

Seminararbeit: Implementierung einer grafischen Benutzeroberfläche zum Backup, Restore und Upload

Sommersemester 2019-2020

Verfasser:	Fengxiang Hu, Xu Jing
Studiengang:	Computer Engineering
Matrikelnummer:	10022267(Hu), 10027279 (Xu)
E-Mail:	fengxianghu026@gmail.com

Prüfer:	Dr.-Ing. Torsten Lilge
Abgabetermin:	12.03.2020

Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	II
Tabellenverzeichnis	III
1. Einleitung	1
1.1. Aufgabenstellung	1
1.2. Motivation	2
2. Grundlagen der Entwicklung der Benutzeroberflächen in Qt	3
2.1. Was ist Qt?	3
2.2. Kommunikation im Projekt	3
3. Erzeugnis	5
3.1. Auslegung der grafischen Arbeitsoberfläche	5
3.2. Implementierung der Funktionen	8
4. Codeanalyse	12
5. Fazit	21

Abbildungsverzeichnis

Abbildung 1:Rangliste der Programmiersprachen in Tobi	2
Abbildung 2: Verzeichnis des Projektes	4
Abbildung 3: die grafische Benutzeroberfläche	5
Abbildung 4: Der Bereich des Backups	5
Abbildung 5: Der Bereich des Restores	6
Abbildung 6: Der Bereich des Uploads	6
Abbildung 7: Der Bereich des Anzeigefeldes	7
Abbildung 8: Der Bereich der Erweiterung	7
Abbildung 9: Das Verzeichnis der Widgets	8
Abbildung 10: Die Auslegung der grafischen Benutzeroberfläche	8
Abbildung 11: Die Auslegung des Bereichs des Backups	9
Abbildung 12: Die Auslegung des Bereichs des Restores	10
Abbildung 13: Die Auslegung des Bereichs des Anzeigefeldes	10
Abbildung 14: Die Auslegung der Erweiterung	10
Abbildung 15: Funktionsdeklarationen von Slots	11
Abbildung 16: Header-Datei	12
Abbildung 17: Der Speicherort der Konfigurationsdatei	13
Abbildung 18: Der Teil der Initialisierung der Codes_1	13
Abbildung 19: Der Teil der Initialisierung der Codes_2	14
Abbildung 20: on_readyReadStandardError() und on_readyReadStandardOutput().....	14
Abbildung 21: Der Destruktor der Codes	15
Abbildung 22: Die Funktion on_pushButton_clicked() zum Backup_1	18
Abbildung 23 Die Funktion on_pushButton_clicked() zum Backup_2.....	19
Abbildung 24: Die Funktion on_pushButton_2_clicked() zum Restore	19
Abbildung 25: Die Funktion on_pushButton_3_clicked() zum Upload.....	20

Tabellenverzeichnis

Tabelle 1: Die Slots-Funktionen.....	17
--------------------------------------	----

1. Einleitung

1.1. Aufgabenstellung

Nach der Aufgabenstellung sollen insgesamt 3 Funktionen, nämlich Backup, Restore und upload, erledigt werden. Die Funktionen sollen mit dem auf Programmiersprache C++ basierenden QT framework implementiert werden. Außerdem sollen die behandelten Dateien oder Dokumente bei Backup und Restore laut Anforderungen verschlüsselt und entschlüsselt und in das Internet hochgeladen werden.

Was Backup angeht, handelt es sich um zwei Arten von Backups, nämlich das vollständige und inkrementelle Backup. Das vollständige Backup bezieht sich auf eine ausgewählte ganze Datei und das inkrementelle Backup filtert die bearbeiteten Dateien in einem bestimmten Arbeitsverzeichnis laut einer Zeitspanne heraus.

Die fertige Benutzeroberfläche soll die Schnittstellen bieten, um das exclude-Muster in der Konfigurationsdatei anzuzeigen und zu bearbeiten. Außerdem lassen sich auch noch die Quelladresse und Zieladresse jeweils bei Backup auswählen. Bei Restore wird einfach eine gebackupte Datei in eine Zieladresse gesichert.

Eine Anzeige für die Dateiliste der vorhandenen Backups wird auch angefordert. Auch ist es nötig, ein paar wichtige Informationen wie zum Beispiel die backupte Dateigröße auf der Benutzeroberfläche anzuzeigen.

Darüber hinaus soll die Codes mit Kommandozeilen von Linux ankoppeln, weil das Erzeugnis in Linux-System laufen soll.

1.2. Motivation

Nicht nur die Kenntnisse der Algorithmen und des Rechners, sondern auch der Programmierung sind den Studenten der Fachrichtung Computer Engineering notwendig, um in der Zukunft nach einer guten und geeigneten Arbeitsstelle zu suchen.

Dec 2019	Dec 2018	Change	Programming Language	Ratings	Change
1	1		Java	17.253%	+1.32%
2	2		C	16.086%	+1.80%
3	3		Python	10.308%	+1.93%
4	4		C++	6.196%	-1.37%

Abbildung 1: Rangliste der Programmiersprachen in Tobi

Bei der kleinen Seminararbeit soll man mit der objektorientierten Programmiersprache C++ auf dem Entwicklungsplattform Qt Creator die Aufgaben erledigen. Zugleich wird mehr und weniger die Erfahrung der Projektentwicklung gesammelt. Auf dem Markt wird in den letzten Jahren die Programmiersprache Python immer mehr populär. Allerdings spiegeln einige Programmiersprachen wie zum Beispiel C++ mit Pointer ein paar Entwicklungsprinzipien des Rechner bei Software und Hardware wider. Deshalb kann mit den Erkenntnissen von C++ auch schneller und leichter andere Programmiersprachen wie Java, Python erfassen. (s. *Abbildung 1*).

Außerdem müssen die angeforderten Funktionen in Linux implementiert werden. In Codes werden die Kommandozeilen von Linux integriert. Somit kann man auch noch während der kleinen Seminararbeit das einen weiten Bereich abdeckende Betriebssystem kennenlernen.

2. Grundlagen der Entwicklung der Benutzeroberflächen in Qt

2.1. Was ist Qt?

Qt ist ein quellfreies Framework und GUI-Toolkit zur plattformübergreifenden Entwicklung von Programmen und grafischen Benutzeroberflächen auf Grundlage der Programmiersprache C++ und für eine große Zahl an Betriebssystemen bzw. Grafikplattformen wie Linux, macOS, Windows, iOS und Android erhältlich.

2.2. Kommunikation im Projekt

Qt Creator ist eine integrierte Entwicklungsumgebung speziell für Qt und reduziert stark die Entwicklungsschwierigkeiten mit den Grundklassen, dem Qt Designer und die von Qt Creator am Anfang erstellten Codes wie Main Funktion, auf die im Folgenden näher eingegangen wird.

Im Qt Creator stehen insgesamt 3 Grundklassen, nämlich QMainWindow, QWidget und QDialog, im ersten Schritt der Entwicklung zur Verfügung. QMainWindow und QDialog sind abgeleitet von QWidget. QWidget bietet einfach ein leeres Fenster. In QMainWindow werden ein paar Leisten wie die Menüleiste, Statusleiste, die Symbolleiste, das Dock-Fenster und das mittlere leere Bereich automatisch erstellt. Und QDialog bietet ein Dialogfenster, um die Interaktion zwischen Benutzern und Programmen zu ermöglichen.

Mit Qt Designer wird das Fenster einfach konstruiert und erfolgreich aufgestellt, weil die gewünschten Menüeinträge einfach auf ein Fenster ziehen. Und gleichzeitig werden die entsprechenden Codes sofort automatisch erstellt und aktualisiert.

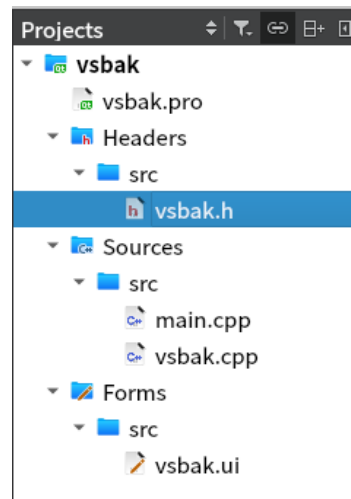


Abbildung 2: Verzeichnis des Projektes

Am Beginn eines Projektes erzeugt Qt Creator automatisch 5 Dateien: eine Main-Datei, Pro-Datei, Header-Datei, Cpp-Datei und Ui-Datei. (s. Abbildung 2).

In der Main Funktion sind zwei Objekte jeweils von der Klasse QApplication und einer der drei Grundklassen bereit. Ausschließlich mit der Methode show() wird das Fenster gezeigt.

Allerdings muss das Objekt der Klasse QApplication am Ende nach return noch die Methode exec() aufrufen. Die Methode dient dem stetigen Empfang der Nachrichten vom Benutzer.

Pro-Datei beinhaltet die relevanten Informationen, wie zum Beispiel die Version der Entwicklungsumgebung, die benutzten Module, header Datei, Quellen Dateien u.s.w. Generell wird die Datei automatisch von IDE nach der gewünschten Konfiguration erstellt. Es ist erwähnenswert, dass die entsprechenden Orte von Cpp-Datei, Main-Datei, Header-Datei und Ui-Datei auch festgelegt werden.

In Ui-Datei steht Qt Designer, nämlich ein grafisches Werkzeug zum Entwerfen und Erstellen der grafischen Benutzeroberflächen, zur Verfügung.

3. Erzeugnis

3.1. Auslegung der grafischen Arbeitsoberfläche

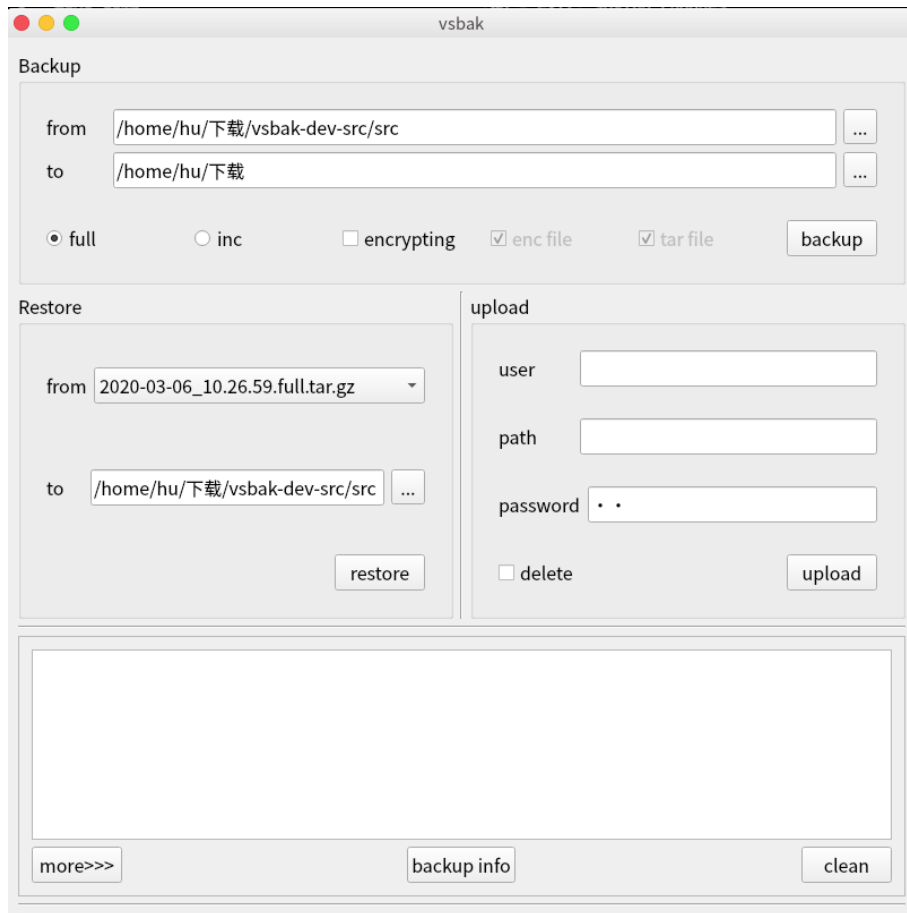


Abbildung 3: die grafische Benutzeroberfläche

Die Arbeitsoberfläche werden in 5 Arbeitsbereiche eingeteilt, auf die im Folgenden näher in den Einzelheiten eingegangen wird. (s. Abbildung 3).

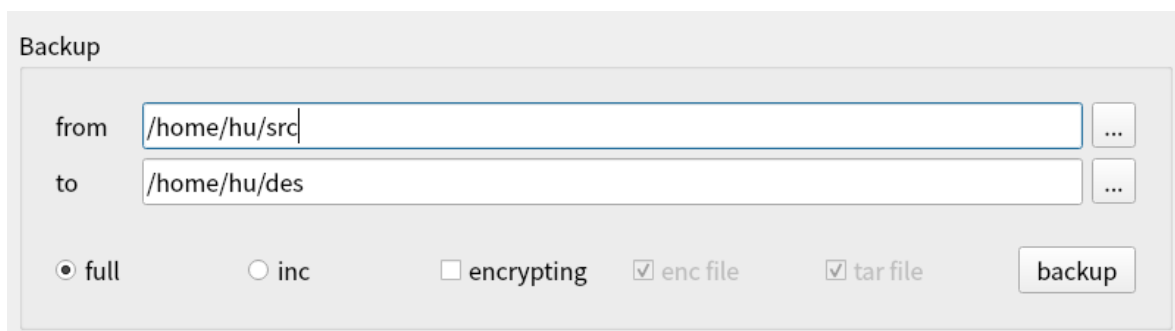
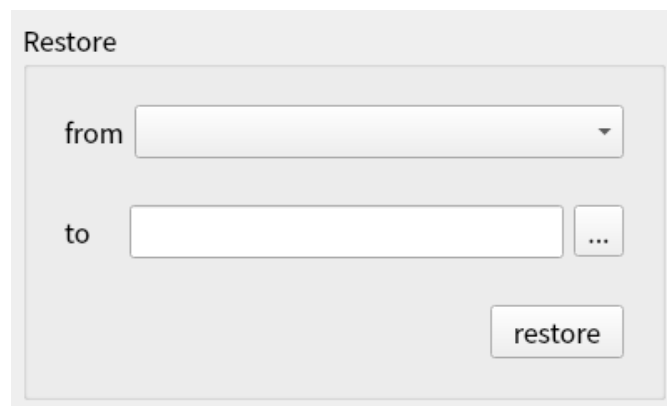


Abbildung 4: Der Bereich des Backups

Im Back-up Bereich lassen sich die Quellen- und Zieladresse bestimmen. Mit den Kombinationen der im untenstehenden 2 Radio Buttons und 3 Checkboxes erfolgen unterschiedliche Backup-Auswahl. Mit der “full” oder “inc”- Auswahl werden die Dateien komplett oder teilweise in die Zieladresse gesichert. Anschließend folgen die Verschlüsselung und Kompression. Mit dem „full“-Button wird eine ganze Datei im ausgewählten Ordner in die Zieladresse gesichert. Im Vergleich dazu ist die inkrementelle Sicherung durch die Auswahl des “inc”-Buttons zu erzielen.

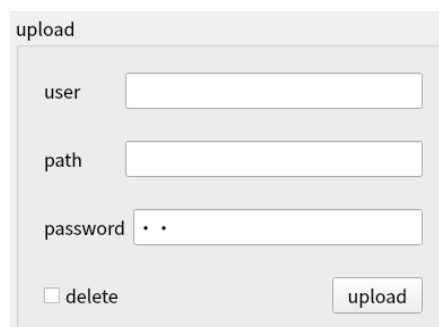
Die Verschlüsselung und Kompression lassen sich durch “encrypting” und “tar file” feststellen. Mit dem Häkchen im Checkbox des „enc file“- und „tar file“-Buttons werden die Nebendatei für Verschlüsselung und die komprimierte Datei deshalb in die Zieladresse gesichert. Man kann die “enc file” und “tar file” aber erst willkürlich entfallen oder nicht, nachdem das „encrypting“- Button erstmals ausgewählt werden, weil die beiden nach der Initialisierung angekreuzt werden. (s. Abbildung 4).



The image shows a 'Restore' dialog box. It has a 'from' dropdown menu and a 'to' text input field with a browse button (...). There is a 'restore' button at the bottom right.

Abbildung 5: Der Bereich des Restores

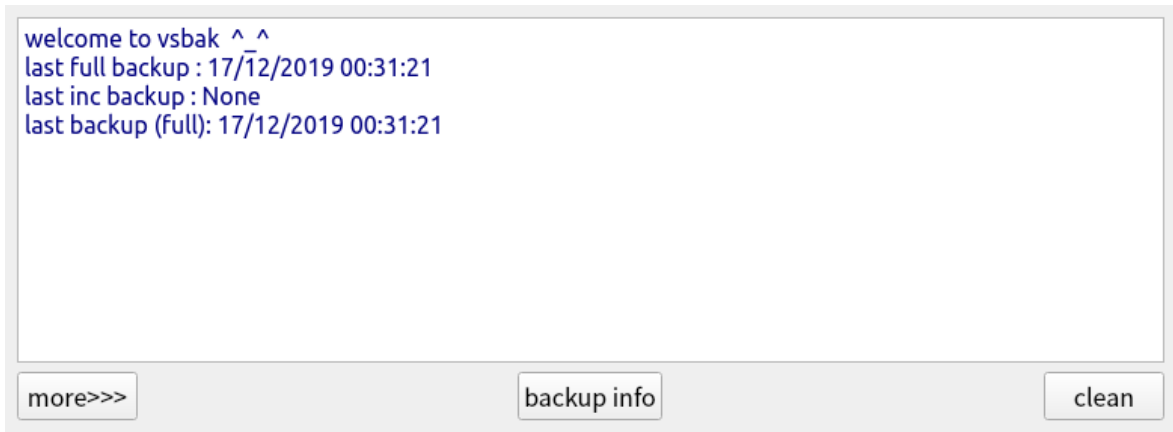
Im Restore Bereich wird die bevorzugte Datei von der im Back-up bestimmten Zieladresse in eine neue Zieladresse wiederhergestellt. (s. Abbildung 5).



The image shows an 'upload' dialog box. It has fields for 'user', 'path', and 'password'. There is a 'delete' checkbox and an 'upload' button.

Abbildung 6: Der Bereich des Uploads

Ausschließlich 3 Eingabefelder für user, path und Passwort werden in Upload-Bereich integriert, um die angeforderte Datei in das Internet hochzuladen. Das delete Button wird benutzt, um die uploadeten Dateien zu löschen. (s. *Abbildung 6*)



welcome to vsbak ^ ^
last full backup : 17/12/2019 00:31:21
last inc backup : None
last backup (full): 17/12/2019 00:31:21

more>>> backup info clean

Abbildung 7: Der Bereich des Anzeigefeldes

Der Anzeigebereich zeigt das temporäre Protokoll nach jedem Schritt oder einen Hinweis auf dem nächsten Schritt. Das „clean“-Button räumt den Anzeigebereich. (s. *Abbildung 7*).



edit config file

gpg_key

next write show all

edit config file

exclude

next write show all

Abbildung 8: Der Bereich der Erweiterung

Mit einem Klicken auf dem „more“-Button öffnet sich eine Erweiterung, in der man in einem Eingabefeld die „exclude“-Daten oder gpg-key der Konfigurationsdatei editieren kann. „Next“-Button sorgt für die Wechsel der zweiten Edierungsmodi. Nach dem Klicken auf dem „write“-Button wird die Konfigurationsdatei nach der Eingabe von neuen „exclude“-Daten oder gewünschtem gpg-Key aktualisiert. „Show all“- Button schreibt den ganzen Inhalt der Konfigurationsdatei in das Anzeigefenster. (s. *Abbildung 8*).

3.2. Implementierung der Funktionen

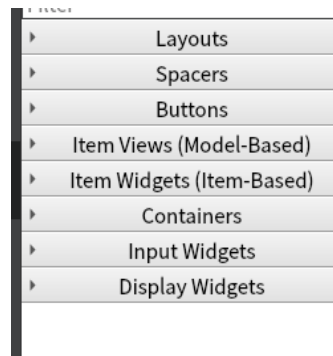


Abbildung 9: Das Verzeichnis der Widgets

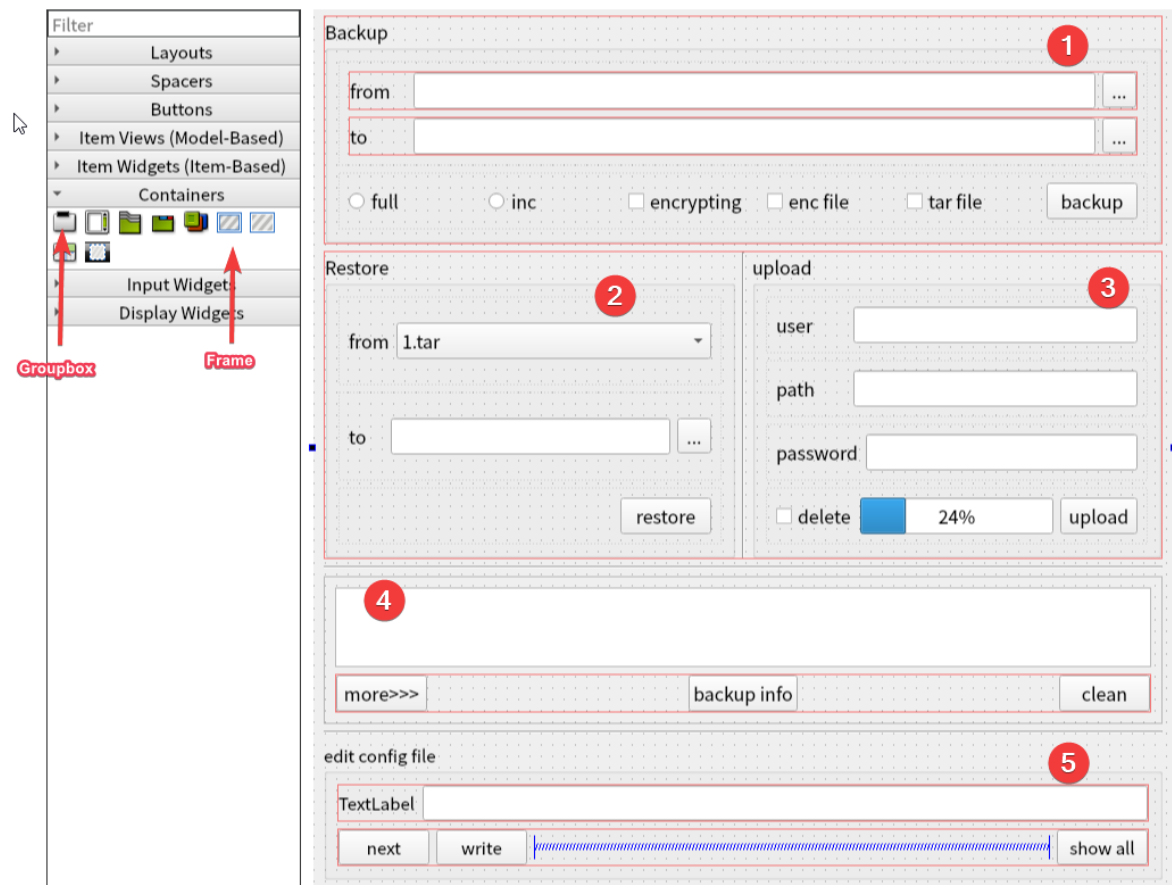


Abbildung 10: Die Auslegung der grafischen Benutzeroberfläche

In UI-Designer werden Arbeitsoberflächen mit den zur Verfügung gestellten Widgets gestaltet, die nach Wunsch horizontal oder vertikal zusammengehört werden können. Am

Beispiel werden die zweite und dritte Zone horizontal kombiniert. Im Wesentlichen werden einzelne Widgets von einer Klasse Q-Widget abgeleitet. (s. Abbildung 9 und 9).

Wie in Kapitel 3.1 beschrieben, gibt es fünf Zonen in der Arbeitsoberfläche. Alle ausgewählten Widgets werden in vier Groupboxes und ein Frame (die fünfte Zone) eingelegt.

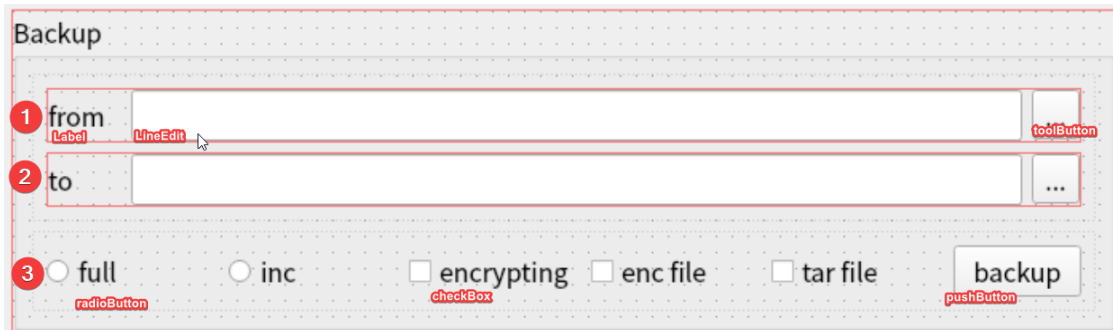


Abbildung 11: Die Auslegung des Bereichs des Backups

Der erste Bereich besteht aus drei Zeilen, deren Widgets jeweils horizontal zusammengehören, und 6 Arten von Widgets, die folgendermaßen zur Beschreibung aufgelistet werden. (s. Abbildung 11).

1. Label - Im „Label“-Widget werden eine ersehnte Schrift als ein Label in einer bestimmten Stelle festgelegt. Am Beispiel in dem Programm weist das Label „from“ darauf hin, wofür die Schriften, nämlich die Quelladresse, im „LineEdit“-Widget steht.
2. LineEdit - Im „LineEdit“-Widget wird die von dem „ToolButton“-Widget ausgewählte Quelladresse in dem Programm angezeigt.
3. ToolButton- Ein „ToolButton“-Widget stellt ein Dialogfenster zur Auswahl der gewünschten Quelladresse im Laufe des Programms zur Verfügung.
4. RadioButton- Jedes „RadioButton“-Widget steht für eine funktionale Auswahl wie zum Beispiel das vollständige und inkrementelle Backup in dem Programm.
5. CheckBox- Ein „CheckBox“-Widget, ist ausschließlich ein zusätzliche Auswahl aber nicht notwendig. In dem Programm werden „enc file“ und „tar file“ am Anfang automatisch angekreuzt, bevor die Auswahl „encrypting“ angekreuzt wird. Nachdem die Auswahl „encrypting“ angekreuzt wird, können „enc file“ oder „tar file“ entfallen lassen.

6. PushButton- Mit einem Klick auf dem „PushButton“-Widget wird eine Funktion wie zum Beispiel Backup in diesem Projekt im Laufe des Programms nach den vorhergehenden Einstellungen dementsprechend aktiviert.



Abbildung 12: Die Auslegung des Bereichs des Restores

Die Anordnung im Restore Bereich ist im ähnlich wie der erste Bereich. Anschließend kommt ein neues Widget, und zwar „ComBox“-Widget , vor, mit dem eine backupte Datei oder backuptes Dokument Zieladresse ausgewählt wird, um in irgendeinem Ort wiederherzustellen. (s. Abbildung 12).

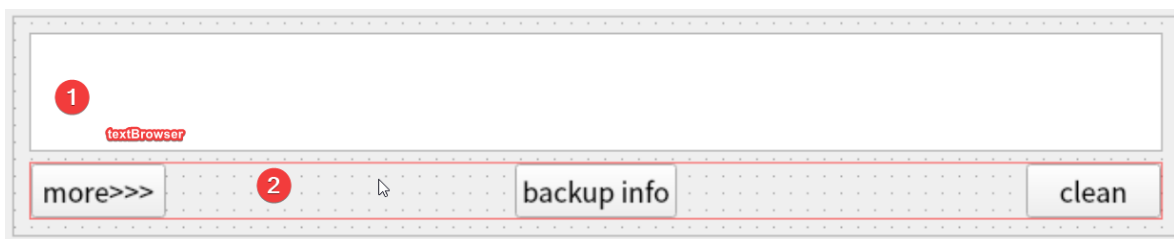


Abbildung 13: Die Auslegung des Bereichs des Anzeigefeldes

Im Bereich von Upload steht ein neues Widget, nämlich „Progressbar“ als ein Fortschrittsbalken und die Anordnung ist gleich wie der zweite Bereich. Im vierten Bereich wird zwei Zeilen, deren Komponente jeweils ein „Textbrowser“-Widget und drei „PushButton“-Widgets sind, gestaltet. Das „Textbrowser“-Widget spiegeln die detaillierten Hinweise auf die Bedienung in Schriften wider. (s. Abbildung 13).

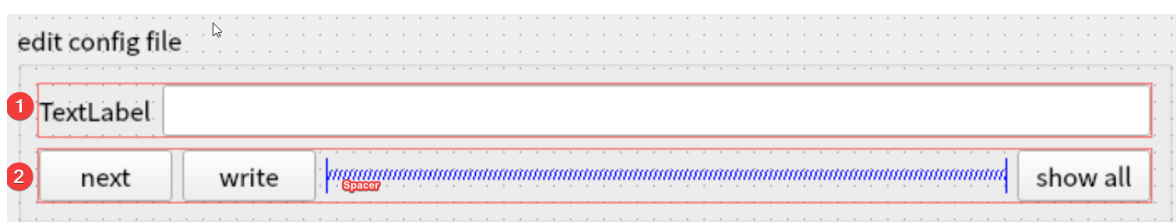


Abbildung 14: Die Auslegung der Erweiterung

In dem letzten Bereich wird nur ein neues Widget, also „Spacer“-Widget, zwischen zwei PushButtons gelegt, um einen richtigen Abstand zu halten. (s. *Abbildung 14*).

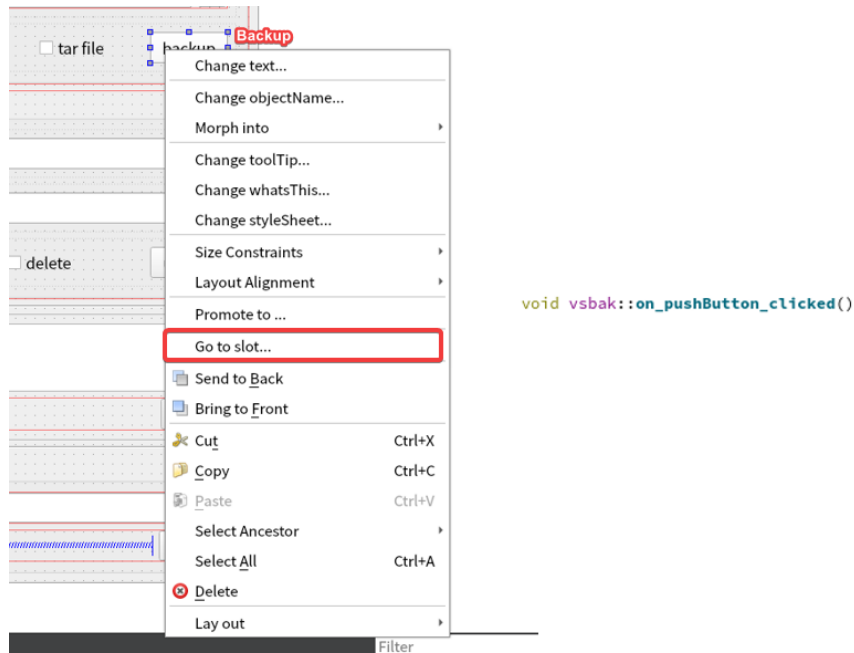


Abbildung 15: Funktionsdeklarationen von Slots

Mit den angeordneten Layouts lassen sich die Funktionsdeklarationen von Slots in der Header-Datei und die Funktionsdefinitionen ohne Inhalte in der Cpp-Datei durch den Klick auf “go to slot” leicht sofort erstellt. (s. *Abbildung 15*).

4. Codeanalyse

```
private slots:
    void on_readyReadStandardError();
    void on_readyReadStandardOutput();
    void on_uploadProgress(qint64 readBytes, qint64 totalBytes);
    void on_toolButton_clicked();
    void on_toolButton_2_clicked();
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void on_checkBox_clicked();
    void on_lineEdit_3_editingFinished();
    void on_lineEdit_4_editingFinished();
    void on_pushButton_3_clicked();
    void on_toolButton_3_clicked();
    void on_lineEdit_editingFinished();
    void on_lineEdit_2_editingFinished();
    void on_lineEdit_5_editingFinished();
    void on_pushButton_6_clicked();
    void on_pushButton_5_clicked();
    void on_pushButton_4_clicked();
    void on_pushButton_9_clicked();
    void on_pushButton_7_clicked();
    void on_pushButton_8_clicked();

private:
    Ui::vsbak *ui;
    QProcess *cmd;

    QString config_file_path;
    QString dir_src;
    QString dir_dest;
    QString dir_restore;
    QString gpg_key;
    QString arc_user;
    QString arc_path;
    QString exclude_from;
    QDateTime lastBackupTime;

    bool signal_disp_output;
    bool signal_disp_error;

    QString backupInfo();
    void outputInfo(QString out, QColor color = QColor(Qt::darkBlue));
    void configFileWrite(QString path, QString str);
    void executeCommand(QString line, QString command, QStringList argument = QStringList(), bool sig_out = true, bool sig_err = true);
```

Abbildung 16: Header-Datei

In Kapitel 2.2 werden all die erstellten Dateien diskutiert. Im Folgenden werden die Header-Datei und Cpp-datei detailliert analysiert.

In der Header Datei gibt es nur 2 Teile, die durch die Zugriffskontrollen private slot und private begrenzt werden. Im ersten Bereich werden die allen Funktionsdeklarationen, wie in Kapitel 4.1 erwähnt, automatisch erstellt. Des Weiteren werden die Deklarationen der manuell erstellten Funktionen und Variablen in Cpp-Datei manuell im zweiten Teil erstellt. (s. Abbildung 16).



Abbildung 17: Der Speicherort der Konfigurationsdatei

```
#define CONFIG_FILE_NAME "vsbak_config.ini"

vsbak::vsbak(QWidget *parent) : QWidget(parent), ui(new Ui::vsbak)
{
    ui->setupUi(this);

    // the config file is ~/.config/vsbak/vsbak_config.ini
    config_file_path = QDir::homePath() + "/.config/vsbak/" + CONFIG_FILE_NAME;
    //config_file_path = QApplication::applicationDirPath() + '/' + CONFIG_FILE_NAME;

    // read config.ini.
    QFile cfile(config_file_path);
    if(cfile.exists()){
        QSettings *configIniRead = new QSettings(config_file_path, QSettings::IniFormat);
        dir_src = configIniRead->value("/main/bak_src").toString();
        dir_dest = configIniRead->value("/main/bak_dest").toString();
        dir_restore = dir_src;
        arc_user = configIniRead->value("/main/arc_user").toString();
        arc_path = configIniRead->value("/main/arc_path").toString();
        gpg_key = configIniRead->value("/main/gpg_key").toString();
        exclude_from = configIniRead->value("exclude/exclude").toString();
        delete configIniRead;
    }else{
        QString err = QString("No file : %1").arg(config_file_path);
        exclude_from = "--exclude *.iso --exclude .gvfs/** --exclude .thumbnails/** --exclude .trash/** --exclude .Trash/** "
            "--exclude *-gitsvn/** --exclude *-cvs/** --exclude *-svn/** --exclude *-git/** --exclude *-sf/** --exclude *.vc --exclude *.tc "
            "--exclude .cache/** --exclude .ccache/** --exclude ./Downloads/** --exclude ./wine/** --exclude ./Documents/Dropbox/**";
        configFileWrite("exclude/exclude",exclude_from);
        outputInfo(err,QColor(Qt::darkRed));
    }
}
```

Abbildung 18: Der Teil der Initialisierung der Codes_1

C++-Datei wird in vier Gruppen, nämlich Initialisierung, Destruktor, Slots-Funktionen, und ein paar weitere selbst definierte Funktionen, zerlegt. (s. Abbildung 17). Vor der Initialisierung wird eine Konfigurationsdatei, die wegen der Initialisierung in einem bestimmten Ort erstellt wird, definiert. In der Initialisierung wird aber durch if-else-Anweisung auch noch bestimmt, ob ein Dokument namens vsbak_config.ini. existiert. Durch den Zugriff auf die Konfigurationsdatei werden den definierten lokalen Variablen ein paar relevante Informationen wie.z.B. Zieladresse, Quelladresse u.s.w. zugewiesen. Ohne Konfigurationsdatei würde eine Warnung und noch eine Sequenz von Informationen, die die exclude_from Variable enthält, auf der Benutzeroberfläche angegeben. (s. Abbildung 18)

```

// init bash terminal
cmd = new QProcess(this);
cmd->setWorkingDirectory(dir_src);

connect(cmd, &QProcess::readyReadStandardError, this, &vsbak::on_readyReadStandardError);
connect(cmd, &QProcess::readyReadStandardOutput, this, &vsbak::on_readyReadStandardOutput);

// hide groupbutton
ui->label_4->setText("gpg_key");
ui->lineEdit_6->setText(gpg_key);
ui->groupBox_edit_config->hide();

// init lineEdit
ui->lineEdit->setText(dir_src);
ui->lineEdit_2->setText(dir_dest);
ui->lineEdit_3->setText(arc_user);
ui->lineEdit_4->setText(arc_path);
ui->lineEdit_5->setText(dir_restore);
ui->lineEdit_7->setEchoMode(QLineEdit::PasswordEchoOnEdit);

// init radiobutton: choice full
ui->radioButton->setChecked(true);
ui->checkBox->setChecked(false);
ui->checkBox_2->setChecked(true);
ui->checkBox_3->setChecked(true);
ui->checkBox_2->setEnabled(false);
ui->checkBox_3->setEnabled(false);

// hide progress bar
ui->progressBar->setVisible(false);

// get backup info
backupInfo();

// output info
outputInfo("welcome to vsbak  ^_^\\n");

```

Abbildung 19: Der Teil der Initialisierung der Codes_2

```

void vsbak::on_readyReadStandardError()
{
    if(signal_disp_error){
        ui->textBrowser->moveCursor(QTextCursor::End);
        ui->textBrowser->setTextColor(QColor(Qt::red));
        ui->textBrowser->insertPlainText(cmd->readAllStandardError().data());
    }
}

void vsbak::on_readyReadStandardOutput()
{
    if(signal_disp_output){
        ui->textBrowser->moveCursor(QTextCursor::End);
        ui->textBrowser->setTextColor(QColor(Qt::black));
        ui->textBrowser->insertPlainText(cmd->readAllStandardOutput().data());
    }
}

```

Abbildung 20: on_readyReadStandardError() und on_readyReadStandardOutput()

Darüber hinaus wird ein Objekt der QProcess-Klasse und damit die Quelladresse als die Arbeitsadresse definiert. Des Weiteren werden zwei Slot-Signal-Paare durch Aufrufe der Connect-Funktion zum Empfang der ausgegebenen Fehler und Ergebnisse des Terminals realisiert, damit die Kommunikation mit Linux-System erfolgreich läuft und sich die Fehler und Ergebnisse auf der grafischen Benutzeroberfläche in TextBrowser mit den Funktionen on_readyReadStandardError() und on_readyReadStandardOutput() anzeigen lassen. Und

was wie z.B. Labels, der anfängliche Zustand des Checkboxes u.s.w. auf der grafischen Benutzeroberfläche festgelegt wird, wird auch dann initialisiert. Am Ende der Initialisierung wird zweimalige Aufrufe einer selbstdefinierten Funktion geschrieben, um die Backup-Informationen wie z.B. die Informationen einer gebackupten Datei und einen Begrüßungssatz auszugeben. (s. *Abbildung 19: Der Teil der Initialisierung der Codes_2 und 20*)

```
vsbak::~vsbak()
{
    delete ui;
    /*
     *if the process is running, terminate() attempts to terminate the process.
     * waitForFinished() returns true if the process finished; otherwise returns false.
     */

    if(cmd->state() == QProcess::Running){
        cmd->terminate();
        cmd->waitForFinished();
    }
}
```

Abbildung 21: Der Destruktor der Codes

In dem Destruktor wird eine if-Anweisung geschrieben, um einen Prozess völlig zu beenden. (s. *Abbildung 21*)

Die in der folgenden Tabelle aufgelisteten Slots-Funktionen werden als Hilfsfunktionen zur Realisierung von Backup, Restore und Upload und einige zusätzliche selbstdefinierte Hilfsfunktionen zur Erleichterung der Bedienung auf der grafischen Benutzeroberfläche aufgebaut. Folglich werden die Hilfsfunktionen und Selbstdefinierten Funktionen und die dazugehörigen Beschreibungen in einer Tabelle aufgeführt.

on_uploadProgress()	Ein Hinweis auf den Uploadsprozess
on_lineEdit_editingFinished()	Ein Hinweis auf die erfolgreiche oder erfolglose Änderung der Quelladresse.

on_lineEdit_2_editingFinished()	Ein Hinweis auf die erfolgreiche oder erfolglose Änderung der Zieladresse zum Backup
on_lineEdit_3_editingFinished()	Ein Hinweis auf die erfolgreiche oder erfolglose Änderung der Information von arc_user
on_lineEdit_4_editingFinished()	Ein Hinweis auf die erfolgreiche oder erfolglose Änderung der Information von arc_path
on_lineEdit_5_editingFinished()	Ein Hinweis auf die erfolgreiche oder erfolglose Änderung der Adresse zum Restore
on_pushButton_6_clicked()	Löschen der gezeigten Inhalte in TestBrowser
on_pushButton_5_clicked	Anzeige der detaillierten Informationen von Backup
on_pushButton_4_clicked()	Öffnen oder Schließen der Erweiterung
on_pushButton_9_clicked()	Anzeige der Inhalte der Konfigurationsdatei.
on_pushButton_7_clicked()	Wechsel der Funktionen der Erweiterung zur Änderung des Schlüsselwortes oder der Inhalte in der exclude from-Datei
on_pushButton_8_clicked()	Ein Hinweis auf die erfolgreiche Änderung des Schlüsselwortes oder der Inhalte in der exclude from-Datei
on_toolButton_clicked()	Änderung der Quelladresse

on_toolButton_2_clicked()	Änderung der Zieladresse zum Backup
on_toolButton_3_clicked()	Änderung der Adresse zum Restore
on_checkBox_clicked()	Initialisierung der Checkboxes
executeCommand(QString line, QString command, QStringList argument ,bool sig_out, bool sig_err)	Ausführung eines formulierten Linux-Kommandos
outputInfo(QString out, QColor color)	Ausgabe der Informationen in den TextBrowser
configFileWrite(QString path, QString str)	Änderung der Inhalte in der Konfigurationsdatei

Tabelle 1: Die Slots-Funktionen

Im Weiteren werden die im Programm eine große Rolle spielenden Funktionen als die Hauptfunktionen zur Realisierung von Backup, Restore und Upload analysiert.

on_pushButton_clicked() (zum Backup)

on_pushButton_2_clicked() (zum Restore)

on_pushButton_3_clicked() (zum Upload)

Laut der Aufgabenstellung stellt die Funktion on_pushButton_clicked() die Auswahlen wie die Verschlüsselung, das inkrementelle und vollständige Backup zur Verfügung. Nach dem Gedankengang wird die Linux-Kommandos zuerst formuliert und bei Bedarf dann ausgeführt werden.

Wie in Kapitel 3.1 erwähnt, können die beiden zusätzlichen Auswahlen, nämlich „enc file“ und „tar file“, aber erst willkürlich entfallen oder nicht, nachdem das „encrypting“-Button erstmals ausgewählt werden, weil die beiden nach der Initialisierung angekreuzt werden. Nach der Bestimmung oder Einstellung auf der grafischen Benutzeroberfläche wird im Programm zuerst die Kompression dann die Verschlüsselung ausgeführt.

```

/*****
 * function backup
 * push button
 *****/
void vsbak::on_pushButton_clicked()
{
    QDateTime current_date_time = QDateTime::currentDateTime();
    QString targz, cmd_find, cmd_tar;

    // if choice full backup
    if(ui->radioButton->isChecked()){
        cmd_find = "find ./ -type f | grep -v 'Permission denied' > TEPLIST";
        targz = current_date_time.toString("yyyy-MM-dd_hh.mm.ss") + ".full.tar.gz";
    }else{
        // if choice inc backup
        if(lastBackupTime.isNull())
            cmd_find = "find ./ -type f | grep -v 'Permission denied' > TEPLIST";
        else{
            int minDiff = (current_date_time.toTime_t() - lastBackupTime.toTime_t()) / 60 + 1;
            cmd_find = QString("find ./ -mmin -%1 -type f | grep -v 'Permission denied' > TEPLIST").arg(minDiff);
        }
        targz = current_date_time.toString("yyyy-MM-dd_hh.mm.ss") + ".inc.tar.gz";
    }

    QStringList cmd_pipe;
    cmd_pipe << "-c" << cmd_find;
    executeCommand(NULL, "bash", cmd_pipe) ;

    cmd_tar = QString("tar --verbose --totals --exclude-vcs %1 --files-from=TEPLIST -czf %2/%3").arg(exclude_from).arg(dir_dest).arg(targz);
    executeCommand("backup", cmd_tar);

    executeCommand(NULL, "rm TEPLIST");
}

```

Abbildung 22: Die Funktion `on_pushButton_clicked()` zum Backup_I

In dieser funktion ist ein aktueller Zeitpunkt am Anfang beim Backup bereit. Anschließend wird die Auswahl zwischen dem inkrementellen und vollständigen backup durch eine if-else-Anweisung realisiert. In dem Zweig des sowohl vollständigen als auch inkrementellen backups wird einer Variable ein Linux Kommando in QString zugewiesen. Das Kommando sorgt dafür, dass alle Dateien mit dem Zugriffserlaubnis in einem Verzeichnis einer Datei und den dazugehörigen Unterverzeichnisse durch das Linux Kommando bei dem vollständigen Backup ausgewählt und in einer Datei namens „TEPLIST“ aufgelistet werden. Aber im Zweig des inkrementellen Backups werden nur die zuletzt nicht korrigierten oder zugegriffenen Dateien mit Zugriffserlaubnis in „TEPLIST“ aufgelistet. Im Abschluss des Teils wird das Kommando ausgeführt. (s. Abbildung 22)

Darüber hinaus wird einer Variable ein Linux Kommando in QString zur Kompression einer Datei zugewiesen. Es ist erwähnenswert, dass die in „TEPLIST“ aufgelisteten Dateien ausgeschlossen sind, wenn Sie dem Muster in `exclude-from` entsprechen.

```

/* Encrypting */
if(ui->checkBox->isChecked()){
    QString cmd_isKey = "gpg -k " + gpg_key;
    executeCommand(NULL, cmd_isKey, QStringList(), false, false); // find or the key exists
    QString cmd_enc;
    QString out = QString::fromLocal8Bit(cmd->readAllStandardOutput());
    if(out.contains(QString("<%1>").arg(gpg_key))) {
        outputInfo(out, QColor(Qt::black));
        cmd_enc = QString("gpg -e -r %1 -o %2/%3.gpg %2/%3").arg(gpg_key).arg(dir_dest).arg(targz);
    }
    else {
        cmd_enc = QString("gpg -c -o %1/%2.gpg %1/%2").arg(dir_dest).arg(targz);
    }
    cmd->close();
    executeCommand(NULL, cmd_enc); // encrypting the tar file

    if(false == ui->checkBox_2->isChecked()){
        executeCommand(NULL, QString("rm %1/%2.gpg").arg(dir_dest).arg(targz));
        outputInfo(QString("delete %1/%2.gpg\n").arg(dir_dest).arg(targz));
    }

    if(false == ui->checkBox_3->isChecked()){
        executeCommand(NULL, QString("rm %1/%2").arg(dir_dest).arg(targz));
        outputInfo(QString("delete %1/%2\n").arg(dir_dest).arg(targz));
    }
}

outputInfo("backup finished!\n");
backupInfo();

```

Abbildung 23 Die Funktion `on_pushButton_clicked()` zum Backup_2

In Anschluss an die Codes der Kompression sind die der Verschlüsselung. Am Anfang wird überprüft, ob ein Passwort existiert oder nicht. Wenn ein Passwort vorhanden ist, kann eine Datei direkt verschlüsselt. Wenn kein Passwort, kann eine Datei durch das symmetrische Verschlüsselungsverfahren verschlüsselt. Das Prozess wird mit einer if-else-Anweisung realisiert. Das entsprechende Kommando wird auch einer Variable in QString zugewiesen.

Am Ende der Funktion wird mit If-else-Anweisung überprüft, ob die Kompression oder Verschlüsselung ausgeführt werden soll. (s.Abbildung 24)

```

/*****
 * function restore
 * push button 2
 *****/
void vsbak::on_pushButton_2_clicked()
{
    QString targz = ui->comboBox->currentText();

    if (targz.endsWith(".tar.gz.gpg")){
        targz = targz.remove(".gpg");
        QFile tarFile(QString("%1/%2").arg(dir_dest).arg(targz));
        if(!tarFile.exists()){ // check if the tar file exists.if not ,do below.
            // find if key and gen exists.
            QString cmd_enc;
            QString cmd_isKey = "gpg -k " + gpg_key;
            executeCommand(NULL, cmd_isKey, QStringList(), false, false); // find or the key exists
            QString out = QString::fromLocal8Bit(cmd->readAllStandardOutput());
            if(out.contains(QString("<%1>").arg(gpg_key))) {
                outputInfo(out, QColor(Qt::black));
                cmd_enc = QString("gpg -d -r %1 -o %2/%3 %2/%3.gpg").arg(gpg_key).arg(dir_dest).arg(targz);
            } else cmd_enc = QString("gpg -d -o %1/%2 %1/%2.gpg").arg(dir_dest).arg(targz);
            cmd->close();
            executeCommand(NULL, cmd_enc);

            QString cmd_restore = QString("tar --verbose --totals -xzf %1/%2 -C %3").arg(dir_dest).arg(targz).arg(dir_restore);
            executeCommand("execute restore", cmd_restore);
            outputInfo("restore finished!\n");
            QString cmd_rm = QString("rm %1/%2").arg(dir_dest).arg(targz);
            executeCommand(NULL, cmd_rm);
            return;
        }
    }

    QString cmd_restore = QString("tar --verbose --totals -xzf %1/%2 -C %3").arg(dir_dest).arg(targz).arg(dir_restore);
    executeCommand("execute restore", cmd_restore);
    outputInfo("restore finished!\n");
}

```

Abbildung 24: Die Funktion `on_pushButton_2_clicked()` zum Restore

In der Funktion `on_pushButton_2_clicked()` wird erstmals bestimmt, ob eine Datei verschlüsselt. Dann wird die Datei dekomprimiert. Der Prozess ist genau umgekehrt zu dem von der Funktion `on_pushButton_clicked()`.

```

* function upload
* push button 3
* lftp -e "put $TARGZ_ENC -o $ARCPATH$TARGZ_ENC_BASE && quit" -u $ARCUSER archiv.luis.uni-hannover.de
*****
void vsbak::on_pushButton_3_clicked()
{
    QString ftp_passwd = ui->lineEdit_7->text();
    if(ftp_passwd.isEmpty()){
        outputInfo("Error: please input password!\n",QColor(Qt::red));
        return;
    }
    // else ui->lineEdit_7->clear(); // clear password for security
    QString upload_target = ui->comboBox->currentText();
    QDateTime current_date_time = QDateTime::currentDateTime();
    QString work_path = cmd->workingDirectory();
    outputInfo(QString("%1 %2: upload %3\n").arg(work_path).arg(current_date_time.toString("yyyy/MM/dd hh:mm:ss")).arg(upload_target),QColor(Qt::darkGreen));
    /* set archive */
    QUrl url;
    url.setScheme("ftp");
    url.setUserName(arc_user);
    url.setPassword(ftp_passwd);
    url.setHost("archiv.luis.uni-hannover.de");
    url.setPort(21);
    url.setPath(QString("%1/%2").arg(arc_path).arg(upload_target));
    /* read date */
    QFile file(QString("%1/%2").arg(dir_dest).arg(upload_target));
    file.open(QIODevice::ReadOnly);
    QByteArray data = file.readAll();
    file.close();
    /* upload */
    QNetworkAccessManager manager;
    QNetworkRequest request(url);
    QNetworkReply *reply = manager.put(request, data);

    ui->progressBar->show();
    ui->progressBar->setValue(0);
    ui->progressBar->setMaximum(100);

    /* wait for upload finish */
    QEventLoop eventLoop;
    connect(reply, SIGNAL(finished()), &eventLoop, SLOT(quit()));
    connect(reply, SIGNAL(uploadProgress(qint64,qint64)), this, SLOT(on_uploadProgress(qint64, qint64)));
    eventLoop.exec();

    if(reply->error() != QNetworkReply::NoError){
        ui->progressBar->hide();
        outputInfo(QString("Error: %1\n").arg(reply->errorString()), QColor(Qt::red));
    }else {
        outputInfo("upload succeed!\n");
        /* delete tar file if need */
        if(ui->checkBox_4->isChecked()){
            QString cmd_rm = QString("rm %1").arg(upload_target);
            executeCommand(NULL,cmd_rm);
            outputInfo(QString("Delete: %1\n").arg(upload_target));
            backupInfo();
        }
    }
}

```

Abbildung 25: Die Funktion `on_pushButton_3_clicked()` zum Upload

Das Dateiübertragungsprotokoll (File Transfer Protocol) wird auf die Funktion `on_pushButton_3_clicked()` zur Realisierung von Upload angewandt. Zum Beginn wird bestimmt, ob das Passwort eingegeben wird. Anschließend werden die grundsätzlichen Informationen wie die aktuelle Uploadzeit, das dazugehörige Arbeitsverzeichnis und die uploadete Datei auf der grafischen Benutzeroberfläche angezeigt. Dann werden die angeforderten Elemente zum Upload mit dem Dateiübertragungsprotokoll Schritte für Schritte konfiguriert. Der Hostname lautet „archiv.luis.uni-Hannover.de“. Außerdem wird ein Fortschrittsbalken durch Aufrufe zweier Connect-Funktionen verwirklicht. (s.Abbildung 25)

5. Fazit

In dieser kleinen Seminararbeit haben wir beiden tatsächlich nicht gut zusammengearbeitet.

Ohne gute Grundkenntnisse der Programmierung hilft mein Partner mir immer. Und gleichzeitig habe ich mehr und weniger die Erfahrungen der Programmierung gesammelt.

Jedes Mal, nach dem Treffen muss ich noch selber ohne Hilfe die gelernten Kenntnisse der Programmierung wiederholen.

Es wird auch herausgefunden, dass es mir solche Erfahrungen und Kenntnisse fehlt. Deshalb muss ich als ein Student von Computer Engineering mehr Zeit in die Programmierung investieren.

