = 6;
00172             int keylen = 24;
00173             int nr = 12;
00174             int expandedKey = 209;
00175         };
00176
00177         struct AES128{
00178             int nk = 4;
00179             int keylen = 16;
00180             int nr = 10;
00181             int expandedKey = 176;
00182         };
00183
00184         quint8 getSBoxValue(quint8 num){return sbox[num];}
00185         quint8 getSBoxInvert(quint8 num){return rsbox[num];}
00186
00187         void addRoundKey(const quint8 round, const QByteArray expKey);
00188         void subBytes();
00189         void shiftRows();
00190         void mixColumns();
00191         void invMixColumns();
00192         void invSubBytes();
00193         void invShiftRows();
00194         QByteArray getPadding(int currSize, int alignment);
00195         QByteArray cipher(const QByteArray &expKey, const QByteArray &plainText);
00196         QByteArray invCipher(const QByteArray &expKey, const QByteArray &plainText);
00197         QByteArray byteXor(const QByteArray &in, const QByteArray &iv);
00198
00199         const quint8 sbox[256] = {
00200             //0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
00201             0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
00202             0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
00203             0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
00204             0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
00205             0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
00206             0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
00207             0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
00208             0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,

```

## 8.27 viewpc.cpp File Reference

Include dependency graph for viewpc.cpp:



Generated by Doxygen

```

00002 #include "ui_viewpc.h"
00003
00004 ViewPC::ViewPC(QWidget *parent) :
00005     QMainWindow(parent),
00006     ui(new Ui::ViewPC)
00007 {
00008     ui->setupUi(this);
00009
00010     progressDialogClosed = true;
00011
00012     // Alerts dictionary setup
00013     QFile file(":/config/ErrorsDict.json");
00014     if(!file.open(QFile::ReadOnly | QFile::Text)) {
00015         alert("Cannot open config file!");
00016         return;
00017     }
00018     QByteArray readData = file.readAll();
00019     file.close();
00020
00021     QJsonParseError error;
00022     QJsonDocument doc = QJsonDocument::fromJson(readData, &error);
00023     errorsDict = doc.object();
00024     isEncrypt = true;
00025 }
00029 ViewPC::~ViewPC()
00030 {
00031     delete ui;
00032 }
00033
00034 void ViewPC::on_encryptMode_clicked()
00035 {
00036     // Encrypt radio button clicked
00037     setEncryptMode(true);
00038 }
00039
00040 void ViewPC::on_decryptMode_clicked()
00041 {
00042     // Decrypt radio button clicked
00043     setEncryptMode(false);
00044 }
00048 void ViewPC::on_fileButton_clicked()
00049 {
00050     // Opening QFileDialog depending on isEncrypt
00051     if(isEncrypt)
00052         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.txt", tr("Text
files (*.txt);;All Files (*)"));
00053     else
00054         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.png", tr("PNG
files (*.png);;All Files (*)"));
00055     // Display the file name
00056     ui->fileLabel->setText(inputFileName.isEmpty() ? "File not chosen" : inputFileName);
00057 }
00070 void ViewPC::on_startButton_clicked()
00071 {
00072     if(isEncrypt)
00073     {
00074         // Getting the data
00075         QString text = ui->text->toPlainText();
00076         QByteArray data;
00077         if(text.isEmpty()) {
00078             if(inputFileName.isEmpty()) {
00079                 alert("no_input_file", true);
00080                 return;
00081             }
00082             // Opening the file
00083             QFile file(inputFileName);
00084             if (!file.open(QIODevice::ReadOnly))
00085             {
00086                 alert("open_file_fail", true);
00087                 return;
00088             }
00089             // Check the data size
00090             auto size = file.size();
00091             if(size > qPow(2, 24)) {
00092                 alert("muchdata", true);
00093                 file.close();
00094                 return;
00095             }
00096             data = file.readAll();
00097             file.close();
00098         }
00099         else
00100             data = text.toUtf8();
00101         // Select image via EncryptDialog
00102         EncryptDialog * dialog = new EncryptDialog(data);
00103         dialog->exec();
00104         if(!dialog->success)

```

```

00105         return;
00106
00107         // Get the data
00108         QByteArray encr_data = dialog->compr_data;
00109
00110         // Save the hash
00111         QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
00112         encr_data = hash + encr_data;
00113
00114         switch (selectedMode) {
00115         case 1:
00116             emit inject(encr_data, &dialog->image, selectedMode, dialog->
bitsUsed);
00117             break;
00118         case 2:
00119             emit encrypt(data, &dialog->image, selectedMode, dialog->
key);
00120             break;
00121         }
00122     }
00123     else
00124     {
00125         // Get the filename of the image
00126         if(inputFileName.isEmpty()) {
00127             alert("no_input_file", true);
00128             return;
00129         }
00130         QByteArray key = requestKey().toUtf8();
00131         if(key.isEmpty())
00132             return;
00133         QImage * res_image = new QImage(inputFileName);
00134         emit decrypt(res_image, key, 0);
00135     }
00136 }
00142 void ViewPC::alert(QString message, bool isWarning)
00143 {
00144     // Get message
00145     if(errorsDict.contains(message))
00146         message = errorsDict[message].toString();
00147     // Create message box
00148     QMessageBox box;
00149     if(isWarning)
00150         box.setIcon(QMessageBox::Warning);
00151     else
00152         box.setIcon(QMessageBox::Information);
00153     box.setText(message);
00154     box.setWindowIcon(QIcon(":/icons/mail.png"));
00155     box.setWindowTitle("Message");
00156     box.exec();
00157 }
00163 void ViewPC::saveData(QByteArray Edata)
00164 {
00165     // Save data using QFileDialog
00166     QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save File"),
"/untitled.txt",
tr("Text (*.txt);;All files (*)"));
00169     QFile writeFile(outputFileName);
00170     if (!writeFile.open(QIODevice::WriteOnly))
00171     {
00172         alert("save_file_fail", true);
00173         return;
00174     }
00175     writeFile.write(Edata);
00176     writeFile.close();
00177     alert("decryption_completed");
00178 }
00184 void ViewPC::saveImage(QImage * image)
00185 {
00186     // Save image using QFileDialog
00187     QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save Image"),
"/untitled.png",
tr("Images (*.png)"));
00190     if(!image->save(outputFileName)) {
00191         alert("save_file_fail", true);
00192         return;
00193     }
00194     alert("encryption_completed");
00195 }
00202 void ViewPC::setProgress(int val)
00203 {
00204     if(val < 0) {
00205         // Create dialog
00206         dialog = new QProgressDialog("Crypton in progress.", "Cancel", 0, 100);
00207         connect(dialog, SIGNAL(canceled()), this, SLOT(abortCircuit()));
00208         progressDialogClosed = false;
00209         dialog->setWindowTitle("Processing");
00210         dialog->setWindowIcon(QIcon(":/icons/loading.png"));

```

```

00211         dialog->show();
00212     }
00213     else if(val > 100 && !progressDialogClosed) {
00214         // Close dialog
00215         dialog->setValue(100);
00216         QThread::msleep(25);
00217         dialog->close();
00218         dialog->reset();
00219         progressDialogClosed = true;
00220     }
00221     // Update the progress
00222     else if(!progressDialogClosed)
00223         dialog->setValue(val);
00224 }
00228 void ViewPC::abortCircuit()
00229 {
00230     // Set the flag
00231     progressDialogClosed = true;
00232     // Close the dialog
00233     dialog->close();
00234     dialog->reset();
00235     emit abortModel();
00236 }
00241 void ViewPC::setEncryptMode(bool encr)
00242 {
00243     ui->text->setText("");
00244     ui->text->setEnabled(encr);
00245     isEncrypt = encr;
00246     ui->startButton->setText(encr ? "Continue configuration" : "Start decryption");
00247     ui->enLabel1->setText(encr ? "Type in the text for encryption:" : "Text input isn't supported in
decryption mode");
00248     ui->enLabel1->setEnabled(encr);
00249     ui->enLabel2->setText(encr ? "Or use the file dialog to choose a file:" : "Choose a file for
decryption:");
00250     ui->comboBox->setEnabled(encr);
00251 }
00256 void ViewPC::setVersion(QString version)
00257 {
00258     // Version setup
00259     versionString = version;
00260 }
00265 QString ViewPC::requestKey()
00266 {
00267     bool ok;
00268     QString text = QInputDialog::getText(this, tr("QInputDialog::getText()"),
tr("Enter the keyphrase:"), QLineEdit::Normal,
QDir::home().dirName(), &ok);
00270
00271     if(text.isEmpty() && ok) {
00272         alert("no_key", true);
00273         return QString();
00274     }
00275     return ok ? text : QString();
00276 }
00277
00278 QByteArray ViewPC::bytes(long long n)
00279 {
00280     return QByteArray::fromHex(QByteArray::number(n, 16));
00281 }
00285 void ViewPC::on_actionAbout_triggered()
00286 {
00287     AboutPC about;
00288     about.setVersion(versionString);
00289     about.exec();
00290 }
00291
00295 void ViewPC::on_actionHelp_triggered()
00296 {
00297     QUrl docLink("https://alekovrigin.me/PictureCrypt");
00298     QDesktopServices::openUrl(docLink);
00299 }
00300
00301 void ViewPC::on_actionJPHS_path_triggered()
00302 {
00303     QString dir = QFileDialog::getExistingDirectory(this, tr("Open JPHS folder"),
"/home",
QFileDialog::ShowDirsOnly
| QFileDialog::DontResolveSymlinks);
00307     emit setJPHSDir(dir);
00308 }
00309
00310 void ViewPC::on_comboBox_currentIndexChanged(int index)
00311 {
00312     selectedMode = index + 1;
00313 }
00314
00315 void ViewPC::on_text_textChanged()
00316 {

```

```

00317     ui->fileButton->setEnabled(ui->text->toPlainText().isEmpty());
00318 }

```

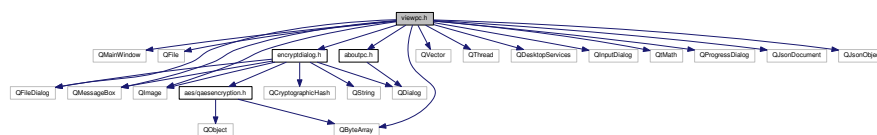
## 8.29 viewpc.h File Reference

```

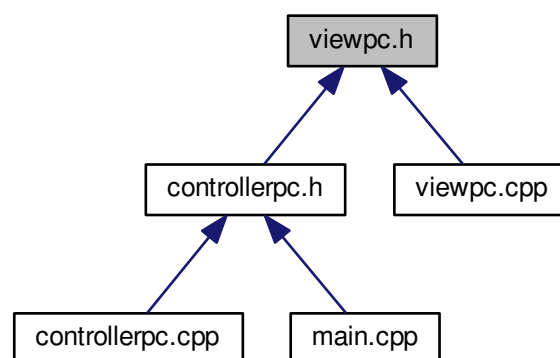
#include <QMainWindow>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QImage>
#include <QByteArray>
#include <QVector>
#include <QThread>
#include <QDesktopServices>
#include <QInputDialog>
#include <QtMath>
#include <encryptdialog.h>
#include <QProgressDialog>
#include <aboutpc.h>
#include <QJsonDocument>
#include <QJsonObject>

```

Include dependency graph for viewpc.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [ViewPC](#)

The [ViewPC](#) class View layer of the app. Controls [EncryptDialog](#) and [ProgressDialog](#).

## Namespaces

- [Ui](#)

### 8.29.1 Detailed Description

Header of [ViewPC](#) class

See also

[ControllerPC](#), [ModelPC](#), [ViewPC](#)

Definition in file [viewpc.h](#).

## 8.30 viewpc.h

```

00001 #ifndef VIEWPC_H
00002 #define VIEWPC_H
00003
00004 #include <QMainWindow>
00005 #include <QFile>
00006 #include <QFileDialog>
00007 #include <QMessageBox>
00008 #include <QImage>
00009 #include <QByteArray>
00010 #include <QVector>
00011 #include <QThread>
00012 #include <QDesktopServices>
00013 #include <QInputDialog>
00014 #include <QtMath>
00015
00016 #include <encryptdialog.h>
00017 #include <QProgressDialog>
00018 #include <aboutpc.h>
00019
00020 #include <QJsonDocument>
00021 #include <QJsonObject>
00022
00023 namespace Ui {
00024 class ViewPC;
00025 }
00035 class ViewPC : public QMainWindow
00036 {
00037     Q_OBJECT
00038
00039 public:
00040     explicit ViewPC(QWidget *parent = nullptr);
00041     ~ViewPC();
00042 private slots:
00043     void on_encryptMode_clicked();
00044
00045     void on_decryptMode_clicked();
00046
00047     void on_actionJPHS_path_triggered();
00048
00049     void on_comboBox_currentIndexChanged(int index);
00050
00051     void on_text_textChanged();
00052
00053 protected slots:
00054     void on_fileButton_clicked();
00055
00056     void on_startButton_clicked();
00057
00058     void on_actionAbout_triggered();
00059
00060     void on_actionHelp_triggered();
00061 public slots:
00062     void alert(QString message, bool isWarning = false);
00063     void saveData(QByteArray Edata);
00064     void saveImage(QImage *image);
00065     void setProgress(int val);

```



```
00066     void abortCircuit();
00067     void setEncryptMode(bool encr);
00068     void setVersion(QString version);
00069 signals:
00077     void encrypt(QByteArray data, QImage *image, int mode, QString key);
00085     void inject(QByteArray data, QImage * image, int mode, int bitsUsed);
00093     void decrypt(QImage * _image, QString key, int mode);
00097     void abortModel();
00102     void setJPHSDir(QString dir);
00103 public:
00108     QProgressDialog * dialog;
00113     bool progressDialogClosed;
00117     QJsonObject errorsDict;
00118 protected:
00119     QString requestKey();
00120 private:
00121     Ui::ViewPC *ui;
00122     bool isEncrypt;
00123     QString inputFileName;
00124     QByteArray bytes(long long n);
00125     QString versionString;
00126     int selectedMode = 2;
00127 };
00128
00129 #endif // VIEWPC_H
```



# Index

- ~AboutPC
  - AboutPC, [16](#)
- ~EncryptDialog
  - EncryptDialog, [22](#)
- ~ViewPC
  - ViewPC, [55](#)
- AES\_128
  - QAESEncryption, [46](#)
- AES\_192
  - QAESEncryption, [46](#)
- AES\_256
  - QAESEncryption, [46](#)
- abortCircuit
  - ControllerPC, [19](#)
  - ViewPC, [56](#)
- abortModel
  - ViewPC, [56](#)
- AboutPC, [15](#)
  - ~AboutPC, [16](#)
  - AboutPC, [16](#)
  - setVersion, [16](#)
- aboutpc.cpp, [65](#)
- aboutpc.h, [66](#)
- Aes
  - QAESEncryption, [46](#)
- alert
  - ModelIPC, [29](#)
  - ViewPC, [56](#)
- alertView
  - ModelIPC, [30](#)
- bitsUsed
  - EncryptDialog, [24](#)
- CBC
  - QAESEncryption, [47](#)
- CFB
  - QAESEncryption, [47](#)
- circuit
  - ModelIPC, [30](#)
- compr\_data
  - EncryptDialog, [24](#)
- ControllerPC, [17](#)
  - abortCircuit, [19](#)
  - ControllerPC, [18](#)
  - setJPHSDir, [19](#)
  - version, [19](#)
  - versionString, [19](#)
- controllerpc.cpp, [67](#)
- controllerpc.h, [68](#)
- Crypt
  - QAESEncryption, [47](#)
- CryptMode
  - ModelIPC, [28](#)
- data
  - EncryptDialog, [24](#)
  - ErrorsDictSetup, [13](#)
- decode
  - QAESEncryption, [48](#)
- Decrypt
  - ModelIPC, [31](#)
  - QAESEncryption, [49](#)
- decrypt
  - ModelIPC, [31](#)
  - ViewPC, [57](#)
- decryptv1\_3
  - ModelIPC, [33](#)
- decryptv1\_4
  - ModelIPC, [34](#)
- defaultJPHSDir
  - ModelIPC, [44](#)
- dialog
  - ViewPC, [63](#)
- ECB
  - QAESEncryption, [47](#)
- encode
  - QAESEncryption, [50](#)
- Encrypt
  - ModelIPC, [35](#)
- encrypt
  - ModelIPC, [35](#)
  - ViewPC, [58](#)
- EncryptDialog, [20](#)
  - ~EncryptDialog, [22](#)
  - bitsUsed, [24](#)
  - compr\_data, [24](#)
  - data, [24](#)
  - EncryptDialog, [22](#)
  - goodPercentage, [24](#)
  - image, [24](#)
  - inputFileName, [24](#)
  - key, [24](#)
  - on\_bitsSlider\_valueChanged, [22](#)
  - on\_buttonBox\_accepted, [22](#)
  - on\_buttonBox\_rejected, [23](#)
  - on\_fileButton\_clicked, [23](#)
  - size, [24](#)

- success, 25
  - val, 25
  - zip, 23
- encryptdialog.cpp, 69
- encryptdialog.h, 70
- encryptv1\_4
  - ModelPC, 36
- error
  - ModelPC, 44
- errorsDict
  - ViewPC, 63
- ErrorsDict.json, 72
- ErrorsDictSetup, 13
  - data, 13
  - f, 13
  - filename, 13
  - indent, 13
  - input\_data, 13
  - key, 14
  - raw, 14
  - value, 14
- ErrorsDictSetup.py, 72, 73
- ExpandKey
  - QAESEncryption, 51
- expandKey
  - QAESEncryption, 51
- f
  - ErrorsDictSetup, 13
- fail
  - ModelPC, 36
- filename
  - ErrorsDictSetup, 13
- goodPercentage
  - EncryptDialog, 24
- ISO
  - QAESEncryption, 47
- image
  - EncryptDialog, 24
- indent
  - ErrorsDictSetup, 13
- Inject
  - ModelPC, 37
- inject
  - ModelPC, 38
  - ViewPC, 58
- input\_data
  - ErrorsDictSetup, 13
- inputFileName
  - EncryptDialog, 24
- jphs
  - ModelPC, 39
- jphs\_mode
  - ModelPC, 28
- key
  - EncryptDialog, 24
  - ErrorsDictSetup, 14
- main
  - main.cpp, 73
- main.cpp, 73
  - main, 73
- Mode
  - QAESEncryption, 46
- ModelPC, 25
  - alert, 29
  - alertView, 30
  - circuit, 30
  - CryptMode, 28
  - Decrypt, 31
  - decrypt, 31
  - decryptv1\_3, 33
  - decryptv1\_4, 34
  - defaultJPHSDir, 44
  - Encrypt, 35
  - encrypt, 35
  - encryptv1\_4, 36
  - error, 44
  - fail, 36
  - Inject, 37
  - inject, 38
  - jphs, 39
  - jphs\_mode, 28
  - ModelPC, 28
  - NotDefined, 28
  - processPixelsv1\_4, 39
  - processPixel, 40
  - saveData, 41
  - saveImage, 41
  - setProgress, 41
  - success, 44
  - unzip, 42
  - v1\_3, 28
  - v1\_4, 28
  - version, 44
  - versionString, 44
  - zip, 43
- modelpc.cpp, 74
- modelpc.h, 84
- multiply
  - qaesencryption.cpp, 87
- NotDefined
  - ModelPC, 28
- OFB
  - QAESEncryption, 47
- on\_actionAbout\_triggered
  - ViewPC, 58
- on\_actionHelp\_triggered
  - ViewPC, 59
- on\_bitsSlider\_valueChanged
  - EncryptDialog, 22
- on\_buttonBox\_accepted

- EncryptDialog, 22
- on\_buttonBox\_rejected
  - EncryptDialog, 23
- on\_fileButton\_clicked
  - EncryptDialog, 23
  - ViewPC, 59
- on\_startButton\_clicked
  - ViewPC, 59
- PKCS7
  - QAESEncryption, 47
- Padding
  - QAESEncryption, 47
- processPixelsv1\_4
  - ModelPC, 39
- processPixel
  - ModelPC, 40
- progressDialogClosed
  - ViewPC, 64
- QAESEncryption, 45
  - AES\_128, 46
  - AES\_192, 46
  - AES\_256, 46
  - Aes, 46
  - CBC, 47
  - CFB, 47
  - Crypt, 47
  - decode, 48
  - Decrypt, 49
  - ECB, 47
  - encode, 50
  - ExpandKey, 51
  - expandKey, 51
  - ISO, 47
  - Mode, 46
  - OFB, 47
  - PKCS7, 47
  - Padding, 47
  - QAESEncryption, 47
  - RemovePadding, 52
  - removePadding, 53
  - ZERO, 47
- qaesencryption.cpp, 86, 88
  - multiply, 87
  - xTime, 87
- qaesencryption.h, 94, 95
- raw
  - ErrorsDictSetup, 14
- RemovePadding
  - QAESEncryption, 52
- removePadding
  - QAESEncryption, 53
- requestKey
  - ViewPC, 60
- saveData
  - ModelPC, 41
- ViewPC, 60
- savelImage
  - ModelPC, 41
  - ViewPC, 61
- setEncryptMode
  - ViewPC, 61
- setJPHSDir
  - ControllerPC, 19
  - ViewPC, 62
- setProgress
  - ModelPC, 41
  - ViewPC, 62
- setVersion
  - AboutPC, 16
  - ViewPC, 63
- size
  - EncryptDialog, 24
- success
  - EncryptDialog, 25
  - ModelPC, 44
- Ui, 14
- unzip
  - ModelPC, 42
- v1\_3
  - ModelPC, 28
- v1\_4
  - ModelPC, 28
- val
  - EncryptDialog, 25
- value
  - ErrorsDictSetup, 14
- version
  - ControllerPC, 19
  - ModelPC, 44
- versionString
  - ControllerPC, 19
  - ModelPC, 44
- ViewPC, 53
  - ~ViewPC, 55
  - abortCircuit, 56
  - abortModel, 56
  - alert, 56
  - decrypt, 57
  - dialog, 63
  - encrypt, 58
  - errorsDict, 63
  - inject, 58
  - on\_actionAbout\_triggered, 58
  - on\_actionHelp\_triggered, 59
  - on\_fileButton\_clicked, 59
  - on\_startButton\_clicked, 59
  - progressDialogClosed, 64
  - requestKey, 60
  - saveData, 60
  - savelImage, 61
  - setEncryptMode, 61
  - setJPHSDir, 62

- setProgress, [62](#)
  - setVersion, [63](#)
  - ViewPC, [55](#)
- viewpc.cpp, [97](#)
- viewpc.h, [101](#)
- xTime
  - qaesencryption.cpp, [87](#)
- ZERO
  - QAESEncryption, [47](#)
- zip
  - EncryptDialog, [23](#)
  - ModelPC, [43](#)