Qt-Projekte mit Visual Studio 2005

Benötigte Programme:

Visual Studio 2005	Vollversion, Microsoft
Qt 4	Open Source s. Qt 4-Installationsanleitung

Tabelle 1: Benötigte Programme für die Qt-Programmierung in Visual Studio

1. Neues Projekt anlegen

   Ein neues Projekt wird über das Menü *Datei -> Neu -> Projekt* erstellt. Im Dialog wird, wie unten abgebildet, ein Visual C++-Projekt vom Typ Win32 erstellt. Dort muss noch einmal “Win32-Projekt” ausgewählt werden, bevor die restlichen Einstellungen getätigt werden können.

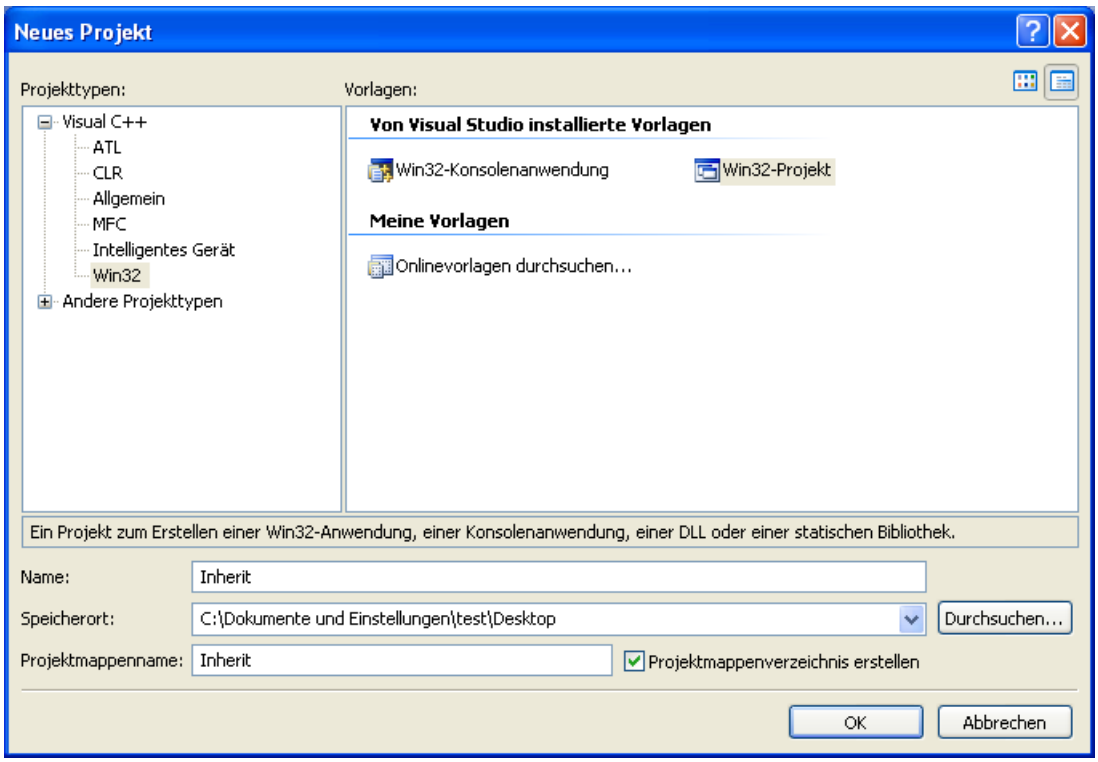

Abbildung 1: Dialog zum Erstellen eines neuen Projekts in Visual Studio

Die Anwendungseintellungen werden, wie unten abgebildet, auf „Windows-Anwendung“ und „Leeres Projekt“ eingestellt.

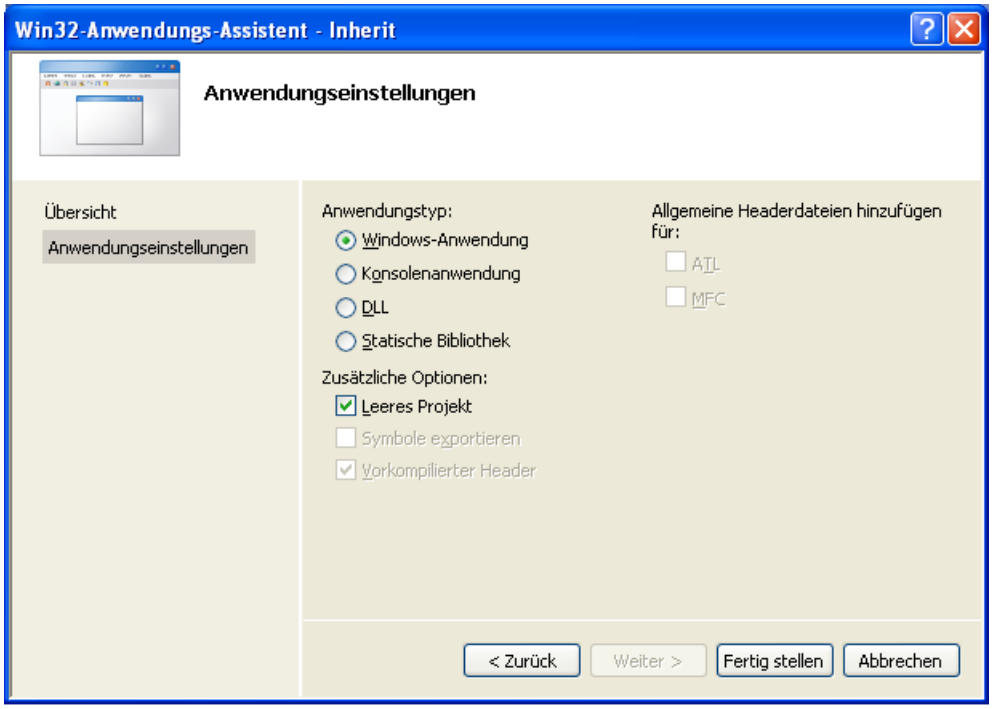

Abbildung 2: Einstellungen für die Erstellung eines neuen Projekts

Wenn jetzt der *Fertigstellen*-Knopf gedrückt wird, dann wird das neue Projekt erstellt.

2. Hinzufügen der zusätzlichen Include-Verzeichnisse

   In den Projekt-Eigenschaften müssen noch einige Änderungen vorgenommen werden, bevor Qt-Programme erstellt werden können. Der entsprechende Dialog kann über das Menü *Projekt -> Eigenschaften* aufgerufen werden.

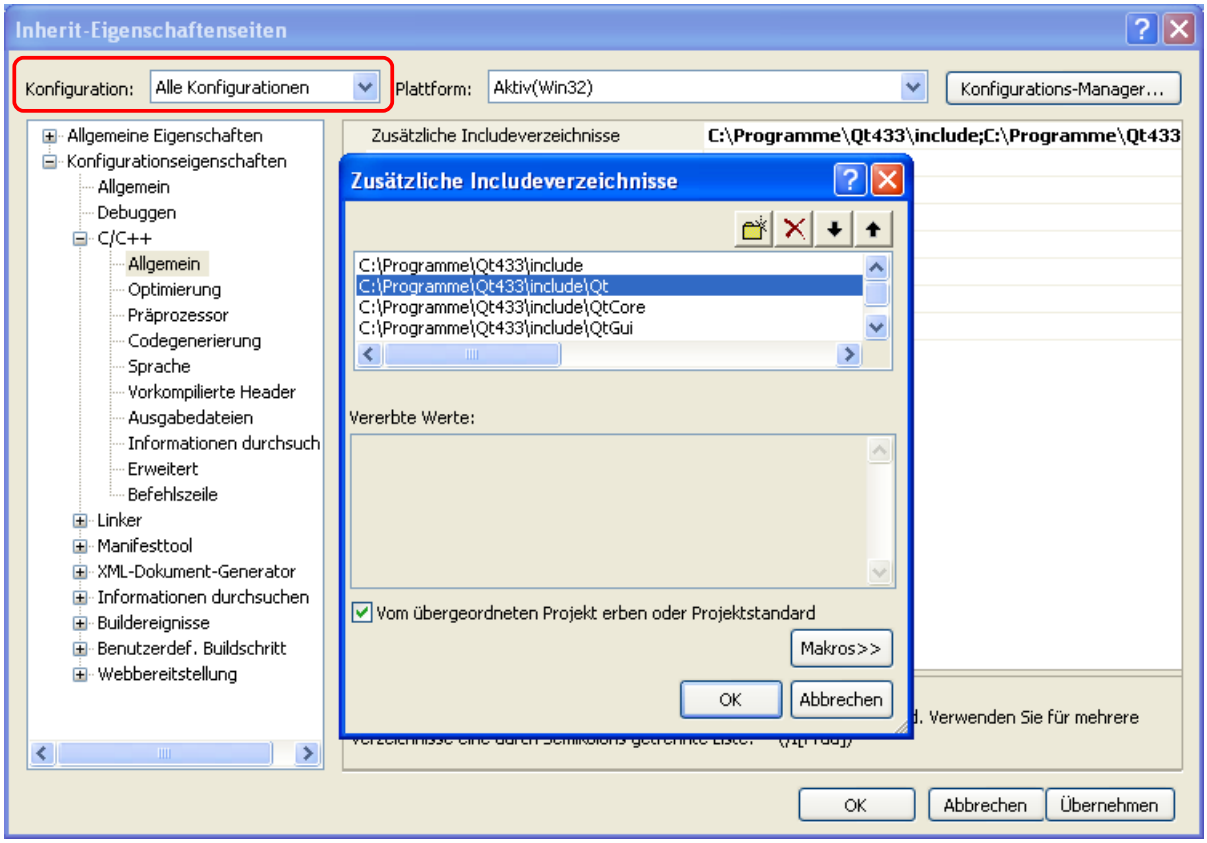

Abbildung 3: Dialog zum Hinzufügen neuer Include-Verzeichnisse

Wichtig ist, dass die Konfiguration, wie in der Abbildung markiert, auf „Alle Konfigurationen“ eingestellt ist, da sonst die Konfiguration nur für die momentante Einstellung, meist „Debug“, geändert wird und die Einstellungen zwei mal geändert werden müssen.

Zusätzliche Include-Verzeichnisse können in der Kategorie *Konfigurationseigenschaften -> C/C++-> Allgemein* hinzugefügt werden. Dort müssen die zusätzlichen Vereichnisse durch einen Strichpunkt getrennt in die Zeile „Zusätzliche Include-Verzeichnisse“ eingetragen werden. Alternativ kann auch der in der Abbildung gezeigte Dialog verwendet werden.

Verzeichnis	**Bedeutung**
$(QTDIR)\include	Hauptverzeichnis, immer nötig
$(QTDIR)\include\Qt	Allgemeine Klassen, immer nötig
$(QTDIR)\include\QtCore	
$(QTDIR)\include\QtGui	Grafische Oberfläche
$(QTDIR)\include\QtNetwork	Netzwerk-Unterstütung
$(QTDIR)\include\QtUiTools	Qt-Designer-Unterstützung
$(QTDIR)\include\QtOpenGl	OpenGL-Klassen
$(QTDIR)\include\QtSVG	SGV-Grafik-Unterstützung
$(QTDIR)\include\QtXml	XML-Dateiverwaltung
$(QTDIR)\include\Qt3Support	Qt 3-Kompatibilitätsklassen
$(QTDIR)\include\QtTest	Zusätzliche Debug-Funktionen

Tabelle 2: Qt-Include-Verzeichnisse

Die benötigten Include-Verzeichnisse können der Tabelle 2 entnommen werden, wobei die ersten drei Einträge immer benötigt werden, um eine Qt-Anwendung erstellen zu können. Da Qt-Programme meist eine grafische Oberfläche haben, ist auch das QtGui-Verzeichnis fast immer notwendig.

3. Zusätzliche Bibliotheken einbinden

Damit das Programm auf die Qt-Funktionen zugreifen kann, müssen die Qt-Bibliotheken beim Linken des Programms eingebunden werden.

Die zusätzlichen Bibliotheken werden ebenfalls in den Projekt-Einstellungen, aber diesmal im Unterpunkt *Konfigurationseigenschaften -> Linker -> Eingabe* eingetragen.

Die Linker-Einstellungen müssen für die Release- und für die Debug-Konfiguration getrennt vorgenommen werden, weil für das Debugging eigene Debug-Bibliotheken mitgeliefert werden. Bei den Debug-Bibliotheken ist dem Namen noch ein „d“ angehängt, z. B. QtCored4.lib anstatt QtCore4.lib.

Wie in Abbildung 4 gezeigt, werden die Bibliotheksdateien in die Zeile „Zusätzliche Abhängigkeiten“ eingetragen.

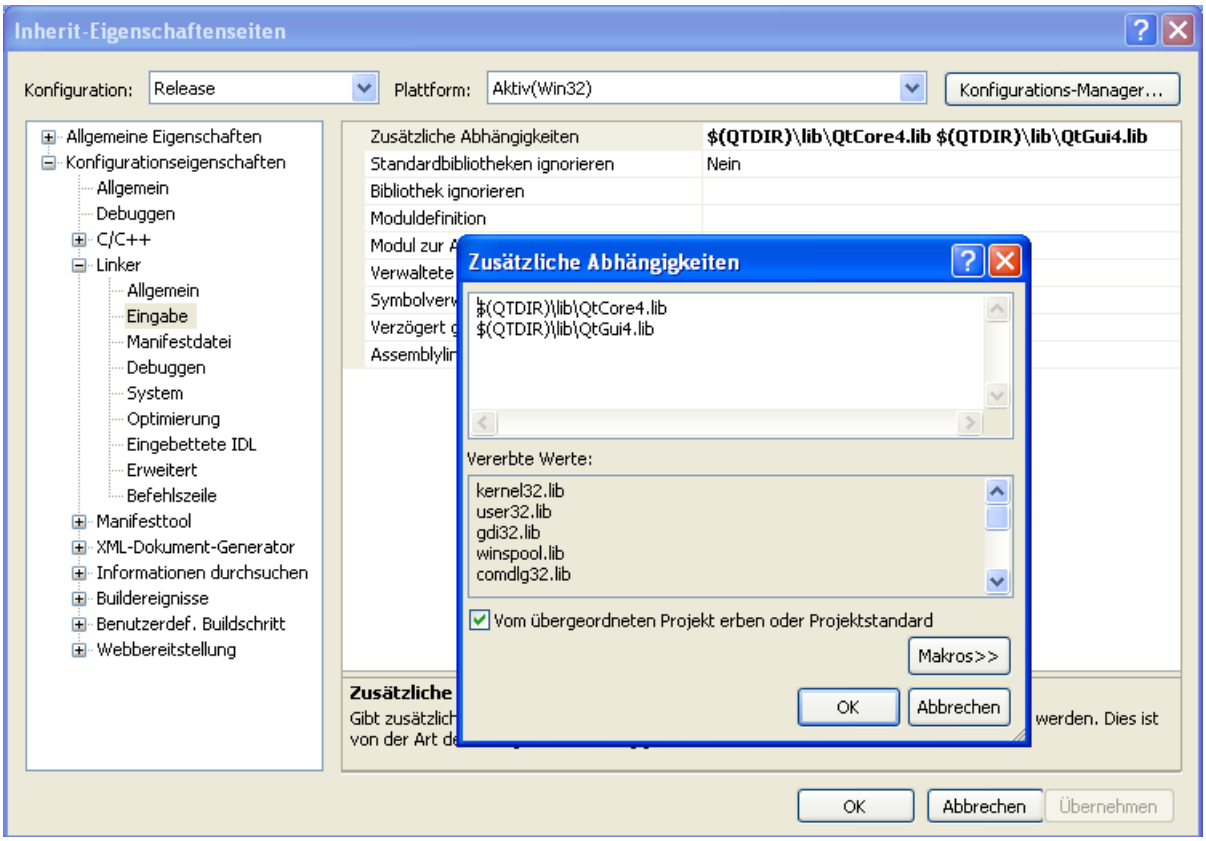

Abbildung 4: Linker-Einstellungen für die Release-Konfiguration

Welche Bibliotheken für das das jeweilige Programm benötigt werden, kann aus der folgenden Tabelle 3 entnommen werden.

$(QTDIR)\lib\QtCore4.lib	Allgemeine Klassen
$(QTDIR)\lib\QtGui4.lib	Grafische Oberfläche
$(QTDIR)\lib\QtNetwork4.lib	Bibliothek für Netzwerk-Programmierung
$(QTDIR)\lib\QtOpenGL4.lib	OpenGL-Unterstützung
$(QTDIR)\lib\QtSql4.lib	SQL-Datenbank-Bibliothek
$(QTDIR)\lib\QtScript4.lib	Qt-Script-Bibliothek
$(QTDIR)\lib\QtSvg4.lib	SVG-Grafik-Unterstützung
$(QTDIR)\lib\QtXml4.lib	XML-Bibliothek
$(QTDIR)\lib\QtUiTools4.lib	Qt-Designer-Unterstützung
$(QTDIR)\lib\Qt3Support4.lib	Qt 3-Kompatibilitätsbibliothek
$(QTDIR)\lib\QtTest4.lib	Bibliothek mit Debugging-Funktionen

Tabelle 3: Die Qt-Bibliotheken

4. Hinzufügen von Dateien

   Bei allen Dateien, die Klassen mit Signalen oder Slots enthalten, müssen zusätzlich Datei-spezifische Einstellungen vorgenommen werden. Dies ist bei der normalen Strukturierung von Qt-Programmen nur bei den Header-Dateien nötig. Wird also eine neue Header-Datei angelgegt, z. B. im Projektmappen-Explorer über *Rechte Maustaste -> Hinzufügen -> Neues Element* und im Dialog *Visual C++ -> Code*, so muss bei der erstellten Datei mit *Rechte Maustaste -> Eigenschaften* der Einstellungsdialog geöffnet werden.

   Es erscheint der in Abbildung 5 gezeigte Dialog.

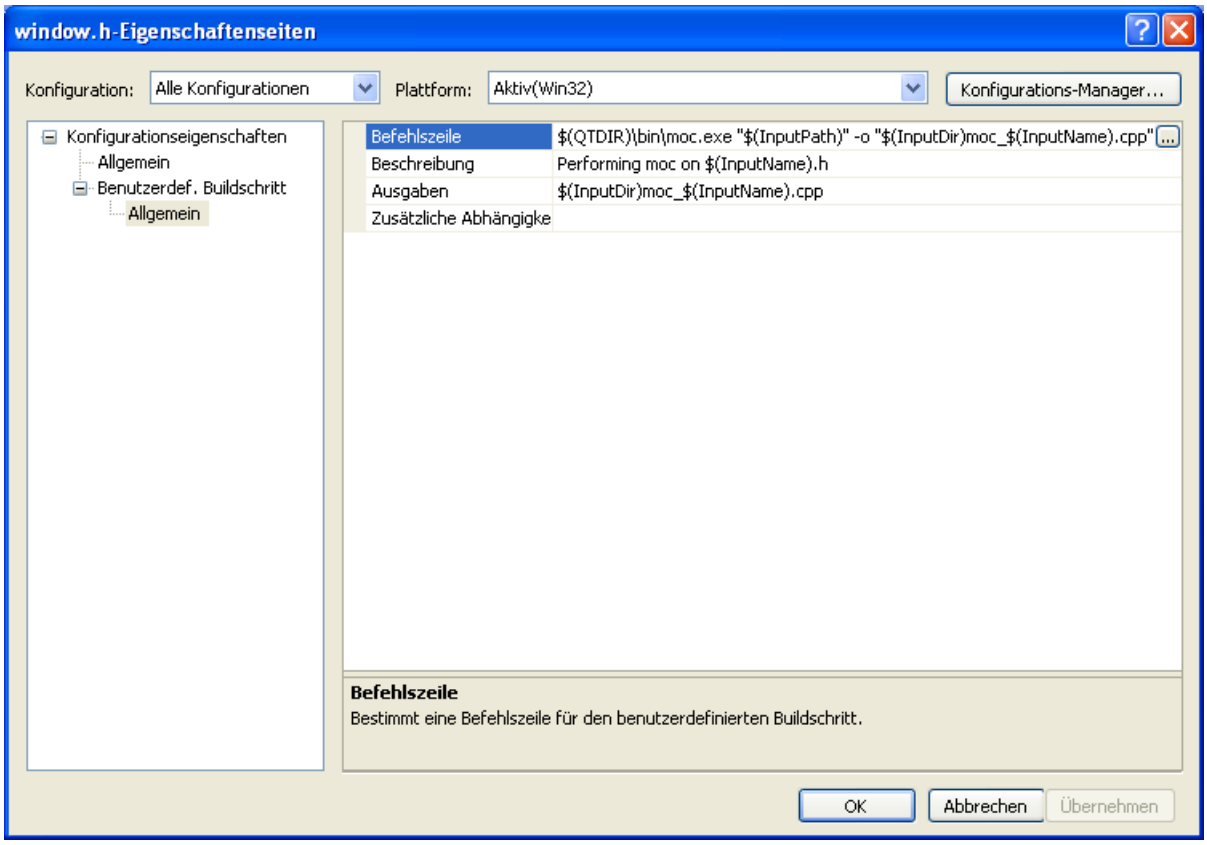

Abbildung 5: Hinzufügen eines Benutzerdefinierten Build-Schritts für den moc-Compiler

Hier wird zuerst die Konfiguration auf „Alle Konfigurationen“ umgestellt. Bei der Auswahl von *Konfigurationseigenschaften -> Benutzerdef. Buildschritt -> Allgemein* werden mehrere Zeilen angezeigt, in die der Aufruf für den Meta Object Compiler eingetragen wird. Die Inhalte sind für jede Datei gleich und können Tabelle 4 entnommen werden.

Befehlszeile:	$(QTDIR)\bin\moc.exe "$(InputPath)" -o "$(InputDir)moc_$(InputName).cpp"
Beschreibung:	Performing moc on $(InputName).h
Ausgaben:	$(InputDir)moc_$(InputName).cpp

Tabelle 4: Einstellungen für den moc-Compiler

Nachdem die Klasse erstellt wurde und der Compiler einmal aufgerufen wurde, wird der Meta Object Compiler aufgerufen und erstellt eine Datei mit dem moc_Dateiname.cpp. Diese Datei muss noch dem Projekt hinzugefügt werden.

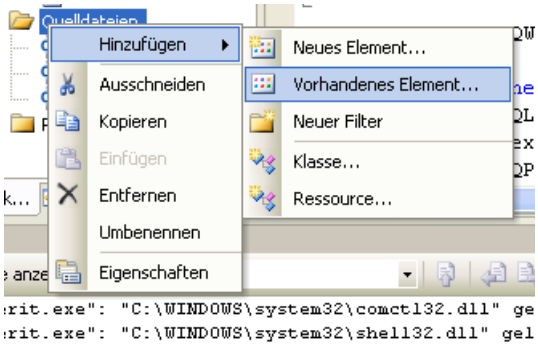

Abbildung 6: Hinzufügen einer moc-Datei

Dazu kann man z. B. im Projektmappen-Explorer über *Rechte Maustaste -> Hinzufügen -> Vorhandenes Element* den entsprechenden Dialog öffnen und die Datei auswählen.

Beim nächsten Kompilieren wird die Datei mit eingebunden und das Programm lässt sich vollständig kompilieren.