QtInstall Documentation

Version 1.0

August 2009

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

Table of Contents

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.

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.

QtInstall does following upon startup:

1. looking for a default package definition file named „definition.csv“. If this file is found, QtInstall will ask you whether to use it for creating a new package.
2. if no definition file found or the question is answered with „no“, QtInstall will search for any installation package „*.cab“. if only one is found, the installer will take and execute it. If more than one packages are found, the installer will ask for which to use.
3. If neither the default definition nor any package file is found, QtInstall will ask you for a package definition file „*.csv“ and use this file for creating a new package.

This kind of operation allows you to easily provide the two most frequent types of install packages. 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 package) on a media like CD or USB stick.

Preparing Your Applications For QtInstall

If you intend to build one or more applications with QT, you should be aware of following things:

- Under Windows, you will probably 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, you should 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.

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.

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				

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
Window Title	Welcome Text (File)	Completion Text (File)	Lic Text File	Component(s)	
...	...	...	...	...	...
Item Type	Source Path	Destination Path	Attributes	Component(s)	Target

Specifying The Package

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.

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***

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.

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.	

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.	

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. **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.	

Windows 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“.

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.	

Using Setup Types and Components

reserved for future use.

Appendix

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.

<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.

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.

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

Datagram Header

```
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
```

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

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

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

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

Datagram Properties Description

To be defined in the future