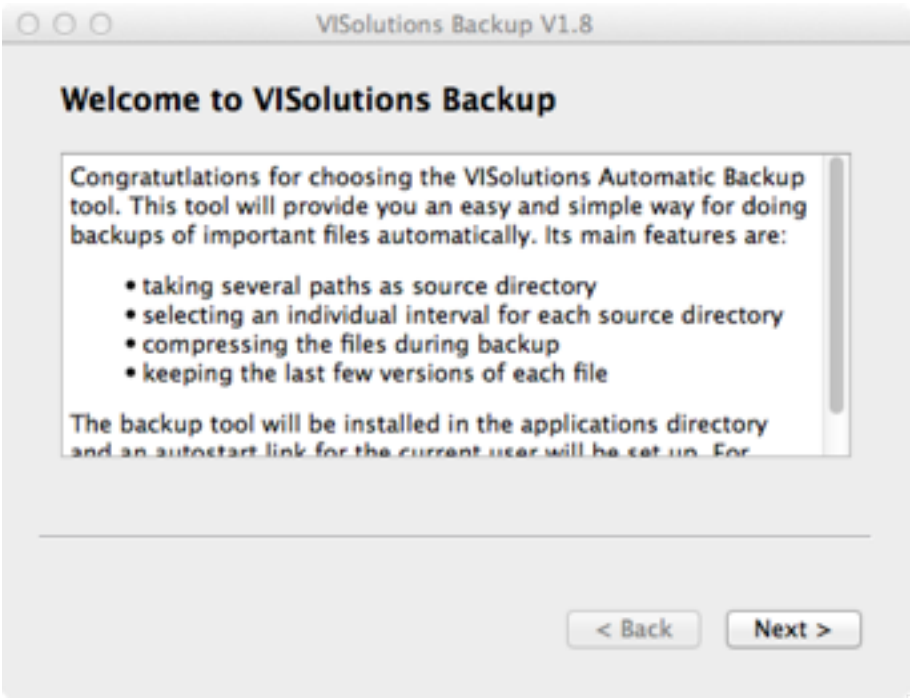
QtInstall Documentation

Version 1.8



the welcome page of the „Automatic Backup“ Installer, written by the same author.

in January, 2013

(C) Valentin Illich, Valentin.Illich@web.de

1. Table of Contents

1. Introduction

QtInstall is a general purpose installer / deinstaller tool which can be used to put any QT application onto the hard disc of a computer. It provides following features:

1. copying as many files as you like in a given destination directory. When updating an existing application, only newer files are copied.
2. creating as many QSettings entries as you like for your application
3. creating application start up links on the desktop or the auto start menu on Mac OS X Leopard or Windows
4. creating special registry settings for finding shared libraries on windows - especially QT itself.
5. providing the end user with an intuitive and straightforward user interface.

QtInstall is based upon the QWizard class which gives the fundamental user interface for creating simple installers. Since most installation processes require almost the same functionality like packing all information in one single package file, providing the user with some standard dialogs like Welcome, License (optional), Installation Progress, Completion and so on, QtInstall does these things around QWizard in a simple and handy way. The user interface allows following steps:

- Welcome Page: This page introduces the product and explains some details.
- License Page (optional): This page displays a license text (e.g. the GPL) and allows continuing only after accepting the license.
- Install Page: This page displays a progress bar which is filled up during installation.
- Complete Page: This page displays a summary of the installation progress. If some errors occurred during installation, it allows the user to view an installation protocol.

QtInstall is self-containing, that means you may create and use (that means install) packages of your own with the same application. QtInstall does not depend on any special libraries except the standard system libraries which belong to the operating system. The package file format is a binary tagged format which is fully documented in the Appendix.

QtInstall is built with a static version of QT, so it is independent of the underlying system - you will need just a cross compile for a new target system. Version 1.0 is fully tested with Windows XP, Windows 7 (Trial), Mac OS 10.5.x. All Pictures in this handbook are taken from a Mac OS system but will be similar according to the target windows style.

2. How To Use QtInstall

QtInstall can be used in the simplest thinkable way. Since the Qt library which is used by QtInstall is compiled in statically, all you need is the QtInstall application itself. It may be copied onto the target hard disc with an entry in the start menu, or it can reside as a copy with each installation package you want to provide.

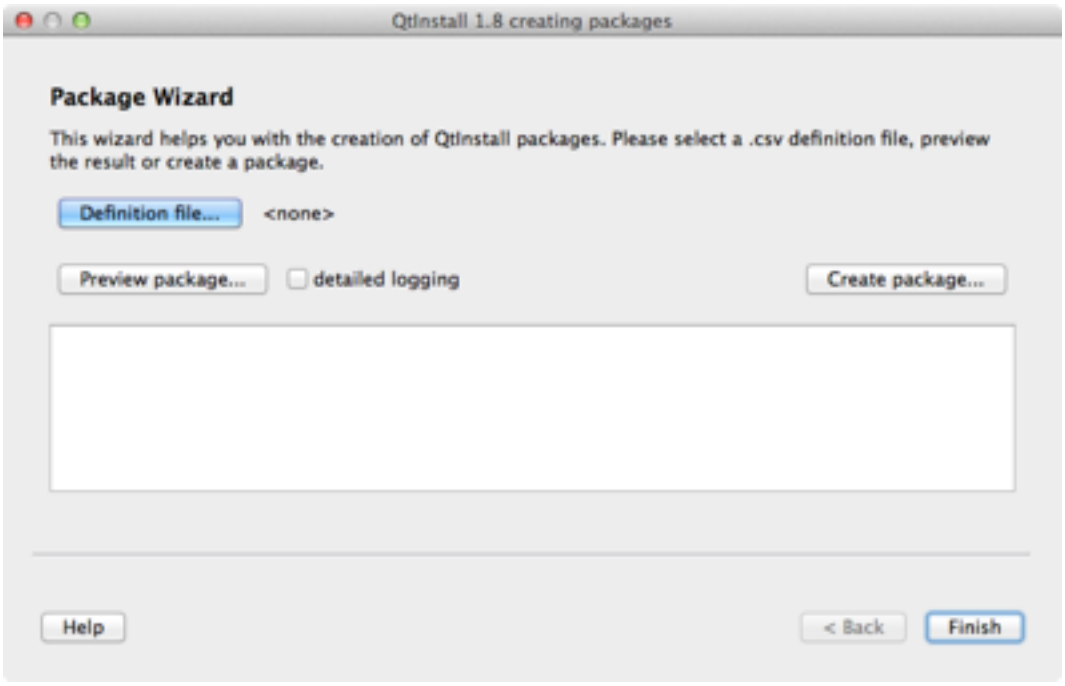
This kind of operation allows you to easily provide the three most frequent types of installing or removing your products from a destination computer. You may provide the QtInstall application itself and any of your applications as package files for example as download link, or you may provide both (the installer itself and your product package) on a media like CD or USB stick. The last variant is to embed the package of your

product package inside the QtInstall application itself. This is the easiest way for deploying applications since you have only one single file to provide. Nevertheless, if deploying more than one application, this method has the disadvantage that you have an overhead of about 5 MBytes per application for that QT library modules which are built in QtInstall.

2.1. Creating a package

QtInstall does following upon startup:

1. If QtInstall finds an embedded package it will always execute this package.
2. If no package is embedded in the installer, QtInstall will look after a package inside the current directory.
3. If no package is found, QtInstall will display a simple wizard which may be used to select a definition file and preview and create packages built with it.



With the „Definition file...“ button the package definition file is selected. After this, the „Preview package...“ creates a temporary package and simulates an installation process. When again clicking on the Preview button, the deinstallation is simulated.

„detailed logging“ will extend the logging during package creation and preview execution. This may be useful if there are any problems with created product packages.

„Create package...“ will create the complete installation package, either as .qip package or as executable standalone installer with an embedded package.

„Help“ will show this documentation.

2.2. Doing an installation

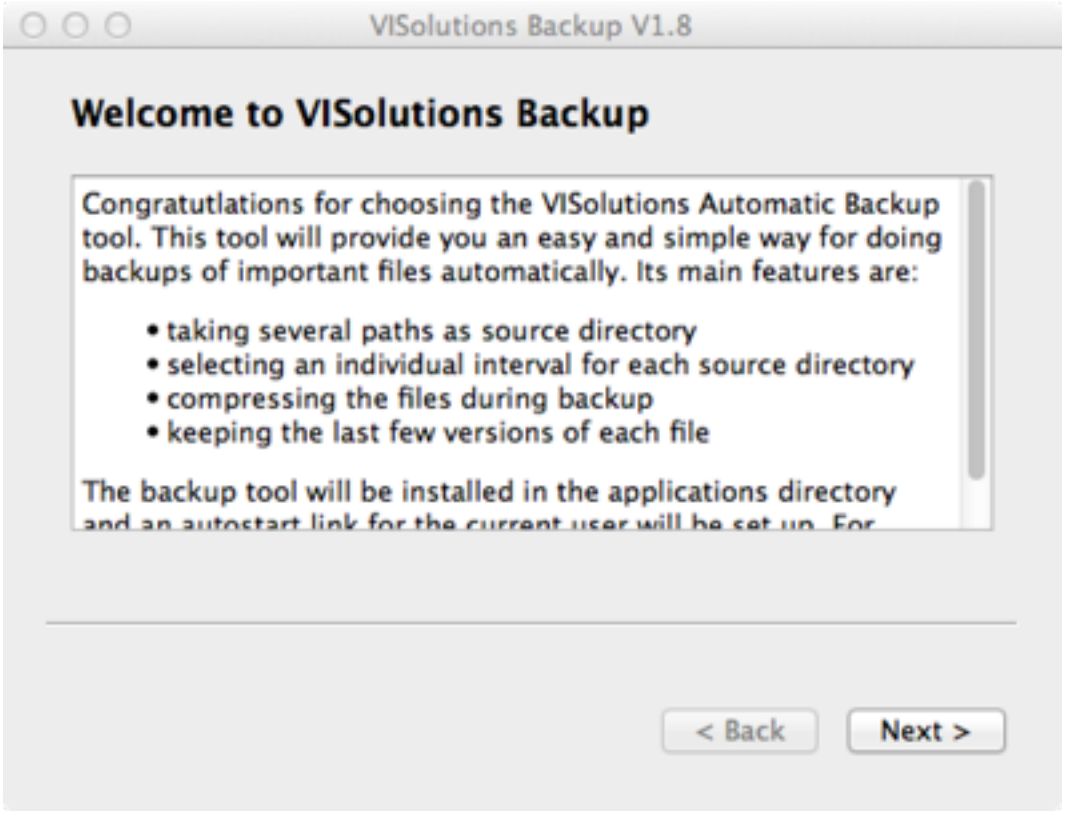
The four basic types of a product installation are

1. Installing the product on an untouched system
2. Deinstalling the complete product
3. Repairing an existing installation - mainly if there were access right problems during (de)installation
4. Updating an existing installation with a new version

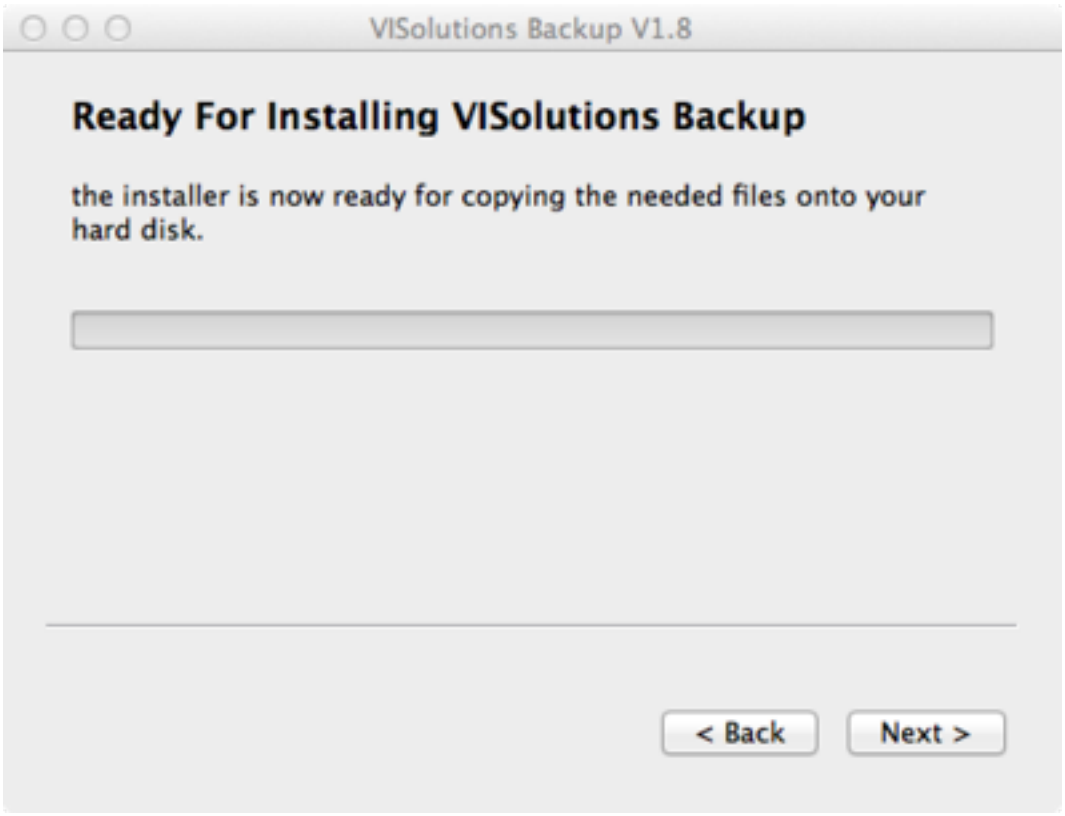
Installation of a product is started if QtInstall is executed with a package which does not exist on the target computer. An installation process consists of following steps:

- Welcome screen with text out of the definition
- Licensing information for the product (optional, when given in definition file)
- Install screen with progress bar
- Complete page with final message
- On first execution on a target computer, QtInstall shows its own GPL license at the very beginning.

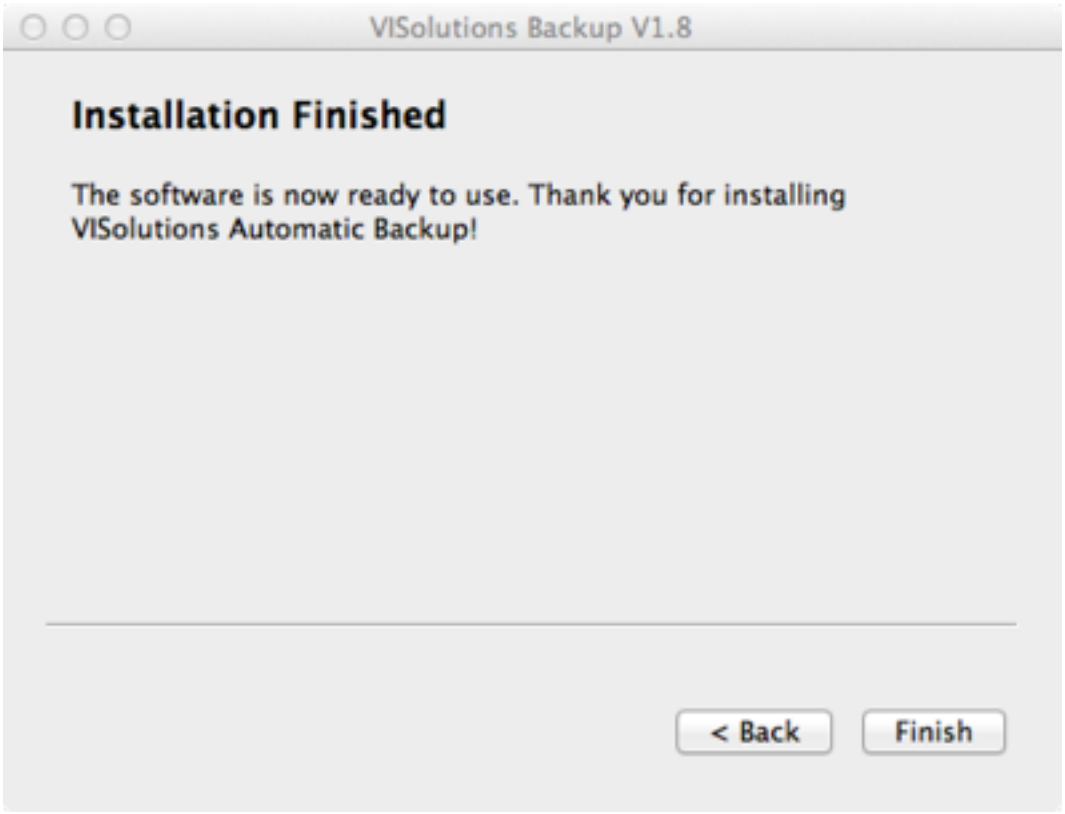
As an example, we show a complete turnaround of a product installation and deinstallation. We take the automatic backup tool, written by the same author. First, the welcome page appears on the screen:



After this, the install page shows up:



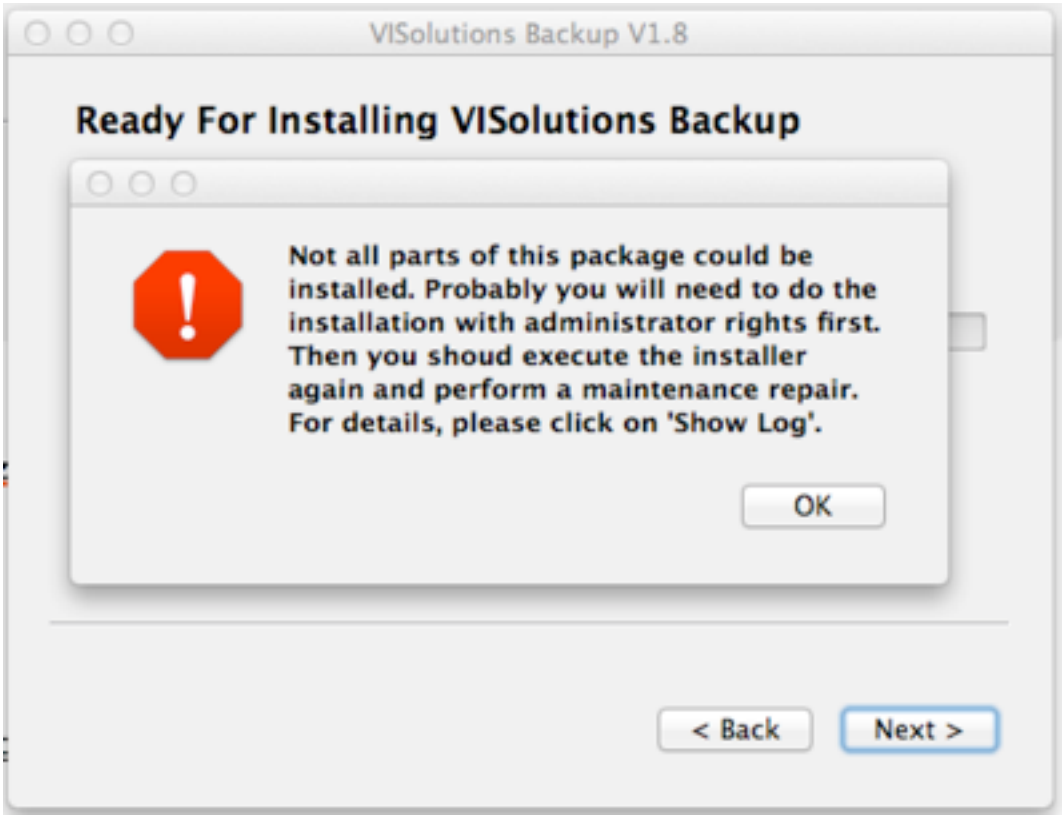
Finally, the complete page is shown:



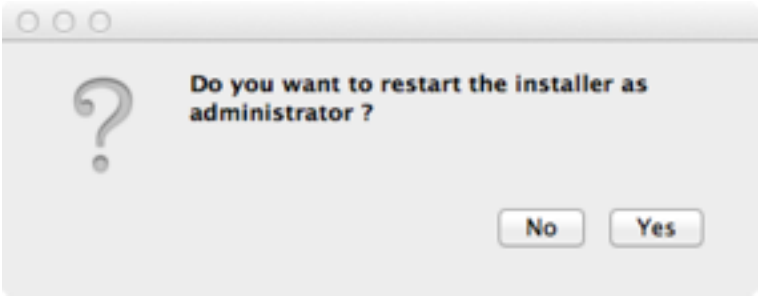
Well, that‘s it.

2.2.1. Accessing system files during installation

Often, you will need the access rights of an administrator for installing a product. QtInstall will recognize this during the installation process, and guide you through an elevation process of your access rights:



After finishing the installer, you will get following question:

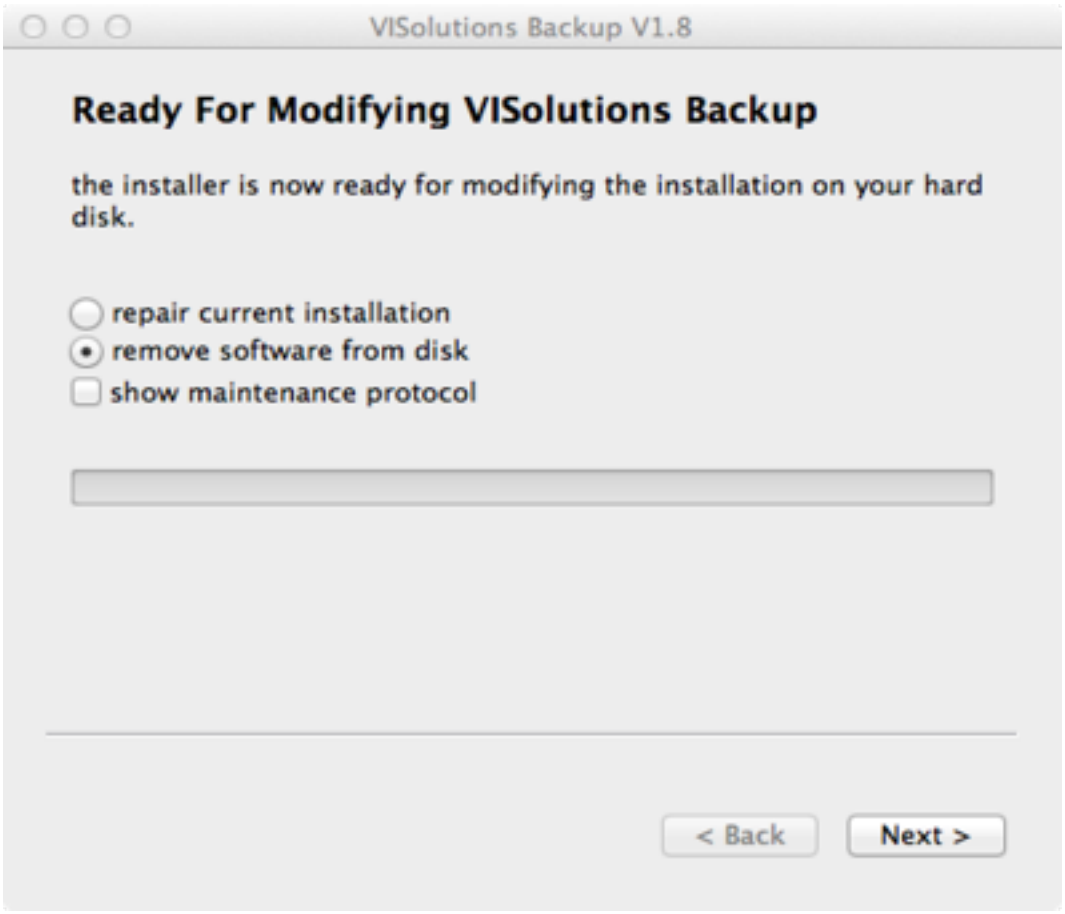


If you click on „Yes“, the system will be asked for privileged rights for accessing system directories. On Windows and Mac OS, system dialog will appear where you must give an admin password. The installer is now executed again with elevated rights.

After the repeated installation as administrator, you normally should do a „repair“ installation to get references and links correctly created.

2.3. Doing a deinstallation

Deinstallation of an existing product is started when QtInstall is executed with the same product version numbers which have been used at installation time. Deinstallation starts with the Modify page, where you select „remove product from disk“:



After deinstalling, again the complete page is shown.

Important: QtInstall will never remove files which are addressed inside the packages but have been existing during installation time. This is done for security reasons, because QtInstall has to assume that someone else needs these files, too. This feature allows installing products on the development machine itself where e.g. the QT library resides in its original version - and of course may never be deleted.

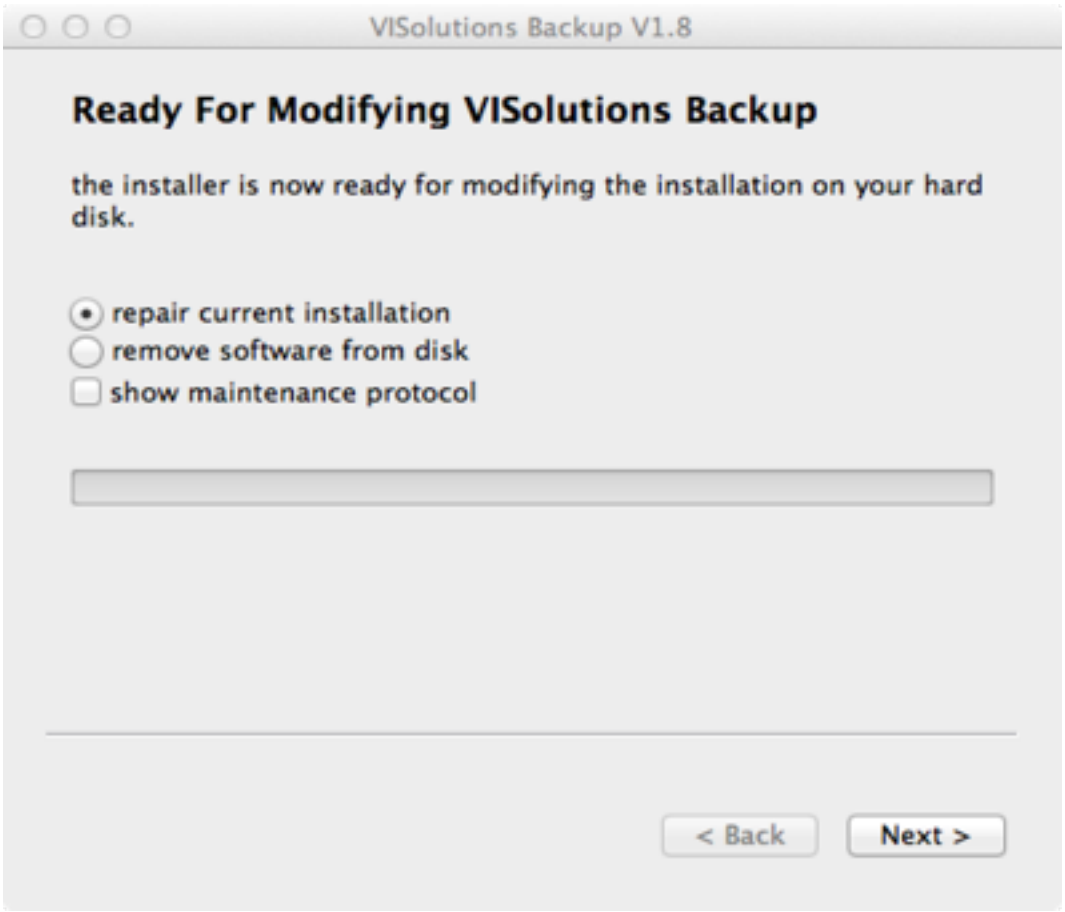
2.4. Modifying an existing installation

There are three kinds of installation modification:

1. The most often modification will be the product update of an existing installation. This is done automatically if QtInstall recognizes that one of the given version numbers has been changed. Please refer to the next section for this topic.

2. The deinstallation of a product is done by the modify process as described in the last section.

3. When during installation a repeated execution has been needed with elevated access rights, or when for some reasons a file of the product got lost, the repair modification will be needed. This dialog is automatically shown if the installer is executed again and no version change in the product is detected:



3.1. Major and minor updates

QtInstall supports both methods, either a major update of a product or a minor update. Although there are no differences in meaning between the common usage of these terms, here a short description of what they mean:

- A minor update is normally done if just files or settings in the product have been changed and their destination as to be updated. Another possibility is to have new files or settings inside the update package which have not been installed yet. QtInstall selects this kind of update if only the minor version string in the package has been changed.
- A major update has normally to be done if the file structure of the product has been changed or if files or settings have been removed from the old to the new version. A good example would be an update where the underlying version of the QT library has been changed, so that the file names of the shared libraries have changed (please refer to the next chapter for this topic). In this case, the old product is automatically removed with an uninstall process before the current product is installed again. QtInstall selects this kind of update if the major and / or minor version string in the package has been changed.

In order to be able to track the version numbers for each product QtInstall creates a QSettings entry of its own with the „Setup ID“ of the product package. Inside this entry all needed informations are stored as string.

Important: QtInstall will never remove files which are addressed inside the packages but have been existing during installation time. This is done for security reasons, because QtInstall has to assume that someone else needs these files, too. This feature allows installing products on the development machine itself where e.g. the QT library resides in its original version - and of course may never be deleted.

4. Preparing Your Applications For QtInstall

If you intend to build one or more applications with QT, you should be aware of the so called „DLLs hell“ problem. This means that different versions of your products need different versions of shared libraries which they use, but all of them have the same file name and location on the target platform. QtInstall allows you to avoid this if you‘re aware of this problem and respect following things:

- Under Windows, you will soon run into the „DLLs hell“ problem if yo want to support several applications with separate installers. At some time, these applications will use different versions of the underlying QT library. To avoid this problem, QtInstall gives you the ability to register separate search paths for each application. Although this feature is documented by Microsoft since Windows XP, it is not well known to the „normal“ QT developer. Therefore, you should set up the QT libraries with the configure option „-prefix ...“ . This will ensure that the QT libraries will be searched always in the given prefix path for all of your compiled applications. With a well defined naming convention of the prefix path, e.g. „c:\Qt\452“, you may tell QtInstall to always copy the correct versions of the libraries. On the target system you may set up an own directory for each version of QT, e.g. „c:\QtLib\452“. Don‘t forget to use the QtInstall (Windows) special settings key for adding the correct search path for your application.
- Under the Mac OS situation is a little bit better since your binaries always search for shared libraries at the same location where they were at compile time. But if you are not careful enough, it will be just the same like on Windows. So you should at least set up QT with the configure option „-prefix ...“ and „-no-framework“ option. This will ensure that the QT libraries will be searched always in the given prefix path for all of your compiled applications. With a well defined naming convention of the prefix path, e.g. „/Library/Qt/452“, you may tell QtInstall to always copy the correct versions of the libraries. On the target system, you must use the same path as destination directory.

Of course, the same problem may occur with your own shared libraries (if you for example are distributing more than one product), if you don‘t separate versions either in file name or in destination path, too. Generally spoken, it is a good style to create a sub directory for all own applications, e.g. „VISolutions“. Under this directory, you should place the QT libraries e.g. under „Qt/452“. On the same level, create a further sub directory for each of your applications.

4.1. A Simple Example

As a simple example we will take a look at the VISolutions backup tool. This tool is a QT application which shall get an installation process for Windows and Mac OS X. Let us assume that the tool sources reside in a network path which may be accessed from Windows and Mac. The QT project file is situated in the root directory. There are several subdirectories which contain additional resources:

```
backup/
     mac-release/
         backup.app/
             ...
     automatic-backup.pro
     ...
     win-release/
         backup.exe
     resources/
         backup.ico
         backup.pdf
```

```
        distribute/
            backup-installation.csv
            welcome.html
            automatic-backup.qip
```

4.1.1. The QtInstall Definition File

Now we want to start with the definition file *„backup-installation.csv“* for QtInstall. As of overview reasons, in the documentation the cells are marked with a number which will be explained below in the text.

Major	Setup ID		Minor
(a)	(b)		(c)

a) Here we write the major version of the product, in this example „1“

b) This is a unique identifier around all thinkable QtInstall products on a target platform; we use the well known UUID concept for providing this ID: „AFFDEA67-782D-45B4-AC7F-39EE2E2F2683“

c) This *is the minor version of the product. Let us assume „5“*

WindowTitle	Welcome Text (File)	Completion Text (File)	Lic Text File	Component(s)	
1)	2)	3)	4)	5)	

1) Here we write „VISolutions Automatic Backup V1.0“

2) The welcome text does not fit into a cell, so we choose the possibility to read it out of a file *„f:welcome.html“*.

3) The completion text is quite short, so we put it into the cell with a line break:
„The software is now ready to use. Thank you for
installing VISolutions Automatic Backup!“

4) The tool doesn't have an extra license text, so we omit this cell.

5) The Components cell is left blank.

Now we continue with the package contents. First, we put a comment line inside the definition:

Item Type	Source Path	Destination Path	Atrributes	Component(s)	Target
#					

4.1.1.1. Packaging the application

The Target column may be used to put different source files into the package depending on the resulting QtInstall package. We use this here in a common definition file for both Win and Mac. Now, we start with the application:

d	../backup.app	<APPDIR>/VISolutions/Automatic Backup/backup.app	copySrc		mac
f	../release/backup.exe	<APPDIR>/VISolutions/Automatic Backup/backup.exe			win32
f	backup.ico	<APPDIR>/backup.ico			win32

As you can see, Mac applications are in real a directory which encapsulates all files needed for the Finder to execute it. The *„copySrc“* attribute means that the file attributes of the underlying UNIX kernel should be taken from the sources (especially the „execute“ flag).

On the opposite, the Win version is straightforward. Here, we additionally copy an icon file onto the target machine which we may use later in the installation process.

4.1.1.2. Packaging the libraries

The Automatic backup is a QT application which needs the QtCore and QtGui library. Let us assume some system paths where the libraries are installed on the development machine. Of course, each developer has to insert his own settings here:

f	/Library/Qt452/lib/libQtCore.4.5.2.dylib	/Library/Qt452dynamic/lib/libQtCore.4.5.2.dylib		mac
f	/Library/Qt452/lib/libQtGui.4.5.2.dylib	/Library/Qt452dynamic/lib/libQtGui.4.5.2.dylib		mac
f	c:/MinGW/bin/mingwm10.dll	<APPDIR>/VISolutions/Qt452/mingwm10.dll		win32
f	c:/Qt/4.5.2/bin/QtCore4.dll	<APPDIR>/VISolutions/Qt452/QtCore4.dll		win32
f	c:/Qt/4.5.2/bin/QtGui4.dll	<APPDIR>/VISolutions/Qt452/QtGui4.dll		win32

Please note that the Windows version also needs the *mingwm10.dl*l which has to do with exception handling in multithreading C++ projects compiled with MinGW for Windows. You will find many issues about this obscure dependency in several newsgroups.

The Windows part of the package will not install the needed DLLs in the system path. Here we use the more advanced version of defining an application specific search path for libraries. QtInstall has built in support for this feature by providing a special settings syntax:

s	<WINCURRENTVERS>\App Paths\backup.exe\Default	<APPDIR>/VISolutions/Automatic Backup/backup.exe		win32
s	<WINCURRENTVERS>\App Paths\backup.exe\Path	<APPDIR>/VISolutions/Qt452/		win32

4.1.1.3. Packaging the default settings

Now we continue with additional steps for the installation process. The Automatic Backup tool uses some QSetting entries for its configuration. One setting has to be set by the installer:

soa	VISolutions.de	Automatic Backup			
s	ConfigFile	<HOMEDIR>/backup.cfg			

With this method, the tool will always find its configuration even if it is installed on different user accounts. Please keep attention to the „soa“ setting. It means something like „set organization and application name“ which may be defined on the QtCoreApplication class. They should be unique around the system and get the appropriate entries here.

Last but not least we want to have startup links on the desktop and in the auto start:

lcs		<APPDIR>/VISolutions/Automatic Backup/backup.app		mac
lcs	<APPDIR>/backup.ico	<APPDIR>/VISolutions/Automatic Backup/backup.exe		win32
lcd		<APPDIR>/VISolutions/Automatic Backup/backup.app		mac
lcd	<APPDIR>/backup.ico	<APPDIR>/VISolutions/Automatic Backup/backup.exe		win32

4.1.1.4. The complete example

Here is the complete definition file with comments:

```
1;AFFDEA67-782D-45B4-AC7F-39EE2E2F2683;5
Window Title;Welcome Text (File);Completion Text (File);Lic Text File;Component(s);
VISolutions Backup V1.0;f:welcome.html;"The software is now ready to use. Thank you
for
```

```
installing VISolutions Automatic Backup!";;;
Item Type;Source Path;Destination Path;Atrributes;Component(s);Target
#;;;;;
#;########## installing application itself ##########;;;;
#;;;;;
d;../backup.app;<APPDIR>/VISolutions/Automatic Backup/backup.app;copySrc;;mac
f;../release/backup.exe;<APPDIR>/VISolutions/Automatic Backup/backup.exe;;;win32
f;backup.ico;<APPDIR>/backup.ico;;;win32
soa;VISolutions.de;Automatic Backup;;;
s;ConfigFile;<HOMEDIR>/backup.cfg;;;
s;<WINCURRENTVERS>\App Paths\backup.exe\Default;<APPDIR>/VISolutions/Automatic Backup/
backup.exe;;;win32
s;<WINCURRENTVERS>\App Paths\backup.exe\Path;<APPDIR>/VISolutions/Qt452/;;;win32
#;;;;;
#;########## installing QT libraries ##########;;;;
#;;;;;
f;/Library/Qt452/lib/libQtCore.4.5.2.dylib;/Library/Qt452/lib/libQtCore.
4.5.2.dylib;;;mac
f;/Library/Qt452/lib/libQtGui.4.5.2.dylib;/Library/Qt452/lib/libQtGui.
4.5.2.dylib;;;mac
f;c:/MinGW/bin/mingwm10.dll;<APPDIR>/VISolutions/Qt452/mingwm10.dll;copySrc;;win32
f;c:/Qt/4.5.2/bin/QtCore4.dll;<APPDIR>/VISolutions/Qt452/QtCore4.dll;copySrc;;win32
f;c:/Qt/4.5.2/bin/QtGui4.dll;<APPDIR>/VISolutions/Qt452/QtGui4.dll;copySrc;;win32
#;;;;;
#;########## creating startup link ##########;;;;
lcs;;<APPDIR>/VISolutions/Automatic Backup/backup.app;;;mac
lcs;<APPDIR>/backup.ico;<APPDIR>/VISolutions/Automatic Backup/backup.exe;;;win32
lcd;;<APPDIR>/VISolutions/Automatic Backup/backup.app;;;mac
lcd;<APPDIR>/backup.ico;<APPDIR>/VISolutions/Automatic Backup/backup.exe;;;win32
```

And here is the welcome.html file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <meta http-equiv="Content-Style-Type" content="text/css">
  <title></title>
</head>
<body>
<p>Congratulations for choosing the VISolutions Automatic Backup tool. This
tool will provide you an easy and simple way for doing backups of important
files automatically. Its main features are:</p>
<ul>
  <li>taking several paths as source directory</li>
  <li>selecting an individual interval for each source directory</li>
  <li>compressing the files during backup</li>
  <li>keeping the last few versions of each file</li>
</ul>
<p>The backup tool will be installed in the applications directory and an
auto start link for the current user will be set up. For further details
please refer to the documentation.</p>
</body>
</html>
```

4.1.2. The installer in action

left blank intentionally

5. The QtInstall Definition File

All QtInstall package definitions are given as so called „csv“ files. This means the definition is given as table with several rows an columns. Each row of the table is written as ASCII into a file with the new line character at the end. Each column is separated by the others by the „;“ character. The csv format is very common on all systems and can be edited very comfortable by special programs like „Microsoft Excel“ or „Mac OS Numbers“. Each input line except the first 3 lines may begin with the numeric hash mark, „#“. All lines beginning with this character will be treated as comments and will be ignored.

A	B	C	D	E	F
#	this is a comment line				

5.1. The Package Header

The first 3 lines of the definition file are the package header. The first line gives a short description of the package specification cells. The second line defines the package itself. The third line of the package header gives a description of the Item definitions. The following lines define the several package items.

A	B	C	D	E	F
Major Version	Setup ID		Minor Version	Sub Version	
...	...		...	...	
Window Title	Welcome Text (File)	Completion Text (File)	Lic Text File	Component(s)	
...	...	...	...	...	...
Item Type	Source Path	Destination Path	Attributes	Component(s)	Target

5.1.1. Specifying The Package

Major Version — This cell contains the Major Version Number of the product. If this number is increased during an update, the old version of the product will first be removed completely.

Setup ID — This cell contains the UUID of this product package. This UUID may not change during lifetime of the product

Minor Version — This cell contains the Minor Version Number of the product. If this number is increased during an update, changed files of the product are overwritten by the installer automatically

Sub Version — This cell contains the Sub Version Number

Window Title — This cell contains the text which will be placed inside the wizard title bar.

Welcome Text (File) — This cell contains the text which will be the welcome message of the package. The text definition may contain the simple RTF tags which are understood by the QT HTML formatter. Here also a file can be defined if the content is more than a few words:

welcome to my simple installer or

f:welcome.txt

Completion Text (File) This cell contains the text which will be the completion message after successful installation. The text definition may contain the simple RTF tags which are understood by the QT HTML formatter. Here also a file can be defined if the content is more than a few words:

all is done! or

f:complete.html

Lic Text File If your application needs accepting a license, here the file is given which contains the complete licensee text. The text definition may contain the simple RTF tags which are understood by the QT HTML formatter.

Components This cell contains the definition of the Components of your package, if you want to define them. For Details please refer to the „Using Setup Types and Components“ section.

5.1.2. Specifying The Package Items

Item Type This cell defines the kind of item which will be added to the package.

f or ***fr***	Specifying a *File* Item
d	Specifying a *Directory* Item
p	Specifying a *Qt project* Item
s or ***soa***	Specifying a *QSettings* Item
lcd or ***lcs***	Specifying a *Symbolic Link* Item
lrd or ***lrs***	Specifying a *Symbolic Link* Item

Source Path This cell defines the source path on the package creating machine.

Destination Path This cell defines the destination path on the target device. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine. For details, please refer to the Appendix, „Predefined Symbolic Paths“.

Attributes This cell defines additional attributes for the given item. For details, please refer to the item description.

Component(s) This cell defines the components to which the item belongs.

Target This cell allows a restriction to a special target type. As of version 1.0, the following target types are recognized:

win32 and ***mac***

5.2. Specifying a *File* Item

Item Type

f or ***fc***	will install or update the given file on the target machine
fr	will remove the given <Destination Path> file on the target machine.

The *fr* option may be used to remove files which are generated by the application on the target system, e.g. help files. So you can ensure that generated files always will be updated by an updated application.

Source Path This cell defines the relative or full specified source file on the package creating machine. Relative paths are given in respect to the path where the csv file resides.

Destination Path This cell defines the full qualified file path on the target device. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine. You may also define an own directory structure in the *Destination Path*. Upon installation process, QtInstall will automatically check the directory tree on the target device and will create missing directories.

Attributes This cell defines additional permissions for the given item.

empty cell	the destination file will be generated with the system default attributes.
CopySrc	the destination file will get the same permissions as the source file. Especially on non-WIndows systems, this may be needed.
exec	the destination file will get the „execute“ permission set. Especially on non-WIndows systems, this may be needed.

5.3. Specifying a *Directory* Item

A directory item will not be stored with an own type. Instead of this, the given directory is scanned recursively and for each file found, a File Item will be added to the package.

Item Type

d	will install or update the given directory on the target machine

Source Path This cell defines the relative or full specified directory path on the package creating machine. Relative paths are given in respect to the path where the csv file resides.

Destination Path This cell defines the full qualified directory path on the target device. This will be the root path for all directories and files contained inside the given directory. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine.

Attributes This cell is reserved for future use.

5.4. Specifying a *QSettings* Item

Item Type

s	will install or update the given QSettings key on the target machine
soa	will set up the organization and application name on the target device

Source Path This cell defines the QSettings key to be used. As mentioned in the QT documentation, this may be a „hierarchical keys using the '/' character“.

Destination Path This cell defines the string value of the QSettings key. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine.

Attributes This cell is reserved for future use.

5.5. Specifying a *Symbolic Link* Item

Item Type	***lcs*** or ***lcd***	will create a startup or desktop link to the given *Destination Path* on the target machine.
	lrs or ***lrd***	will remove the startup or desktop link to the given *Destination Path* on the target machine.
Source Path	**Windows only**: This cell defines the icon file on the target device which will be used for the symbolic link.	
Destination Path	This cell defines the full qualified file path for the link destination file (typically an application) on the target device. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine. Please refer to the appendix.	
	Mac only: To create a desktop or startup link to an application, you must specify the „.app“ directory as destination, e.g. „<APPDIR>/backup.app“	
Attributes	This cell is reserved for future use.	

5.6. Win Only: Specifying a *Search Path for Dlls* Item

Item Type	***s***	will create a native Windows registry entry as soon as the symbolic name <*WINCURRENTVERS*> is found in the *Source Path*.
Source Path	This cell defines the native registry key. If the end of the key name is „Default“, the Default key of the given registry key is set.	
Destination Path	This cell defines the contents of the given key. QtInstall understands several symbolic names which will be replaced by the appropriate paths on the destination machine.	
Attributes	This cell is reserved for future use.	

To set up a search path for an installed Qt application called „backup.exe“, you typically must define following keys:

s	<WINCURRENTVERS>\App Paths\backup.exe\Default	<APPDIR>/backup.exe
s	<WINCURRENTVERS>\App Paths\backup.exe\Path	c:/test/452

As you can easily see from the definition, the Qt library DLLs will be searched inside the directory „c:/test/452“.

5.7. Specifying a *Qt Project* Item

Item Type	***p***	will install or update all files of the given Qt project on the target machine
Source Path	This cell defines the relative or full specified file name of the Qt qmake project file.	
Destination Path	This cell defines the full qualified base directory on the target device. This will be the root path for all directories and files specified in the project. QtInstall	

	understands several symbolic names which will be replaced by the appropriate paths on the destination machine.
Attributes	This cell is reserved for future use.

6. Using Setup Types and Components

reserved for future use.

7. Appendix

7.1. Predefined Symbolic Paths

<APPDIR>	will be replaced by the directory where normally the applications are situated. On Mac OS, this will be „/Applications“, on Windows the path given by the environment variable %ProgramFiles%, typically „C:\Program Files“. Please be aware that you will need admin rights in order to modify this directory.
<SYSDIR>	will be replaced by the system directory. On Mac OS, this will be „/Library“, on Windows the Windows root directory given by the environment variable %SystemRoot%, typically „C:\Windows“. Please be aware that you will need admin rights in order to modify this directory. QtInstall will hardly try to find out if it has the appropriate rights upon installation, but this process may fail especially under Windows 7 where the „Virtual Store“ mechanism obscures what really is going on. The mechanism of QtInstall is to recognize if one or more files should be copied into the <APPDIR> or <SYSDIR> path. In this case, if errors occur during file copy, it tries to create a dynamic library inside the <SYSDIR> path. If this fails, the complete page of the install wizard will recommend to repeat the installation with admin rights.
<HOMEDIR>	will be replaced by the users home directory. On Mac OS, this will be the %HOME % directory, typically „/Users/<username>“. On Windows, this will be the %USERPROFILE% directory, typically „C:\Documents and Settings\<username>“
<APPDATADIR>	will be replaced by the directory reserved for extra data of applications. On Mac OS, this will be „/Library/Application Support“, on Windows XP the %APPDATA% directory.

7.2. QtInstall Package file format

The QtInstall Package format is defined as binary format with a header, followed by several datagrams. Each datagram has a standard header with its type and size information so that a unknown datagram always may be skipped. The file ending is defined as „.cab“.

```
+----------------------+-------------+-------------+-----+-------------+
| package header | data | <datagram 1> | <datagram 2> | ... | <datagram n> |
+----------------------+-------------+-------------+-----+-------------+
```

The *package header* contains a magic word to be sure that the given file really is a QtInstall package, followed by the version number of the file format, the number of datagrams inside the package. At the end of the package header comes the detailed description of the package which contains the complete informations out of the definition file.

All integer values are defined in the Intel notation.

7.2.1. Package Header

```
struct cabinetMagic
{
    unsigned int magic;
    unsigned int version;
    unsigned int nelements;
    unsigned int descrLength;
};
```

magic is the magic word which declares the file as QtInstall package. The value is ‚VISL‘

version is the file format version: <main>*1000000 + <sub>*1000 + <minor>

nelements number of datagrams inside the package

descrLength length of the following data block with the complete description

Now following the Package Header Data:

data is a String object which contains the summary of all package properties. The properties are separated by the keyword „@propval@“. The position inside the data string has following meaning:

property index	property content
0	window title
1	welcome text (may be HTML)
2	completion text (may be HTML)
3	license text (may be HTML)
4	components definition
5	Setup ID
6	Setup major version (String)
7	Setup minor version (String)

7.2.2. Datagram Headers

```
struct cabinetHeader
{
    cabinetDatagramID ID;
    unsigned int attributes;
    unsigned int dataLength;
};
```

ID is the numeric ID of the following datagram. The IDs are defined as following:

datagram ID	datagram type
0	file datagram
1	settings datagram
2	links datagram

attributes is a combination of several bit flags which differ from datagram type to datagram type.

Bit mask definition for File Datagrams:

```
#define useFilePermissions      0x00000001
#define executablePermission    0x00000002
#define removeDestination       0x00000004
```

Bit mask definition for QSettings Datagrams:

```
#define settingsAppAndOrgName   0x00000001
#define settingsRemoveSettings  0x00000002
```

dataLength is the length of the following datagram data (including sub header)

7.2.2.1. File Datagram Sub Header / Data

```
struct fileDataHeader
{
    unsigned int destinationLength;
    unsigned int dataLength;
    unsigned int propertiesLength;
    unsigned int lastModified;
    unsigned int filePermissions;
};
```

destinationLength is the length of the destination path definition which follows in the data block

dataLength is the length of the file contents binary data

propertiesLength is the length of the property list string data. For a description of these please refer to „Datagram Properties Description“.

lastModified is the time stamp of the modification date of the source file in seconds since 1st of January, 1970, 00:00

filePermissions is the bit mask of the filePermissions which the destination should get. The bit mask is defined by QT.

Now following the File Datagram Data:

dstFileName is the data block for the destination path string

dstProperties is the data block for the property list string. For a description of these please refer to „Datagram Properties Description“.

dstBinData is the data block for the binary file data

8. Settings Datagram Sub Header / Data

```
struct settingsDataHeader
{
    unsigned int keyLength;
    unsigned int valueLength;
    unsigned int propertiesLength;
};
```

keyLength is the length of the QSettings key string

valueLength is the length of the QSettings value string

propertiesLength is the length of the property list string data. For a description of these please refer to „Datagram Properties Description“.

Now following the Settings Datagram Data:

keyName is the data block for the QSettings key string

keyProperties is the data block for the property list string. For a description of these please refer to „Datagram Properties Description“.

keyValue is the data block for the QSettings value string

9. Links Datagram Sub Header / Data

```
struct linksDataHeader
{
    unsigned int targetLength;
    unsigned int iconFileLength;
    unsigned int propertiesLength;
    unsigned int operation;
};
```

targetLength is the length of the target file name

iconFileLength is the length of the icon file name

propertiesLength is the length of the property list string data. For a description of these please refer to „Datagram Properties Description“.

operation is the desired link operation on the target device. The operations are:

op code	operation
0	create an auto start link for the current user
1	remove the auto start link for the current user
2	create a desktop link for the current user
3	remove the desktop link for the current user

Now following the Links Datagram Data:

targetFile is the data block for the target file name string

linkProperties is the data block for the property list string. For a description of these please refer to „Datagram Properties Description“.

targetIconfile is the data block for the icon file name string

9.1.1. Datagram Properties Description

To be defined in the future