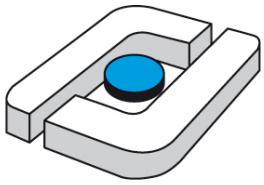


Fachhochschule Osnabrück
University of Applied Sciences

Fakultät Ingenieurswissenschaften und

Informatik

Bachelorarbeit

über das Thema

Freie Software Werkzeugkette zur Software-Entwicklung

für Windows CE

vorgelegt durch

Andre Heinecke

FACHHOCHSCHULE OSNABRÜCK

Fakultät für Ingenieurswissenschaften und Informatik

Bachelorarbeit

Thema: Freie Software Werkzeugkette zur Software-Entwicklung für Windows CE

für

Herrn: Andre Heinecke geboren am: 28.06.1986 in: Siegburg

Matrikel Nr: 350649

Erstprüfer/in: Prof. Dr.-Ing. Jürgen Wübbelmann

Zweitprüfer/in: Dipl.Systemwiss., MSc. Bernhard Reiter

Beginn: 01.03.2010

Abgabe: 21.05.2010

_____ _____

Erstprüfer/in Bachelor

_____

eingereicht am: Verlängerung genehmigt bis:

Inhaltsverzeichnis

Zusammenfassung

Als Kern der Microsoft Betriebssysteme für mobile Geräte und Smartphones kommt Windows CE weltweit millionenfach in embedded Systemen zum Einsatz. Zur professionellen Software-Entwicklung werden dabei überwiegend die Microsoft Visual Studio Werkzeuge verwendet, da diese als einziger Weg gelten, stabile und leistungsfähige Softwareprodukte für Windows CE zu erstellen. Die vorliegende Arbeit evaluiert Möglichkeiten der Software-Entwicklung für Windows CE, stellt eine freie Werkzeugkette vor und untersucht, ob es möglich ist, allein auf Basis Freier Software, Entwicklung für die CE Plattform zu betreiben.

Abstract

Windows CE, as the core of Microsoft’s embedded and smartphone operating- systems, is used in millions of devices all over the world. The professional software development for Windows CE is currently mainly focused on the Visual Studio toolset, which is deemed to be the only way to produce stable customer friendly software solutions for this particular platform. This paper focuses on Free Software to develop professionally for Windows CE and discusses ways and means to achieve a development toolchain, which is solely based on Free Software.

1 Einleitung

Im vergangenen Jahrzehnt konnte man beobachten, wie durch neue Anwendungen und Dienste der Alltag in den Industrienationen zunehmend vom Internet durchdrungen wurde. Mit der rasanten Verbreitung sozialer Online-Netzwerke und der Entwicklung leistungsfähigerer und sparsamerer Systeme entstand ein Massenmarkt für mobile Kommunikationsgeräte, die sich als Kombination eines Mobiltelefons mit einem Personal Data Assistant (PDA) von Spielzeugen technikverliebter Nutzergruppen zu einem Standardwerkzeug des modernen Medienkonsumenten entwickelten. Auch immer mehr Unternehmer verwenden heutzutage diese neuen Kommunikationsmittel, um durch die Einbindung ihrer Mitarbeiter und Führungskräfte in das Unternehmensnetz deren Arbeitsabläufe effizienter zu gestalten. Diese Vorraussetzungen schufen einen stark wachsenden Markt für Softwaredienstleistungen mit speziell auf mobile Plattformen zugeschnittenen Lösungen.

Besitzen klassische Mobiltelefone meist nur eine vordefinierte Benutzeroberfläche, welche in einigen Fällen vorsieht, innerhalb einer Laufzeitumgebung z.B. durch Java-Anwendungen erweitert zu werden, verfügen Smartphones in der Regel über ein deutlich umfangreicheres Betriebssystem, welches es dem Benutzer ermöglicht, selbst Programme nach Belieben zu installieren und über die vorinstallierte Konfiguration hinaus zu verwenden. Die Programme selbst sind dabei nicht durch eine Laufzeitumgebung eingeschränkt, sondern können direkt nativen Code auf der Plattform ausführen und so sämtliche Ressourcen eines Gerätes effektiv nutzen.

Beschleunigt durch die rasante Entwicklung der Hardware kann neue Software zum Einsatz gebracht werden, die noch vor einigen Jahren aufgrund ihrer Anforderungen auf “Handheld”-Geräten undenkbar gewesen wäre. Fehlende Standards und der Konkurrenzkampf unterschiedlicher Hersteller, die ihre Marktanteile durch proprietäre Software und Patente zu sichern suchen, führen jedoch dazu, dass die Portierung bestehender Software auf mobile

Geräte ein teueres und aufwendiges Unterfangen ist.
Windows CE, mit dem sich diese Arbeit beschäftigt, bildet die Grundlage der von Microsoft, dem weltweit größten Entwickler von Betriebssystemen, vertriebenen Betriebssysteme für Smartphones und andere mobile Geräte. Während andere Unternehmen ihre mobilen Betriebssysteme auf etablierten Kernen wie Linux oder BSD aufbauten, entwickelte Microsoft zu diesem Zweck in den neunziger Jahren ein eigenes System. Zu Beginn des letzten Jahrzehnts besaß Windows CE noch eine dominierende Marktstellung bei PDAs, verlor jedoch mit dem Durchbruch der Smartphones zunehmend an Bedeutung. Mit Windows Mobile wurde zwar im Jahr 2003 ein Betriebssystem speziell für diese Geräte zusammengestellt, aber die Entwicklung des Kerns stagniert seit 2004, als das letzte größere Update vorgenommen wurde. So wurden von Microsoft wichtige Markttrends, wie etwa berührungsempfindliche Displays, vernachlässigt. Dennoch besitzt Windows Mobile, gerade auch im für Software-Dienstleister wichtigen Geschäftskundenbereich immer noch einen großen Marktanteil, der mit bis zu 40% angegeben wird. [Can]

Für überwiegend an Unterhaltung und Freizeitnutzung interessiere Konsumenten ist insbesondere die mangelnde Verfügbarkeit von Applikationen ein Argument gegen Windows Mobile. Hohe Einstiegshürden bei der Software-Entwicklung für Windows CE, wie etwa die Abhängigkeit von einer kommerziellen Lizenz für Microsoft Visual Studio, tragen dazu bei, dass die Entwicklung kleiner, aber innovativer Software nur in eingeschränktem Maße stattfindet. Aufgrund der Kosten für die Software-Entwicklung werden speziell Hobbyisten, die keinerlei Gewinnerwartungen mit Software verbinden, davon abgehalten neue Anwendungen oder Portierungen von Freier Software für Windows CE zu entwickeln.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist es, die Chancen und Möglichkeiten der Arbeit mit freien Werkzeugen in der Software-Entwicklung für Windows CE aufzuzeigen und auf ihre Praxistauglichkeit hin zu prüfen. Auftretende Schwierigkeiten sollen dabei dokumentiert und mögliche Lösungswege aufgezeigt werden. Besonderheiten der Software-Entwicklung für Windows CE werden dabei im Allgemeinen beschrieben und es wird aufgezeigt, welche Hilfsmittel man als Freie Software zur Verfügung hat, um existierende (Freie) Software auf diese Platt-

form zu portieren.

1.2 Gliederung der Arbeit

Die vorliegende Arbeit gliedert sich in fünf Kapitel. Das erste Kapitel bietet eine Einführung in die Freie Software sowie Informationen über die technischen Grundlagen von Windows CE und geht auf Schwierigkeiten bei der Software-Entwicklung speziell für diese Plattform ein. Daran schließt sich im zweiten Kapitel die Beschreibung der proprietären Werkzeuge des Unternehmens Microsoft an, woraufhin im dritten Kapitel Freie Software-Alternativen für diese proprietären Werkzeuge vorgestellt und erläutert werden. Das vierte Kapitel beschreibt die Ansätze, vorhandene Freie Software auf Windows CE zu portieren. Dabei werden die freien Werkzeuge mit der Erstellung des Qt Frameworks zur Anwendung gebracht und auf ihre Praxistauglichkeit hin überprüft. Im letzten Kaptitel wird ein Fazit dieser Arbeit gezogen sowie ein Ausblick auf die Entwicklung der Windows CE Plattform und die Zukunft Freier Software auf dieser gegeben.

2 Grundlagen

In diesem Kapitel soll eine Einführung in die Freie Software gegeben werden sowie eine Erläuterung zur genauen Definition des Begriffs. Anschließend wird auf das Betriebssystem Windows CE eingegangen, um ein grundlegendes Verständnis für die Software-Entwicklung für diese Plattform zu schaffen.

2.1 Werkzeuge zur Software-Entwicklung

Im Rahmen dieser Arbeit sollen Werkzeuge untersucht werden, welche es ermöglichen Software-Entwicklung für Betriebssysteme auf Basis des Windows CE Kernels zu betreiben. Der Schwerpunkt wird nicht auf Werkzeuge zum Entwickeln des Quelltextes oder Hilfsmittel für das Software-Engineering gelegt, sondern auf die Werkzeuge, welche es einem Entwickler[1] erlauben Quellcode in ein natives, also nicht von einer Laufzeitumgebung abhängiges Programm für Windows CE zu überführen, zu testen und auszuliefern. Dabei soll untersucht werden, ob es möglich ist, den in 2.1 dargestellten iterativen Entwicklungsprozess für Windows CE allein auf Basis Freier Software umzusetzen.

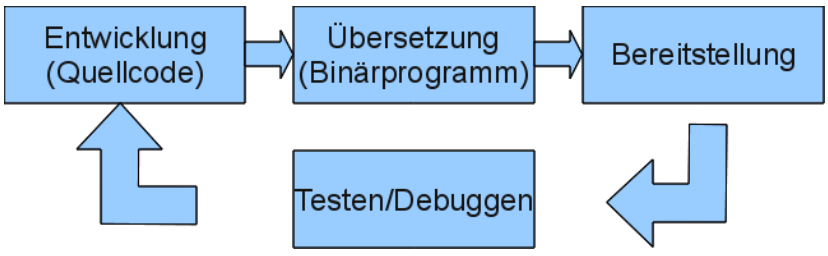


Abbildung 2.1: Iterativer Entwicklungszyklus

[1] Um die Lesbarkeit zu vereinfachen wird auf die zusätzliche Formulierung der weiblichen Form verzichtet. Es wird darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form explizit als geschlechtsneutral verstanden werden soll.

Die hier visualisierte Art der Vorgehensweise in der Software-Entwicklung ist das übliche Verfahren, nach dem Freie Software entwickelt wird. Als Testen wird dabei auch die Anwendung einer Software verstanden. Aufgrund dieses Verständnisses der Software-Entwicklung als evolutionärem Prozess, welcher zu immer neuen Änderungen und Verbesserungen führt, wird in dieser Arbeit auch der Begriff des Freien Software “Projektes” vermieden. Wenngleich viele Freie Software Entwickler ihre Arbeit selbst als Projekt verstehen, ist die Bezeichnung Projekt irreführend, denn ein Projekt ist dadurch definiert, dass es ein klar definiertes Ende besitzt. Dies ist aber bei nahezu allen so genannten Freie Software Projekten nicht der Fall, da sich diese stetig weiterentwickeln und nie zu einem klaren Ende kommen. Aus diesem Grund ist es korrekter von Software Initiativen zu sprechen.

2.2 Freie Software

> *Denn jede Sache, die durch Weitergabe an andere nicht verliert, besitzt man nicht, wie man sollte, solange sie nur besessen und nicht an andere weitergegeben wird.*
>
> Frei nach Augustinus von Hippo, Römischer Philosoph und Kirchenlehrer,
>
> De doctrina christiana 397 n. Chr.

Bei Freier Software handelt es sich nicht um ein Produkt im klassischen Sinne, sondern in erster Linie um Wissen über die Verarbeitung von Informationen, also eine Ware, welche durch die Weitergabe an andere im Allgemeinen nicht verliert. So ist es in der Wissenschaft üblich, seine Erkenntnisse zu veröffentlichen und andere Wissenschaftler daran teilhaben zu lassen, damit diese davon lernen und durch Kritik konstruktiv an der Weiterentwicklung teilnehmen können. Im Gegensatz dazu ist es in der Informatik verbreitet, Wissen als Eigentum zu behandeln, dieses geheim zu halten, zu schützen und als Ware zu verkaufen.

Freie Software ist der Gegenansatz zu dieser proprietären Art der Software-Entwicklung. Das bedeutet jedoch nicht, dass Freie Software antikapitalistisch wäre oder dass man mit Freier Software kein Geld verdienen kann. (Genauso wenig wie dies bei anderen Wissenschaften der Fall ist, in denen Forschungsergebnisse veröffentlicht werden.) Die Freiheit von Software bezieht sich demnach auf das Wissen, welches in einer Software steckt, nicht auf den Preis. Freie Software kann man zwar ohne Kosten verwenden, aber sie erlaubt es auch jedem, sie

zu kopieren, zu modifizieren und weiter zu vermarkten. [Fref]

Von der Free Software Foundation Europe [Frea] wird Freie Software durch vier Grundregeln definiert, auf denen eine Software basieren muss, um wirklich frei im Sinne von Freiheit [Fref] von Wissen zu sein.

1. Die Freiheit, das Programm für jeden Zweck auszuführen.
2. Die Freiheit, die Funktionsweise eines Programms zu untersuchen und es an seine eigenen Bedürfnisse anzupassen.
3. Die Freiheit, Kopien weiterzugeben und damit seinen Mitmenschen zu helfen.
4. Die Freiheit, ein Programm zu verbessern und die Verbesserungen an die Öffentlichkeit weiterzugeben, so dass die gesamte Gesellschaft davon profitieren kann.

Voraussetzung für die Erfüllung dieser Kriterien ist, dass der Quelltext der Software jedem frei zugänglich sein muss. Oft wird deshalb fälschlicherweise Open Source (offener Quelltext) als Synonym für Freie Software verwendet„ obwohl Quelltexte, auch wenn sie veröffentlicht werden, nicht in der Form lizensiert sein müssen, dass alle oben genannten Kriterien erfüllt sind. [Freb] Ist aber eine dieser Freiheiten nicht erfüllt, spricht man von proprietärer Software.

Verglichen mit Open Source bietet Freie Software nicht nur ein technisches Modell, mit dem bessere Software entwickelt werden kann, sondern darüber hinaus eine Philosophie. Die grundsätzliche Idee, die hinter der Philosophie Freier Software steckt, ist, das Wissen einer ganzen Gemeinschaft zur Weiterentwicklung von Software nutzen zu können. Dabei heißt dies auch hierbei nicht, dass die Software-Entwicklung nicht auch kommerziell sein kann. Ein freies Programm muss ebenso für den kommerziellen Gebrauch, die kommerzielle Entwicklung und für die kommerzielle Verteilung zur Verfügung stehen. Die kommerzielle Entwicklung Freier Software ist heutzutage nicht mehr unüblich; Laut einer Studie der Europäischen Union aus dem Jahr 2006 hatten Freie Software und Freie Software Dienstleistungen einen Marktanteil von 20,5 % an den Softwareinvestitionen in der EU. [RG06] Bei Freier Software hat jeder die Möglichkeit sie zu verkaufen, wenn er Kunden findet, die bereit sind für den Mehrwert, der ihnen geboten wird, zu bezahlen. Jedoch macht es keinen Unterschied, ob man für Software bezahlt oder sie kostenlos erhalten hat. Man besitzt immer die Freiheit, die Software zu kopieren, sie zu verändern, sogar Kopien davon zu ver-

kaufen. [Gra02]

2.2.1 Lizenzkategorien

Um diese Rechte zu schützen, gibt es eine Reihe von Lizenzen, welche die Nutzungsrechte von Software regeln. Der Verfasser einer Software kann nach dem deutschen Urheberrecht sein Werk unter eine Lizenz stellt, mit welcher er Nutzern Nutzungsrechte für seine Software erteilt. Auch wenn alle Lizenzen die o.g. vier Grundpfeiler der Freien Software einhalten, gibt es große Unterschiede zwischen Freien Software Lizenzen. Grafik 2.2 ist [Rei04] entnommen und schlägt eine Kategorisierung Freier Software Lizenzen in vier Gruppen vor, welche in den nächsten Abschnitten erläutert werden.

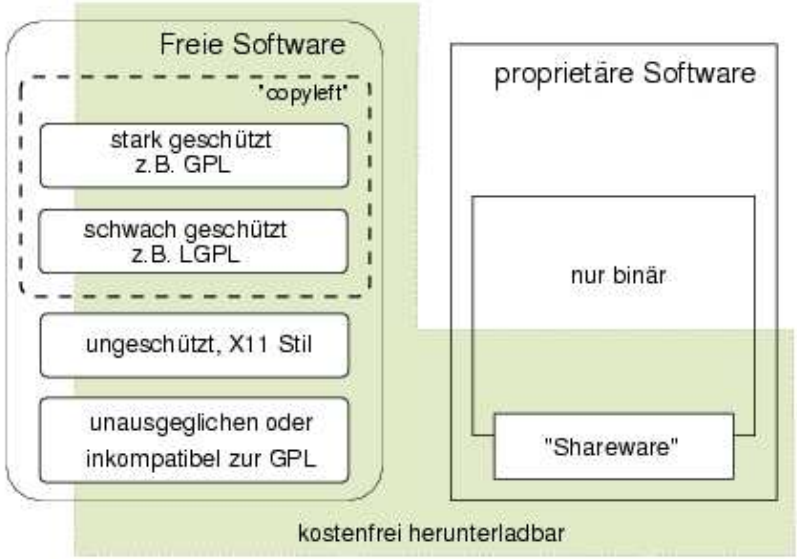


Abbildung 2.2: FS Lizenzkategorien (nach Reiter [Rei04])

2.2.1.1 Stark schützende Lizenzen

Damit es unkooperativen Nutzern nicht ermöglicht wird, Freie Software in proprietäre Software umzuwandeln, gibt es die stark schützenden Lizenzen. Eine stark schützende Lizenz versucht die Freiheit einer Software auf ewig zu sichern. Dabei werden die o.g. Grundpfeiler Freier Software sämtlich erfüllt, aber es wird sichergestellt, dass niemand das Werk verwenden kann, ohne diese Grundrechte auch anderen Nutzern zu gewähren. Die GNU[2] General

[2]GNU ist ein rekursives Akronym für GNU's not UNIX

Public License (GPL) [Free] ist eine derartige Lizenz. Richard Stallman hatte als Autor der GPL die Idee *»einen Bestand an nützlicher Software aufzubauen«* [Gra02], deren abgeleitete Werke ebenfalls einer stark schützenden Lizenz unterstellt werden müssen. Diese Lizenzen werden oft auch als Copyleft-Lizenzen bezeichnet. Copyleft stellt dabei ein Programm explizit unter das Copyright, um dann als Rechtsmittel Vertriebsbestimmungen hinzuzufügen, welche es allen erlauben, den Code des Programms oder jedes davon abgeleiteten Programms zu verwenden, zu ändern und weiter zu verteilen, aber nur wenn die Vertriebsbestimmungen unverändert bleiben. So werden mithilfe des Urheberrechts der Quellcode und die gewährten Freiheiten rechtlich untrennbar. Diese Idee, auf der das Copyleft basiert, wird von Stallman wie folgt erklärt: *»Ich denke mir, dass während die Entwickler proprietärer Software das Copyright verwenden, um uns am Austausch zu hindern, wir das Copyright nutzen können, um anderen, die wie wir zusammenarbeiten wollen, einen eigenen Vorteil zu sichern: die Verwendung unseres Codes«* [Sta].

2.2.1.2 Schwach schützende Lizenzen

Es gibt allerdings Bereiche in denen sich stark schützende Lizenzen weniger eignen. Insbesondere bei Funktionsbibliotheken kann eine derartige Restriktion dazu führen, dass diese Software nicht verwendet wird, denn jedes Programm, welches Funktionen aus einer unter GPL lizenzierten Bibliothek verwendet, muss wieder Freie Software sein. [Freg] Schwach schützende Lizenzen, wie die LGPL (Lesser General Public License), geben dieselben Freiheiten wie stark schützende Lizenzen. Der Unterschied besteht lediglich darin, dass Bibliotheken einer schwach schützenden Lizenz von Programmen benutzt werden können, ohne dass diese selbst einer entsprechenden Lizenz unterstellt werden müssen. Die GNU C-Bibliothek ist ein gutes Beispiel für den sinnvollen Einsatz dieser Lizenz, durch die erreicht wird, dass die Software frei bleibt, aber mit der auch proprietäre Software-Entwicklung möglich ist. Ebenso wie die stark schützenden werden auch schwach schützende Lizenzen Copyleft-Lizenzen genannt.

2.2.1.3 Nicht schützende Lizenzen

Nicht schützende Lizenzen gewähren die Freiheiten der Nutzung Freier Software, wie BSD oder X11, schützen diese aber nicht. Jedem wird dadurch die Möglichkeit gegeben, diese Software nicht nur in proprietären Programmen zu verwenden, sondern sie auch selbst als proprietäre Software weiterzuentwickeln. [X C] Im amerikanischen Recht gibt es zudem noch die Möglichkeit seine Software als Public Domain zu deklarieren, sie also ins Gemeineigentum zu übergeben und auf jedes Copyright zu verzichten. Diese Möglichkeit sieht das kontinentaleuropäische Copyright nicht vor. [Gra02]

2.2.1.4 General Public License inkompatible Lizenzen

Um verschiedene Software oder substantielle Teile dieser miteinander in einer größeren Arbeit zu kombinieren, benötigt man eine Kompatibilität dieser Lizenzen. Da es sich bei der GPL um die mit Abstand weitest verbreitete Lizenz für Freie Software handelt, unterscheidet man zwischen GPL kompatiblen und inkompatiblen Lizenzen. GPL Inkompatibilität kann verschiedene legale Gründe haben und ist zumeist unerwünscht, da, wie bereits erwähnt, ein Großteil Freier Software die GPL verwendet.

2.3 Windows CE

Neben Windows NT, auf welchem alle modernen Microsoft Windows Desktop-Betriebssysteme (XP, Vista, 7, u.a.) basieren, entwickelte das Unternehmen Microsoft noch den Windows CE Kernel. Dieser seit 1993 entwickelte Kernel bildet die Basis für verschiedene Betriebssysteme von Microsoft, welche für eingebettete Systeme (embedded systems) konzipiert sind, beispielsweise für Windows Mobile als Betriebssystem für Smartphones. Windows CE wurde entworfen als das kleinste von Microsofts Betriebssystemen, um aus einem möglichst kleinen Read Only Memory (ROM) Chip ausgeführt werden zu können und dabei eine Untergruppe von Funktionen der WinAPI (siehe Abschnitt 2.4) zu implementieren. Dadurch konnte Microsoft Windows und die Windows API in Märkten und auf Geräten etablieren, für die sowohl der Windows NT Kernel, als auch der Windows 9x Kernel zu groß waren. [Bol01]

2.3.1 Marketing Namen

Der Name Windows CE steht nur für den Betriebssystemkern; als Microsoft Windows Embedded CE wird ein Betriebssystem für diesen Kern vertrieben. Dieses bietet die Möglichkeit, auf Basis der Plattform Builder Software, nur bestimmte Teile des Betriebssystems in seinen Produkten zu verwenden oder sogar den unter proprietärer Lizenz stehenden Programmcode des Betriebssystems seinen eigenen Bedürfnissen anpassen zu können. Dadurch wird weiter zur Heterogenisierung der Plattform beigetragen und nahezu unmöglich gemacht, alle Funktionen, die ein Hersteller in seine Plattform spezifisch eingefügt hat, mit freien Werkzeugen zu unterstützen. Mit Windows Mobile bietet Microsoft eine Zusammenstellung des Windows Embedded CE Betriebssystems auf Basis des Windows CE Kernels als Komplettpaket für mobile Geräte für die ARMv4 Prozessorarchitektur an. Dabei wird dann noch unterschieden zwischen “Classic”, “Standard” und “Professional”. Classic ist eine Variante ohne die Unterstützung von Telefonfunktionen, Standard steht für Geräte mit Telefonfunktionen, aber ohne berührungsempfindliches Display und Professional heißt die Version, welche sowohl Telefonfunktionen unterstützt, als auch ein berührungsempfindliches Display besitzt. [Lan] Zusammenfassend:

- **Windows CE**: Der Betriebssystem Kern
- **Windows Embedded CE**: (Seit Version 6.0) der Name des konfigurierbaren Betriebssystems auf Basis von Windows CE
- **Windows Mobile, Windows Phone, Pocket PC, Windows Automotive,..**: Betriebssystem-Distributionen, die auf einer Version des Windows CE Kernels und verschiedenen Softwarekonfigurationen aufgebaut werden.

Außer es ist explizit ein spezifisches System gemeint, wird im Rahmen dieser Arbeit allgemein von Windows NT gesprochen, wenn Windows Betriebssysteme gemeint sind, welche auf dem NT-Kernel basieren und entsprechend von Windows CE, wenn Systeme mit dem CE-Kernel gemeint sind.

2.3.2 Technische Grundlagen

Im Folgenden werden die technischen Einschränkungen und Eigenheiten des Windows CE Betriebssystems beschrieben. Die Ausführungen und Daten beziehen sich auf Windows CE 5.02, welches in den aktuellen Systemen Windows Mobile 6.1 und Windows Mobile 6.5 Verwendung findet. Um zu verstehen, welche Schwierigkeiten in der Software-Entwicklung für diese Plattformen existieren, wird an dieser Stelle die Verwaltung von Systemressourcen von Windows CE eingeführt.

2.3.2.1 Speicherlayout

> *I have to say that in 1981, making those decisions, I felt like I was providing enough freedom for 10 years. That is, a move from 64k to 640k felt like something that would last a great deal of time. Well, it didn't - it took about only 6 years before people started to see that as a real problem.*
>
> Bill Gates, 1989 ([Gat])

Mit diesem Zitat und der zugrunde liegenden Einstellung, dass ein Betriebssystem nur Hardware für eine bestimmte Zeitspanne unterstützen sollte, sind vielleicht einige der Einschränkungen von Windows CE 5.02 zu erklären. Im Jahr 2004, als Windows CE 5.0 veröffentlicht wurde, schienen 32MiB Adressraum für Prozesse eines eingebetteten Systems noch ausreichend. Dass sechs Jahre später die mobilen Betriebssysteme von Microsoft immer noch mit diesen Einschränkungen arbeiten müssen, kann unter anderem auch erklären, warum es wenig Software und Interesse an der Software-Entwicklung für Windows CE gibt. Die Einschränkungen ergeben sich aus der Tatsache, dass es für den Kernel nur einen einzelnen Adressraum gibt, dessen eine Hälfte aus einem Systemspeicherbereich besteht, in welchem der Kernel ausgeführt wird, wohingegen sich die andere Hälfte aus 64 jeweils 32MiB großen Speicherbereichen zusammensetzt, von denen wiederum nur in 32 Bereiche Prozesse geladen werden können. [Bol01] Diese 32 Bereiche sind nochmals unterteilt in 64KiB große Segmente, in die jeweils nur ein Objekt geladen werden kann. Durch diese Aufteilung werden großen Teile des Adressraums für kleine Prozesse verschwendet, während große Prozesse, trotz ausreichender Hardwareressourcen, nicht mehr als 32MiB Adressraum zur

Verfügung haben. Dies erschwert, beziehungsweise verhindert die Portierung einer großen Bandbreite von Software, welche die Plattform als solche durchaus attraktiver machen könnte (Beispielsweise Mozilla Firefox). Die Tabelle 2.1 gibt wieder, wie sich der Speicher für den Kernel genau aufteilt:

Beginn	**Ende**	**Größe**	**Inhalt**	**Beschreibung**
8000 0000	FFFF FFFF	2 GiB	Kernel	Systemreserviert (Kernel Speicher)
7E00 0000	7FFF FFFF	32 MiB	Bereich 63	Dynamisch linkbare Ressourcen (Schriften, Bilder, kein ausführbarer Code)
7C00 0000	7DFF FFFF	32 MiB	Bereich 62	Geteilter, dynamischer “Heap Speicher”
7A00 0000	7BFF FFFF	32 MiB	Bereich 61	Dynamisch linkbare Bibliotheken (> 64 KB)
7800 0000	79FF FFFF	32 MiB	Bereich 60	Dynamisch linkbare Bibliotheken (> 64 KB)
7600 0000	77FF FFFF	32 MiB	Bereich 59	Gerätemanager Stapelspeicher für Threads
4200 0000	75FF FFFF	832 MiB	Bereiche 32-58	Speicher-abgebildete Dateien (geteilt)
4000 0000	41FF FFFF	32 MiB	Prozess 31	Ausführbarer Prozess
...	...	...	...	...
0400 0000	05FF FFFF	32 MiB	Prozess 1	Ausführbarer Prozess
0200 0000	03FF FFFF	32 MiB	Bereich 1	Dynamisch linkbare Bibliotheken des ROM sowie Module
0000 0000	01FF FFFF	32 MiB	Bereich 0	Aktiver Prozess

Tabelle 2.1: Windows CE 5.02 Speicheranordnung

Für einen einzelnen Prozess wiederum stellt sich der Speicher anders dar. Sobald ein Prozess aktiv wird, wird sein Speicher in den Bereich 0 abgebildet. Dadurch ist es möglich, dass Code und Daten positionsabhängig adressiert werden können. Der Prozess selbst bekommt damit einen eigenen virtuellen Adressraum, von welchem er aber nur einen kleinen Teil (seinen eigenen 32MiB Bereich) frei verwalten kann.

Alle dynamisch linkbaren Bibliotheken (DLL), die zur Zeit der Ausführung geladen sind, stehen für alle Prozesse zur Verfügung und werden in jeden virtuellen Adressraum eingebunden. Dies hat in den Versionen vor Windows CE 5.02 dazu geführt, dass durch das Laden einiger großer DLLs der Speicher für jeden Prozess immer weiter reduziert wurde, wodurch der so genannte “DLL Crunch” Effekt eintrat, bei welchem ein Prozess von Bibliotheken “erdrückt” wurde. Als eine der wichtigsten Neuerungen von Windows CE 5.02 (auf wel-

chem die 6.x Versionen von Windows Mobile basieren) gilt deshalb auch die Einführung der Bereiche 60 und 61 als Adressräume, in die Bibliotheken geladen werden, die größer als 64KiB (ein Segment) sind. [Bol]
Durch diese Bereiche ist es nun möglich, auch umfangreichere Funktionsbibliotheken wie etwa das Qt Framework zu verwenden.

Tabelle 2.2 gibt einen Überblick darüber, wie sich der virtuelle Speicher aus Sicht eines kleinen Prozesses darstellen könnte, die mit ca. versehenen Bereiche sind dabei variabel.

Beginn	**Ende**	**Größe**	**Art**	**Beschreibung**
01FC 0000	01FF FFFF		Abgebildete Adressen	Geteilte Dll's (<64 KB)
01F8 0000	01FB FFFF	0.4 MiB	Frei	
01AB 0000	01F7 FFFF	ca. 4.9 MiB	Abgebildete Adressen	Systemreserviert
018E F000	01AA FFFF	ca. 1.7 MiB	Frei	
018C 0000	018E EFFF	180 KiB	Privat	Stack (nach oben wachsend)
017C 0000	018B FFFF	1 MiB	Privat	Heap (nach unten wachsend)
0001 E000	017B FFFF	ca. 23 MiB	Frei	
0001 4000	0001 DFFF	40 KiB	Reserviert	Debug informationen
0001 3000	0001 3FFF	4 KiB	Read-Only	Daten Sektion
0001 2000	0001 2FFF	4 KiB		Daten Sektion
0001 1000	0001 1FFF	4 KiB	Read-Only	Code Sektion (Einsprungpunkt)
0001 0000				Erste benutzbare Adresse
0000 0000	0000 FFFF	64 KiB	Reserviert	Systemdaten

Tabelle 2.2: Windows CE 5.02 Anwendungsspeicher

Zu erkennen ist, dass selbst durch die Hinzunahme der Bereiche 60 und 61, um den Speicherdruck durch Bibliotheken zu verringern, der Speicher eines Prozesses immer noch durch weitere Grenzen stark eingeschränkt ist. In einem Test mit einem HTC Touch Pro 2 Gerät mit 196MiB Speicher war es nicht möglich, mehr als 22MiB Heap-Speicher zu allozieren.

2.3.3 Prozess und Threadverwaltung

Der Prozesswechsel läuft bei Windows CE nach einem preemptiven "Round Robin" Verfahren. Die Preemption basiert dabei auf der Thread-Priorität dabei werden Threads mit hoher

Priorität zuerst ausgeführt und es kommt bei Threads mit gleicher Priorität ein Round Robin (Ringverteilung) zum Einsatz, bei welchem jeder Prozess nacheinander eine Zeitscheibe (Quantum) zur Ausführung erhält. Das Standardquantum von Windows CE 5 sind dabei 25ms. [Bol01]

Um Kriterien der Echtzeitfähigkeit einzuhalten, müssen auch Interrupts in einer maximalen Zeit erkannt und verarbeitet werden können. Bei Windows CE werden zu diesem Zweck Interrupts in einen “Interrupt-Service Thread” sowie eine “Interrupt Service Routine” zerteilt. Tritt an einer entsprechenden Hardware-IRQ-Leitung ein Interrupt auf, führt er zum Start der Service-Routine. Diese maskiert gegebenenfalls mehrere auftretende Interrupts und reicht die Identifikationsnummern der Interrupts dann an den Betriebssystemkern weiter, welcher wiederum ein entsprechendes Ereignis erzeugt. Die Interrupt Service Threads werden von diesem Ereignis geweckt und führen die restliche Interruptverarbeitung durch. Da bei dieser Verarbeitung unmöglich geschachtelte Interrupts auftreten können, ist es möglich die Reaktionszeit eines Systems auf einen Interrupt abzuschätzen, beziehungsweise eine feste Obergrenze für diese auszurechnen. Die Mindestanforderungen an ein Echtzeitbetriebssystem werden damit von Windows CE unterstützt. Die Wartezeit, bis ein Interrupt abgearbeitet wird, ist bedingt vorhersagbar und begrenzt und die Ausführungszeiten für Systemaufrufe sind vorhersagbar sowie unabhängig von der Systemlast.

2.4 Die Windows API

Mit dem Windows Application Programming Interface (WinAPI[3]), stellt Microsoft eine Programmierschnittstelle für Windows- Anwendungen zur Verfügung, um auf Funktionen des Systems und verschiedene Systemdienste zugreifen zu können. Beispielsweise bietet die WinAPI Funktionen zur Verwendung von Netzwerkdiensten oder um ein Programminterface darzustellen. Um die plattformübergreifende Software-Entwicklung für Windows NT und Windows CE zu erleichtern, sollte sich diese Schnittstelle zum Betriebssystem möglichst in allen Windows-Varianten ähnlich darstellen, so dass Entwicklern der Umstieg von einer Plattform auf eine andere erleichtert wird. [Mich]

Anmerkung: In manchen Texten wird zwischen der Windows API und der C-Laufzeitumgebung

[3]Im Rahmen dieser Arbeit wird die Abkürzung WinAPI stellvertretend für die deutsche Übersetzung Windows Programmierschnittstelle verwendet, wie es umgangssprachlich üblich ist.

von Windows unterschieden. Da diese aber für Windows CE weder durch unterschiedliche Bibliotheken noch eine klare programmatische Trennung abgegrenzt sind, wird im Folgenden der Begriff WinAPI als Sammlung aller vom Betriebssystem bereitgestellten Funktionen unter Windows Systemen verwendet.

2.4.1 Einschränkungen der WinAPI für Windows CE

Für Windows CE ist jedoch nur ein eingeschränkter Satz der WinAPI verfügbar. Zwar ist dieser größtenteils kompatibel mit der WinAPI, allerdings besitzen Funktionen zum Teil nur eingeschränkte Funktionalität, wodurch wiederum Fehler bei der Portierung von Windows NT Software auf Windows CE auftreten können. In Abschnitt 2.4.1.1 wird dies anhand eines Beispiels genauer erläutert.
Eine weitere wichtige Einschränkung ist der Verzicht auf alle “ASCII-Systemaufrufe” So ist es etwa nur möglich mit Unicode-Parametern Systemaufrufe zu machen (in der Windows NT WinAPI sind alle Funktionen doppelt als ASCII und Unicode-Variante enthalten). Dies stellt zwar funktionell keine Einschränkung dar, erfordert aber, dass man jeden Systemaufruf einer nicht mit Unicode verfassten Software im Quelltext überarbeiten und doppelten Speicherverbrauch (Unicode verwendet 16 Bit pro Zeichen, im Gegensatz zu 8 Bit ASCII) für Texte einrechnen muss. Außerdem wurden viele Funktionen, welche in der WinAPI auf NT Systemen vorhanden sind um die Kompatibilität zu den POSIX-Standards zu erhöhen (z.B. open, close), nicht implementiert. Dazu gehört beispielsweise auch das Konzept von vererbbaren Dateideskriptoren, welche es möglich machen, dass Prozesse ihre Ein- und Ausgaben von anderen Prozessen erhalten oder an diese weiterleiten. Die WinAPI von Windows CE erfüllt so keinerlei Standards, nicht einmal den ISO C-89 (ANSI C) Standard, welcher von Windows NT noch erfüllt werden kann.

Windows CE sieht Konsolenanwendungen zudem nicht vor. Microsoft stellte zuletzt für das Betriebssystem PocketPC 2003 eine Implementierung einer Konsole zur Ein- und Ausgabe zur Verfügung, welche auf neueren Systemen nur noch zur Ausgabe verwendet werden kann. Weitere bemerkenswerte fehlende Funktionalitäten der WinAPI unter Windows CE sind Umgebungsvariablen, das Konzept eines Arbeitsverzeichnisses sowie geschützte, mit dem _s Suffix versehene Varianten verschiedener Standardfunktionen. [Micj] Das Fehlen von Sicherheitsfunktionen führt dazu, dass man bei Verwendung von Software entsprechend

der neuesten Versionen der WinAPI mehr Probleme bei der Portierung hat, als bei Nutzung von veralteten (vom Compiler als deprecated angewarnten) Funktionen.

2.4.1.1 Unterschiedliches Verhalten gleicher Funktionen

Um weiter zu erläutern was damit gemeint ist, dass sich gleiche Funktionen unter Windows CE anders verhalten als unter Windows NT, soll im Folgenden die Funktion “CreateProcess” [Micb] genauer betrachtet werden. Diese zum Starten eines neuen Prozesses zu verwendende Funktion besitzt zwar die gleiche Anzahl an Parametern unter Windows CE wie unter NT, allerdings werden etwa die Parameter Umgebungsvariable oder Vererbung von Datei -Handles nicht unterstützt und ignoriert. Dass diese Funktionalität unter Windows CE nicht vorhanden ist, muss man wissen und bei der Portierung einer Software beachten. Schwerwiegender ist es jedoch, wenn Parameter anders aufgebaut sein müssen. CreateProcess etwa erwartet als zweiten Parameter einen Pointer auf einen Unicode Zeichenkette, welche die Argumente enthält, die beim Aufruf des Prozesses an den neuen Prozess übergeben werden. Unter Windows CE wird an diesen noch der Name des Moduls, welches aufgerufen wird, als erster Wert angefügt, was unter Windows NT nicht der Fall ist, so dass der Name explizit in dem Parameter mit übergeben werden muss. Dadurch gibt es keinen Aufruf von CreateProcess mit Kommandozeilenoptionen, der gleichermaßen auf Windows NT und Windows CE funktioniert. Wenngleich man ein Programm also auf beiden Systemen mit dem gleichen Aufruf kompilieren kann, ohne dass dieser Fehler auffällt, verhält sich die Prozesserstellung unterschiedlich, was in der Regel zu einem Fehler während der Ausführung führen wird.

3 Proprietäre Werkzeuge

Aufbauend auf die grundlegende Beschreibung der Windows CE Plattform werden in diesem Kapitel die proprietären Werkzeuge des Unternehmens Microsoft eingeführt um im darauf folgenden Kapitel freie Alternativen zu diesen Werkzeugen darzustellen.

3.1 Microsoft Visual Studio

Das Microsoft Visual Studio ist das Herzstück der Software-Entwicklung von Microsoft und die einzige offiziell von Microsoft, als dem Hersteller des Betriebssystems, unterstützte Werkzeugsammlung, um Software-Entwicklung sowohl für Windows NT als auch für Windows CE zu betreiben. Das Visual Studio stellt eine integrierte Entwicklungsumgebung (IDE) dar, welche eine Benutzeroberfläche für andere Anwendungen anbietet und diese miteinander verbindet. So ist für einen Entwickler vom Verfassen des Quellcodes, über die Konfiguration der Compiler und Linkeroptionen, bis zur Übertragung von Anwendungen auf ein Testgerät und dem Debuggen alles in einer Oberfläche integriert. Komfortabel und effizient kann dies sein, wenn man in einem Team den kompletten Entwicklungsprozess mit Visual Studio koordiniert und in der Benutzeroberfläche des Studios arbeitet. Allerdings ist diese Art der integrierten Entwicklung auch recht unflexibel und aufwändig mit existierenden Softwareprojekten zu verbinden.

Vor der Version 2005 des Visual Studio gab es noch die Software Microsoft Embedded Visual C++ als Alternative zur Entwicklung von C/C++ Code. Embedded Visual C++ wird zwar noch zum kostenlosen Herunterladen auf der Webseite von Microsoft angeboten, die Software läuft allerdings weder auf neueren Windows-Systemen (nach Windows XP), noch werden “Software Developement Kits” (SDK), also Software Entwicklungspakete, für Win-

dows Mobile angeboten.

Visual Studio wird überwiegend in drei verschiedenen Varianten vertrieben, einer Express Edition sowie den Varianten Standard und Professional. Die Express Edition ist zwar zum kostenlosen Herunterladen verfügbar, ermöglicht jedoch nur einen sehr eingeschränkten Funktionsumfang, mit welchem beispielsweise die Entwicklung von Software ausschließlich für auf dem Windows NT Kern basierende Windows Betriebssysteme auf x64 und x86 Architekturen möglich ist. Mit den kostenpflichtigen Standard- und Professional-Lizenzen hingegen wird durch Werkzeuge, welche auch andere Architekturen als x64 und x86 unterstützen, auch die Entwicklung für Windows CE Systeme ermöglicht. Für nicht kommerzielle Entwickler stellt diese Lizenzpolitik allerdings eine hohe Einstiegshürde im Hinblick auf die Software-Entwicklung für Windows CE dar und erhöht dementsprechend die Kosten und den Wartungsaufwand einer Freien Software, da weniger Entwickler Zugang zu den benötigten Werkzeugen haben um die Softwarebasis, den Quellcode, weiterzuentwickeln.

3.1.1 Der Visual C Compiler

Neben der integrierten Benutzeroberfläche gehören zu den Kernkomponenten von Visual Studio die Programme zum Erstellen von ausführbaren Dateien. Das Programm CL (Compiler-Linker) dient dabei als Einstiegspunkt, ähnlich dem in Abschnitt 4.1.1 ausführlich beschriebenen GCC, welches weitere Programme, wie beispielsweise den Linker, aufruft. Diese Werkzeuge wurden auch dazu entwickelt, für andere Architekturen als diejenige, mit der sie ausgeführt werden, Programme zu erstellen und werden in Versionen für x86, x64 für Windows NT sowie ARM, x86, SH-4, MIPS16, MIPS32 und MIPS64 für Windows CE ausgeliefert.

3.2 Microsoft Software Developement Kits

Durch die in Abschnitt 2.3.2 beschriebene Variabilität von Windows CE-Plattformen wird bei Microsoft Visual Studio nur ein kleiner Teil der Windows API mit ausgeliefert. Dieser umfasst gerade nur Basisfunktionen, welche zum Starten von Programmen unabdingbar sind. Nahezu alle Funktionen der WinAPI oder auch spezielle Schnittstellen von Geräte-

software müssen in den bereits erwähnten Software-Entwicklungspaketen einem Entwickler zur Verfügung gestellt werden. Für mit dem Plattform-Builder 2.3.1 angepasste Plattformen liegt es in der Verantwortung der Hersteller, für ihre Plattformen ein entsprechendes Paket zusammen zu stellen, um mit modifizierten Header-Dateien die passenden Schnittstelleninformationen bereit zu stellen. Für Windows Mobile bietet Microsoft vorgefertigte SDKs zum Herunterladen an, zu deren Installation man allerdings eine Microsoft Visual Studio Standard- oder Professional-Lizenz benötigt.
Diese SDKs enthalten neben den Exportdefinitionsdateien der auf dem Gerät vorhandenen Funktionsbibliotheken, in denen diverse Systemaufrufe definiert sind, auch die Header mit den zugehörigen Deklarationen, welche benötigt werden, um die Funktionsbibliotheken aus anderen Anwendungen heraus verwenden zu können.

3.2.1 Buildkonfigurationen

Zur Unterstützung verschiedener Buildkonfigurationen, zur genaueren Beschreibung, wie der Übersetzungsprozess einer Software durchgeführt werden soll und zur Information, welche Dateien relevanten Quellcode enthalten, gibt es für Microsoft Visual Studio so genannte Projektdateien. Diese enthalten Informationen über die Compiler / Linker-Optionen sowie Regeln darüber, wie eine Anwendung für ein bestimmtes Zielsystem gegebenenfalls bereitgestellt werden soll. Die Projektdateien lassen sich nur über die Benutzeroberfläche des Visual Studio effektiv bearbeiten, was gerade bei vielen unterschiedlichen Konfigurationen für verschiedene Plattformen recht aufwändig sein kann, da die Menüs zur Bearbeitung der Konfiguration komplex und entsprechend stark ineinander verschachtelt sind und man nur schwierig den kompletten Überblick darüber behält, was genau eingestellt wurde. Die Fehleranfälligkeit wird dadurch erhöht und wenn man bestehende Software in dieses Buildsystem einbringen möchte, ist dies arbeitsintensiv und somit auch teuer. Um diesen Vorgang zu umgehen, wird bei Visual Studio mit der Software NMake auch ein Programm mitgeliefert, welches mit der Freien Software GNU Make vergleichbar ist und Makefiles mit Buildkommandos abarbeitet. Eine automatische Konfiguration für eine bestimmte Plattform, wie es das in Abschnitt 4.4.2 vorzustellende GNU Buildsystem ermöglicht, kann Visual Studio nicht leisten.

3.2.2 Visual Studio Debugger

Der Visual Studio Debugger ist ein in die Benutzeroberfläche von Visual Studio eingebettetes Werkzeug, um den Ablauf eines Programms nachzuvollziehen und zu steuern. Um den Visual Studio Debugger mit Software verwenden zu können, welche auf einem anderen System abläuft als der Debugger selbst, benötigt man die Professional-Variante des Visual Studio Softwarepakets. In dieser Variante besitzt er die zur Entwicklung von embedded Systemen notwendige Fähigkeit, sich über Active Sync und die Remote API mit einem Gerät zu verbinden und den Ablauf eines Programms so fernzusteuern. Bei Konsolenanwendungen ermöglicht es die Active Sync Verbindung außerdem Konsolenausgaben, welche auf dem Gerät aufgrund einer mangelnden Konsole (seit Windows CE 4) nicht angezeigt werden, innerhalb der Visual Studio-Benutzeroberfläche zu verfolgen.

Für die Verwendung dieser Möglichkeiten des Debuggers reicht es allerdings nicht, sein Projekt über Makefiles zu beschreiben, sondern man benötigt dazu unbedingt eine Build- und Bereitstellungskonfiguration in Form von Visual Studio-Projektdateien. Eine Anwendung kann dann zwar unmittelbar nach dem Erstellen auf das Zielsystem übertragen und dort getestet werden, allerdings kann die Konfiguration und die Bereitstellung von Abhängigkeiten gerade bei größeren Projekten, die nicht in einer Visual Studio-Projektdatei zusammengefasst sind, aufwändig werden. Für diese Form der Bereitstellung und des Debuggens benötigt zudem jeder Tester einer Software eine vollständig eingerichtete Visual Studio-Instanz mit direkter USB- oder Bluetooth-Verbindung zu einem Zielgerät.

3.2.3 Paketierung von Software

Da es in der Software-Entwicklung ebenso wie in anderen Professionen üblich ist, Entwicklung und Qualitätssicherung zu trennen, benötigt man geeignete Mechanismen die Programmversionen der Programmierer an andere Personen weiterzureichen. Gerade bei Freier Software, wo das Testen in der Regel von einer Gemeinschaft interessierter Nutzer durchgeführt wird, sollte es möglich sein, mit schneller und unkomplizierter Paketierung Software jederzeit an interessierte Menschen ausliefern zu können und ihnen so zu ermöglichen, durch dieses Testen an der Software-Entwicklung teilzuhaben.

Um Softwarepakete für Windows CE bereitzustellen, so dass diese sich nativ in vorhandene Paketverwaltungssysteme einfügen, muss man das Microsoft Cabinet (CAB) Format verwenden. Eine Anwendung müsste ansonsten selbst eine Verwaltung implementieren, so dass die Applikation beispielsweise auch wieder deinstalliert werden kann, und würde nicht mit den vom System bereitgestellten Mechanismen zur Paketverwaltung harmonieren. Je nach eingestellter Sicherheitsrichtlinie auf dem Gerät sind signierte CAB-Installer zudem die einzige Möglichkeit, ausführbare Software auf Windows CE Geräten zu installieren.

Mittels Microsoft Visual Studio gibt es zwei Arten Softwarepakete für Windows Mobile zu erstellen; zum einen die Möglichkeit einer Installationsanwendung für die Ausführung auf einem Windows NT Desktoprechner, der einen Installationsdialog anbieten kann, zum anderen mit einer Cabinet-Installationsdatei, welche direkt auf dem Gerät ausgeführt werden muss. Die Installationsanwendung stellt dabei keine Funktionalität für das Gerät selbst zur Verfügung, sondern überträgt nur einen Cabinet-Installer auf das Mobilgerät und führt diesen mittels der Microsoft Remote API dort aus. Ausführbare Installationsanwendungen sind dabei dafür geeignet, dass der Nutzer das Beschaffen und Installieren der Software von einem anderen System (etwa einem Desktop-Rechner) vornehmen kann und man auf dem Gerät selbst nichts aus dem Dateisystem heraus starten muss. Außerdem ist es damit möglich, auf dem ausführenden System, welches eine Verbindung mit dem Gerät herstellt, noch zusätzliche Software zu installieren. Geht man diesen Weg, benötigt man allerdings einen Computer mit einem auf Windows NT basierenden Betriebssystem und eine Verbindung des Zielgerätes mit der Software Active Sync oder dem so genannten Windows Mobile Device Center. Unabhängig davon, wie die Software auf das Gerät übertragen wird, benötigt das Softwareverwaltungssystem von Windows CE immer ein CAB-Archiv, welches die Informationen zur Installation und die Daten der Anwendung enthält.

3.2.3.1 Microsoft CAB Wizard

Der Microsoft CAB Wizard ist eine im Visual Studio-Paket enthaltene Anwendung, welche in der integrierten Entwicklungsumgebung dazu verwendet wird, Installationsdateien für ein Softwareprojekt zu erstellen, sofern man dieses als “Intelligentes Gerät”-Projekt konfiguriert. CAB Wizard benötigt zur Paketierung von Software für Windows Mobile eine Datei im INF Format, die speziell formatiert sein muss. Diese Datei wird bei der integrierten Ent-

wicklung von Visual Studio generiert und der CAB Wizard konvertiert die Informationen daraus zu einer XML Datei, welche wiederum die vom Windows CE-Paketmanagement benötigten Informationen enthält. Diese so genannte _setup.xml wird anschließend zusammen mit den Dateien, die in dem Paket enthalten sein sollen, zu einem CAB-Archiv verbunden. Mit diesem Schritt wird aus einem CAB-Archiv (welches auch für andere Daten verwendet werden kann) ein von Windows CE akzeptiertes Installationspaket.

3.2.4 Signieren von Paketen

Um die Integrität von Installationspaketen zu gewährleisten, sieht Microsoft die Option vor, diese mittels eines X.509 Zertifikates zu signieren. Das hierbei zu verwendende Verfahren wird als “Authenticode” [Mica] bezeichnet und es kommen für CE- und NT-Systeme die gleichen Werkzeuge zur Anwendung. Die notwendigen Werkzeuge erfordern von daher keine lizenzierte Visual Studio Version, sondern werden sowohl in den Windows Mobile Entwicklungspaketen, als auch in den Entwicklungspaketen für die diversen Desktopsysteme ausgeliefert.

Microsoft bietet die Werkzeuge “makecert” und “pvk2pfx” an, welche die Erstellung und Konvertierung eines Zertifikats im PKCS #7 Standard [RSA] ermöglichen. Das Erstellen der Signatur erfolgt durch das “signtool”. Hat man damit eine Datei signiert, ist es möglich dass diese auf dem Gerät vor der Ausführung hinsichtlich ihrer Integrität überprüft werden kann. Dabei wird eine Hash-Summe (SHA1 oder MD5) der zu signierenden Datei generiert und diese Summe anschließend verschlüsselt. Sowohl die verschlüsselte Hash-Summe als auch die Zertifikatsdaten werden dabei von “signtool” direkt als PKCS #7 Objekt in die zu signierende Datei eingebettet, so dass keine zusätzlichen Dateien zur Überprüfung der Authentizität und Integrität eines Programms benötigt werden.

Damit die Authentizität eines Software-Herausgebers auf dem Gerät anerkannt wird, benötigt man ein von einer vertrauenswürdigen Zertifizierungsstelle signiertes Zertifikat. Diese Zertifikate, die Windows CE als vertrauenswürdig betrachtet, sind im so genannten “Certificate Store” im Gerät hinterlegt. Bei Windows Mobile ab Version 5.0 sind dies die Zertifikate folgender Zertifizierungsstellen [Mice]:

- Class 2 Public Primary Certification Authority (VeriSign, Inc.)

- Class 3 Public Primary Certification Authority (VeriSign, Inc.)
- Entrust.net Certification Authority (2048)
- Entrust.net Secure Server Certification Authority
- Equifax Secure Certification Authority
- GlobalSign Root CA
- GTE CyberTrust Global Root
- GTE CyberTrust Root
- Secure Server Certification Authority (RSA)
- Thawte Premium Server CA
- Thawte Server CA

Einem Mobilfunkbetreiber oder Gerätehersteller wird die Option gegeben, Zertifikate, die nicht von einer dieser Stellen signiert wurden, zu verbieten. Wenn das Zertifikat akzeptiert wird, kann man ein individuelles Zertifikat als Wurzelzertifikat hinzufügen, um so, beispielsweise in einem Unternehmen, Software auf den Geräten der Mitarbeiter zu erlauben, welche ein vertrauenswürdiges Softwareunternehmen bereitstellt. Dass eine Software auf allen Windows Mobile Geräten als signiert erkannt wird (und installiert werden darf), erreicht man allerdings nur, wenn man ein Zertifikat einer dieser auf allen Geräten hinterlegten Zertifizierungsstellen verwendet.

4 Freie Software Werkzeuge

In diesem Kapitel werden freie Software Werkzeuge als Alternativen zu den proprietären Werkzeugen aus 3 vorgestellt und erläutert. Dazu wird die GNU Compiler Collection (GCC) als Kernkomponente zum Erstellen Freier Software eingeführt und anschließend werden weitere Werkzeuge beschrieben, die zum Aufbau funktionierender Entwicklungszyklen notwendig sind. Das Kapitel schließt mit der Beschreibung eines beispielhaften Entwicklungsprozesses mit freien Werkzeugen.

4.1 Die GNU Compiler Collection

Als im Jahr 1984 Richard Stallmann die GNU Initiative ins Leben rief, die es sich als Ziel setzte, ein nur mit Freier Software geschriebenes und nur aus Freier Software bestehendes UNIX-artiges Betriebssystem zu erstellen, wurden für dieses Vorhaben freie Werkzeuge zur Software-Entwicklung benötigt. Da es zu dieser Zeit noch keinen als Freie Software vorliegenden C-Compiler gab, war es eine der ersten und wichtigsten Aufgaben der GNU Initiative, einen solchen von Grund auf neu zu entwickeln, so dass mithilfe von Freien Software Werkzeugen Freie Software-Entwicklung betrieben werden und ein freies Betriebssystem entstehen konnte.
1987 wurde schließlich die erste Version der GCC (welche sich dann selbst kompilieren konnte) veröffentlicht und damit die Grundlage für Freie Software, wie wir sie heutzutage verwenden, geschaffen. [Gou04]

Anfänglich ein reiner C-Compiler entwickelte sich das Programm GCC von einem spezifisch für die Programmiersprache C verfassten Compiler (dem GNU C-Compiler) hin zu einer Steueranwendung, welche den Ablauf der Quellcode-Übersetzung einer Vielzahl von

Programmen kontrolliert und dabei Funktionen anderer (in der GCC vorhandenen) Anwendungen unterstützt. Zum einen umfasst sie sehr portable C-Anwendungen, die auf den meisten der heutzutage verbreiteten Systeme (sowie auf vielen historischen) betrieben werden können. Zum anderen bietet sie die Möglichkeit des “Cross-Compilings”, also Programme für andere Rechnerarchitekturen oder andere Betriebssysteme zu erstellen, als für diejenigen auf welchen sie selbst ausgeführt wird. Da es ebenso möglich ist mit der GCC selbst die GCC zu übersetzen, kann man so auch “native” Compiler für diese Zielplattformen erzeugen. [Gou04]

Neben dem umfangreichen Funktionsumfang und den vielfältigen und leistungsstarken Optimierungsoptionen ist das wichtigste Merkmal der GCC jedoch ihre Lizenz als Freie Software. So konnte die GCC von einer großen Gemeinschaft von Firmen und Entwicklern weiterentwickelt und auf neue Plattformen portiert werden und erst dadurch eine derartige Reife und Verbreitung finden, wie es heutzutage der Fall ist. So ist es mittlerweile schwierig ein System zu finden, welches die GCC nicht unterstützt. Während Unternehmen wie Apple oder Sun sie in ihre Plattformen integrieren, ist es hauptsächlich Microsoft, das versucht seine eigenen Werkzeuge gegenüber der GCC durchzusetzen.

4.1.1 Aufbau der GCC

Wie bereits in der Einführung zur GCC erwähnt ist die GCC selbst nur eine Verbindung einzelner Unterkomponenten, welche genau einen spezifischen Zweck erfüllen. Dies hat einerseits den Vorteil, dass man als Entwickler den Übersetzungsprozess seiner Anwendung genauer kontrollieren kann, indem man selbst jedes Werkzeug einzeln anwendet (oder die Konfiguration des Übersetzungsprozesses anderen Werkzeugen überträgt), und andererseits den Vorteil einer erhöhten Wartbarkeit. Die GCC, als eine der ältesten und kompliziertesten Freie Software Initiativen, wäre als einzelne gewaltige Anwendung nur schwer wartbar. Viele ihrer Komponenten sind zudem vollkommen unabhängig und können somit auch von Entwicklern weiterentwickelt und verbessert werden, die weder den kompletten Überblick über die Software haben noch diese in ihrer Gänze verstehen. Die GCC Entwicklung selbst wird von einem 12-köpfigen Lenkungskomitee geleitet, welches versucht den Überblick und Zusammenhang aller Komponenten im Auge zu behalten. [Frec]
Für die Werkzeuge für Windows CE bedeutet dies, dass nur an sehr wenigen systemspezi-

fischen Komponenten Änderungen vorgenommen werden mussten, etwa am Linker und am Relocator. Doch nur aufgrund der Freiheit der GCC ist es überhaupt möglich, dass Freie Software Werkzeuge für Windows CE existieren können. Im April 2010 bestand der Quellcode der für Windows CE angepassten GCC Werkzeuge aus 6416635 Zeilen; ein nahezu unmöglich zu reproduzierendes Werk freien Wissens. xy zeigt die Zusammensetzung der Codebasis der CEGCC Initiative und auch wenn die Zeilenanzahl selbst nicht viel aussagt, vermittelt sie doch einen gewissen Eindruck, von welch einem gewaltigen Werk freier Kollaboration gesprochen wird.

4.1.2 Ablauf des Übersetzungsprozesses

Ruft man die Anwendung GCC auf, hat man die Möglichkeit eine Kommandozeile zu übergeben, welche Parameter für weitere Programme enthält. So ist es durch einen Aufruf des Programms GCC möglich, von sprachspezifischen Optionen bis zu Optionen zur Verknüpfung einer Anwendung alle zur gewünschten Erstellung des ausführbaren Codes notwendigen Optionen einzustellen. Grafik 4.1 gibt einen schematischen Überblick, wie eine Software bestehend aus C- und Assemblercode mithilfe der GNU Compiler Collection erstellt wird.

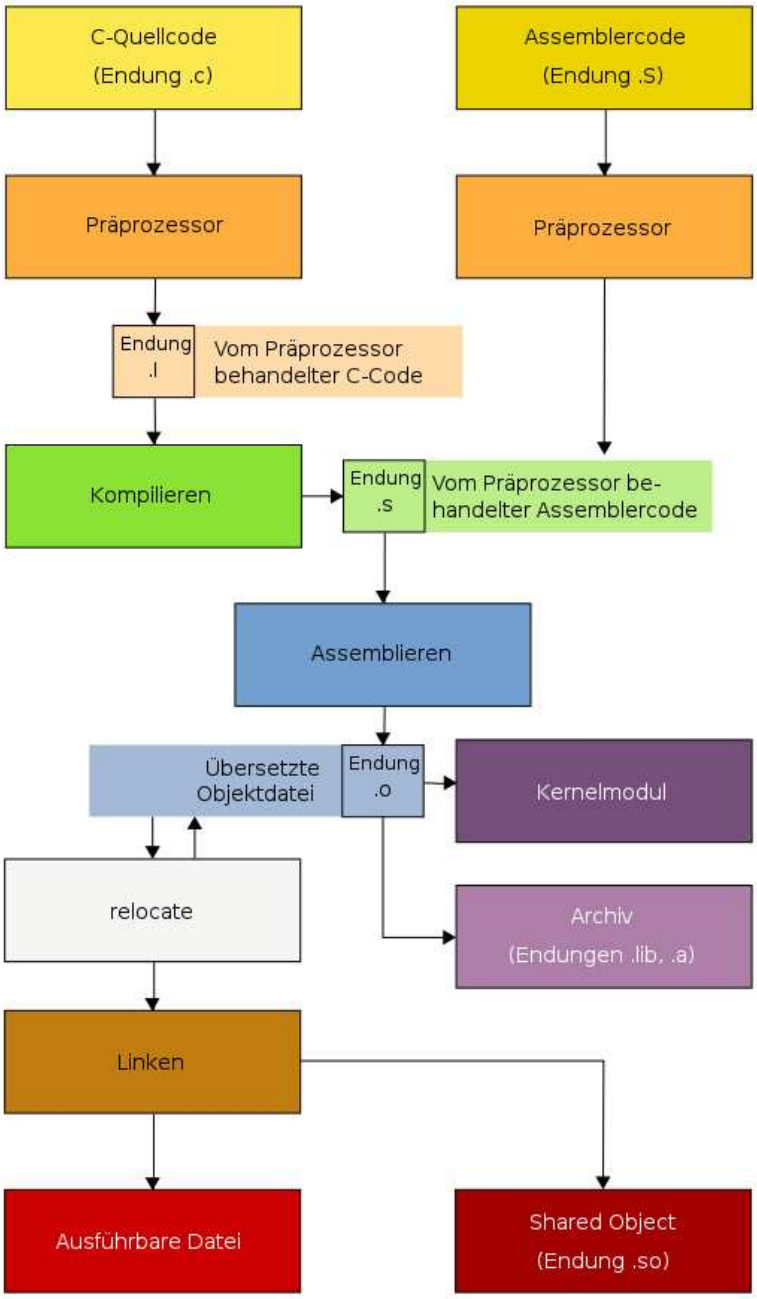


Abbildung 4.1: Schematischer Ablauf der GCC([Sve])

Nach dem Aufruf wird automatisch untersucht, in welcher Programmiersprache die übergebene Quelldatei vorliegt, und der entsprechende Präprozessor wird angewandt. Dieser wertet spezielle Anweisungen im Code aus, um etwa Header-Dateien einzubinden und durch “Compile-Time-Schalter”-bedingte Codeübersetzung zu ermöglichen. Der vorverarbeitete Code wird danach an den passenden Sprachcompiler übergeben und in Assembler übersetzt. Diese in Assemblersprache vorliegende Ausgabe des Compilers wird anschließend an den

Assembler übergeben, welcher daraus den Binärcode, der auf dem Zielsystem direkt ausgeführt werden kann, erstellt. Die ausführbaren Daten können im nächsten Schritt, je nachdem was das Ziel der Übersetzung sein soll, weiter verarbeitet werden, so dass aus ihnen entweder eine ausführbare Datei, ein statisches Objekt oder ein geteiltes Objekt erstellt werden kann. Dabei werden entweder im Zuge der Relocation die einzelnen Objektdateien in gemeinsame Sektionen zusammengeführt und die Adressen entsprechend statisch gesetzt, so dass diese in einer einzelnen ausführbaren Datei verbunden werden oder es können, wenn man geteilte Objekte verwendet, im Windows Sprachgebrauch als Dynamic Linked Librarys(DLL) bezeichnet, durch Symbolauflösung externe Funktionen aus anderen Programmen (oder Bibliotheken) eingefügt werden. [Gro02]

4.1.3 GNU Binutils

Neben den in Grafik 4.1 dargestellten Werkzeugen zum unmittelbaren Erstellen einer ausführbaren Anwendung enthält die GNU Compiler Collection noch weitere Werkzeuge, welche teilweise getrennt entwickelt werden und unter dem Namen GNU Binutils bekannt sind. Diese werden von dem Programm GCC nach Bedarf aufgerufen und sind Teil der gesamten Werkzeugsammlung, die die GCC bildet. Im Folgenden werden sie aus diesem Grund nicht gesondert unter der Bezeichnung Binutils erwähnt. [Fred]
Die GNU Binutils enthalten unter anderem[1]:

- ar - Erstellt, modifiziert und extrahiert Objektarchive.
- c++filt - Löst C++ Symbole durch einen Filter auf.
- dlltool - Erstellt Dateien zum Verarbeiten und Verwenden von DLLs .
- gprof - Erhebt Informationen zum genauen Programmablauf.
- nm - Listet die in Objektdateien enthaltenen Symbole auf.
- objcopy - Kopiert und übersetzt Objektdateien.
- objdump - Extrahiert die in Objektdateien enthaltenen Informationen.
- ranlib - Generiert einen Index für die in einem Archiv enthaltenen Objekte.

[1] Der Übersicht halber seien an dieser Stelle nur die für Windows CE relevanten, von der CEGCC Initiative portierten Programme genannt

- readelf - Extrahiert Informationen aus Objekten im ELF Format, etwa ausführbaren Linux Anwendungen.
- size - Listet die Sektionsgrößen von Objekten oder Archiven auf.
- strings - Listet die in Objekten enthaltenen Zeichenketten auf.
- strip - Löscht nicht unbedingt notwendige Symbole.
- windmc - Compiler für Nachrichtenressourcen(.mc).
- windres - Compiler für Ressource-Definitionsdateien (.rc).

4.2 Das CEGCC Projekt

Aufgrund der freien Natur der GNU Compiler Collection konnten verschiedene Projekte daran arbeiten, diese Werkzeuge für Windows CE anzupassen. Eines der ersten Projekte dazu war im Jahr 2003 GNUWINCE. [Unb] Es bot auf Basis des GNU C-Compilers sowie des GNU Assemblers und des GNU Linkers eine erste Möglichkeit nur mit Freien Software Werkzeugen ausführbaren Code für Windows CE herzustellen.

Diese ersten Werkzeuge verwendeten noch Newlib als Standard C-Bibliothek und bildeten eine POSIX-artige Umgebung für Programme auf Windows CE. Zwar konnte man damit vermeiden Windows CE Software spezifisch auf das WinAPI anzupassen, allerdings hatte man dann auch nicht die Möglichkeit, diese Funktionen spezifisch zu verwenden oder vorhandene Windows CE Anwendungen mit Freier Software weiterzuentwickeln.

Das nächste Projekt, welches es sich zum Ziel machte Freie Software Werkzeuge für Windows CE zu erstellen, war das Projekt wince-xcompile, das ebenfalls die Newlib C-Bibliothek nutzte, um vorhandene POSIX-kompatible Software auf Windows CE Geräten zu verwenden. [Bac] Dieses im Dezember 2005 von Danny Backx gegründete Projekt erreichte erstmals nennenswerte Verbreitung. Bald darauf, im Januar 2006, gründete Pedro Alves das CEGCC Projekt, welches nun nicht mehr nur Newlib-basierte Entwicklung ermöglichte sondern auch native WinAPI unterstützte. [CEG] Danny Backx und weitere Entwickler von wince-xcompile schlossen sich dem CEGCC Projekt an, das seitdem große Fortschritte gemacht hat und weiterhin unter Leitung der beiden Gründer aktiv weiterentwickelt wird.

Der vom CEGCC Projekt gewählte Ansatz war, ein Set von gleichen Werkzeugen zu verwenden und je nachdem, ob man native Entwicklung unter Verwendung der WinAPI Systemaufrufe betreiben oder auf Basis von Newlib POSIX-konforme GNU/Linux Anwendungen auf Windows CE Geräte portieren wollte, gegen unterschiedliche Bibliotheken zu linken. Durch diese Möglichkeit konnten sich Initiativen mit unterschiedlichen Zielen vereinigen und eine gemeinsame Codebasis, mit der Software kompiliert werden kann, entwickeln, aber bei der Software-Entwicklung selbst unterschiedliche Entwicklungsansätze ermöglichen..

4.2.1 Newlib

Im vorangegangen Abschnitt wurde Newlib oft genannt als Ersatz für die WinAPI. Newlib ist eine, von dem Unternehmen Red Hat verwaltete, C-Bibliothek, welche speziell für embedded Systeme konzipiert wurde. Im Gegensatz zur WinAPI versucht Newlib möglichst viele POSIX Standards einzuhalten, was die Entwicklung für Programmierer aus der GNU/Linux Welt deutlich erleichtert sowie die Portierung vorhandener GNU/Linux Software besser ermöglicht. [Gat02] Nahezu alle Programme des GNU Systems und viele weitere Anwendungen, welche das GNU Buildsystem verwenden, können mit den von Newlib bereitgestellten Funktionen erstellt werden. Dabei müssen Anwendungen allerdings ihre eigene C-Bibliothek verwenden und können nicht oder nur schlecht von Funktionen der Windows API Gebrauch machen, da diese nicht kompatibel sind. Newlib ist Freie Software, was es ermöglicht, dass jeder Interessierte sie auf andere Plattformen portieren kann. Dies hat, ähnlich wie bei der GCC, den Effekt, dass Newlib mittlerweile in nahezu allen verbreiteten Systemen eingesetzt werden kann. Ursprünglich wurde die Bibliothek von der Firma Cygnus Solutions (welche heute Red Hat gehört) entwickelt. Sie wird unter anderem auch in Cygwin verwendet, um unter Windows eine Kompatibilitätsschicht für GNU/Linux Anwendungen bereitzustellen. Die Lizenzierung von Newlib ist GNU-kompatibel und Teile der Bibliothek sind entweder als Public Domain oder unter nicht schützenden Lizenzen lizenziert. [Red]

4.2.2 Minimalist GNU for Windows

Die Minimalist GNU for Windows (MinGW) Bibliotheken stellen im Gegensatz zu Newlib keine Kompatibilitätsschicht für Windows zur Verfügung, um Linux oder Unix System-

aufrufe zu ermöglichen oder POSIX Kompatibilität zu erreichen. Die Software sollte ursprünglich, ähnliche wie Cygwin, nur eine minimale Umgebung für Software der GNU Initiative auf Windows bereitstellen, wurde jedoch durch Aufnahme frei verfügbarer Header- und Importbibliotheken für die WinAPI zu einem wichtigen Werkzeug, um native Software-Entwicklung für Windows mit Hilfe der GCC zu betreiben. [Mina] Mit den Headern der MinGW Initiative werden Schnittstellen zu den Funktionen der WinAPI beschrieben, welche bei der Softwareerstellung verwendet werden können, so dass damit erstellte Software zur Laufzeit auf die Funktionen in den Bibliotheken eines Windowssystems zugreifen kann. Dies ist die gleiche Art der Softwareerstellung, die auch bei Visual Studio angewandt wird, um WinAPI Funktionen zu verwenden. Die Bibliotheken müssen dazu nicht während des Linkens des Programms zur Verfügung stehen, sondern nur Informationen darüber, an welcher Stelle zur Laufzeit welches Symbol (Funktion oder Variable) adressiert werden soll. Die Auflösung dieser Symbolnamen in Adressen wird dann während der Ausführung vom “Loader” des Betriebssystems übernommen. Die Symbolnamen und benötigten Informationen sind in DLL Dateien oder geteilten Objekten gespeichert und können extrahiert werden. Durch die so vorhandene Trennung von Deklaration und Definition (die Deklarationen müssen als Freie Software zur Verfügung stehen, die Definitionen können als Binärdaten vorliegen) ist es möglich, Freie Software-Entwicklung auch für proprietäre Systeme zu betreiben, sofern man eigene, nicht proprietäre, Deklarationen verwendet. [Minb] Die von CEGGC verwendeten MinGW Header und Bibliotheken wurden dabei speziell an Windows CE angepasst, um möglichst den Funktionsumfang der von Microsoft bereitgestellten Bibliotheken zu erreichen. Zusätzlich erweiterte das CEGCC Projekt diese Bibliotheken um weitere 275 Funktionen, welche als Teil von libmingwex Funktionalitäten bieten, die nicht von Microsoft bereitgestellt werden. Dabei wurden bewusst nur solche Funktionen implementiert, die keine Probleme erzeugen, wenn man sie zusammen mit von Visual Studio erstellten Anwendungen verwendet.

4.3 Erweiterung der MinGW32ce WinAPI

Die Implementierung der MinGW WinAPI beruht darauf, dass der Laufzeitaufruf von Funktionen geschützter Software, beispielsweise durch dynamisches Linken, nicht, etwa durch

das Urheberrecht, verboten werden kann. Ist es erlaubt Windows CE zu verwenden, ist es auch erlaubt Software zu verwenden, welche wiederum Funktionen in den Windows CE Systembibliotheken aufruft. Allerdings kann Microsoft die Header-Dateien, die Deklarationen über die Art der Schnittstellen sowie verschiedene Wertedefinitionen und kleinere “inline” Funktionen[2]enthalten, schützen und tut dies auch.

4.3.1 Exportdefinitionsdateien

Um Funktionen der Windows CE Systembibliotheken mithilfe der GCC in einer eigenen Anwendung einbinden zu können, benötigt man Informationen darüber, welche Schnittstellen diese zur Verfügung stellen. Diese so genannten Exportsymbole werden innerhalb von DLL-Dateien in Exporttabellen gespeichert und geben Auskunft darüber, welche Symbole von anderen Programmen aus adressiert werden können. Die Exportsymbole besitzen einen eindeutigen Namen, mit dessen Hilfe die Speicheradresse der sich hinter dem Symbol verbergenden Adresse einer Variablen oder einer Funktion zur Laufzeit aufgelöst werden kann. Das bereits in 4.1.3 erwähnte dlltool bietet die Option diese Informationen aus einer DLL-Datei zu extrahieren. Mit dem Kommando:

```
dlltool -z somedll.def --export-all-symbols somedll.dll
```

ist es möglich, die Exporttabellen aus beliebigen DLL-Dateien zu extrahieren. Wendet man dies auf alle Dateien, die Teile der WinAPI für Windows CE enthalten, an, kann man daraus, wiederum mithilfe des dlltools, Objektarchive erzeugen Diese Objektarchive werden bei der Kompilierung der CEGCC Werkzeugkette jeweils erneut erzeugt; so fließen Änderungen an den Definitionsdateien direkt in die Ausgabebibliotheken mit ein. Dabei werden aus den Definitionsdateien, welche die Informationen der Exporttabellen enthalten, Archive von Objektdateien erzeugt, die es dem Linker ermöglichen Software in einer Art zu binden, dass dieser zur Laufzeit auf die entsprechenden Funktionen verweist. Damit dies jedoch funktioniert, benötigt man allerdings noch mehr Informationen als nur die Information, welche Datei die Funktion unter welchem Symbolnamen anbietet, was zu den Unterschieden in der Software-Entwicklung mit Microsoft Visual Studio und CEGCC führt.

[2]Inline Funktionen definieren Code, der während der Übersetzung an den Ort kopiert wird an welchem er aufgerufen wird

4.3.2 Erstellen zugehöriger Header

Wie bereits erwähnt, können Funktionsaufrufe als feste Adressen innerhalb von geladenen Bibliotheken nicht urheberrechtlich geschützt werden. Dies ist bei den zugehörigen Header-Dateien, welche die exportierten Funktionen deklarieren und zusätzliche Definitionen enthalten können, jedoch möglich. Aus diesem Grund wurde beispielsweise auch die in Abschnitt 2.2.1.2 vorgestellte LGPL des GNU Projekts für die vom GNU System standardmäßig verwendete C-Bibliothek glibc entworfen, so dass andere Programme die in den Header-Dateien enthaltenen Informationen verwenden können, ohne selbst als Folge des Copylefts unter der GPL veröffentlicht werden zu müssen.

Möchte man die MinGW(ce) API erweitern, muss man diese Header-Dateien selbst verfassen. Zumeist ist dies dadurch möglich, dass man die API Dokumentation des “Microsoft Developer Networks’ [Micf] für die Reproduktion entsprechender Informationen verwendet, oder aber man benutzt die Microsoft Werkzeuge zur Erstellung von Software, welche diese Funktionen aufruft, um aus deren Rückgabewerten deren Arbeitsweise, Deklaration sowie weitere Definitionen rekonstruieren zu können.

4.3.2.1 Beispiel: Headerentwicklung durch Rekonstruktion

Bei der Erstellung von Qt mit den Mingw32ce Bibliotheken fällt auf, dass für die Funktion:

```
BOOL IsProcessorFeaturePresent(
  DWORD dwProcessorFeature
);
```

zwar die Deklaration an der entsprechenden Stelle in Winbase.h enthalten ist und auch mit der passenden Adresse innerhalb der Coredll.dll verknüpft wird, aber Definitionen darüber fehlen, welches Prozessorfeature mit welchem Zahlenwert von dwProcessorFeature überprüft werden kann. In der Microsoft Dokumentation ist diese Funktion vorbildlich dokumentiert. So sind auf der Webseite [Micc] Prozessorfeatures aufgelistet, die überprüft werden können, und es ist angegeben, in welcher Bibliothek sich die Funktion seit welcher Version befindet, sowie welcher Header die Definitionen enthält. Allerdings ist nicht dokumentiert, welchem Zahlenwert diese entsprechen müssen, damit sie korrekt von der Funktion verar-

beitet werden. Der Header (Winbase.h), in dem die Werte definiert sind, ist urheberrechtlich geschützt und kann dementsprechend nicht in die MinGW API kopiert werden. Erstellt man allerdings ein Programm, um sich die Werte der Definitionen ausgeben zu lassen, kann man diese Ausgabe dazu verwenden, diese entsprechend in der Winbase.h der MinGW32ce API zu definieren.

4.3.3 Limitationen dieser Herangehensweise

Die hier vorgestellten Beispiele sind der einfacheren Art, da die WinAPI für Windows CE leider nicht vollständig dokumentiert oder ähnlich rekonstruierbar ist. Entwickelt man Software von Grund auf neu, kann man es in der Regel vermeiden nicht oder nur unzureichend dokumentierte Funktionen zu verwenden. Grundsätzlich stellt es jedoch ein Risiko dar, wenn man existierende Software portieren oder verwenden möchte, dass diese Windows CE Funktionen erwartet, die zwar mit den Microsoft Bibliotheken verfügbar sind, aber in den MinGW Headern nicht oder nicht vollständig beschrieben werden. In diesem Fall bleibt einem nur übrig die Kompilierung ohne die erwarteten Funktionen durchzuführen und wann immer Fehler auftreten diese zu beheben. In 5.2 wird darauf eingegangen, wo das Qt Framework eine solche nicht dokumentierte Header-Datei verwendet und wie man mit den daraus entstehenden Fehlern umgehen kann.

4.4 Hilfswerkzeuge zur Software-Entwicklung

In diesem Abschnitt sollen weitere Werkzeuge vorgestellt werden, welche man zwar nicht zwingend dazu benötigt ausführbare Programme für Windows CE zu erstellen, jedoch um den in 2.1 vorgestellten Entwicklungszyklus effizient umsetzen zu können.

4.4.1 Der GNU Debugger

Der GNU Debugger(GDB) wurde zugehörig zur GNU Compiler Collection Mitte der achtziger Jahre als Teil des GNU Systems vorwiegend von Richard Stallman geschrieben. Der

GDB dient dabei dazu den Ablauf eines Programms nachverfolgen zu können und gegebenenfalls zu modifizieren, um so Fehler zu finden und zu beseitigen.

Bei der Software-Entwicklung für Windows CE ist es besonders interessant die Möglichkeit des GDB zu nutzen, von einem entfernten Desktopsystem Programme auf ein vollkommen unabhängiges Zielsystem zu debuggen. Zu diesem Zweck existiert das Hilfsprogramm gdbserver, welches für das Zielssystem erstellt und dort ausgeführt wird, und eine eingehende Verbindung eines anderen Rechners erwartet. Über diese Verbindung kann dann der Ablauf des Programms, welches der gdbserver auf dem Zielsystem geladen hat, gesteuert und mit dem GDB so “debugt” werden, als würde die Ausführung auf dessen System ablaufen. Eine detailiertere Einführung in den GDB und seine Funktionen würde den Rahmen dieser Arbeit sprengen. Bemerkenswert ist, dass die offizielle Codebasis des GDB den vollen Funktionsumfang auch auf Windows CE unterstützt. Um den Debugger verwenden zu können reicht es, die Befehlsfolge:

```
./configure --target=arm-mingw32ce --prefix=/opt/mingw32ce && make &&
    make install
```

im Quellordner des GDB auszuführen, um diesen für das System, auf welchem man sich gerade befindet, in einer Art zu erstellen, dass er dazu verwendet werden kann Zielprogramme, welche für arm-mingw32ce erstellt wurden, zu verarbeiten.

Den Server, welcher auf dem Windows CE System ausgeführt werden muss, erzeugt man dann mit den Werkzeugen der CEGCC Initiative durch das Kommando:

```
gdbserver/configure --host=arm-mingw32ce --prefix=/opt/mingw32ce &&
make && strip gdbserver.exe && make install
```

4.4.2 Das GNU Buildsystem

Mit der Portierung der GCC Werkzeuge auf immer neue Plattformen wurde es notwendig, den Bauprozess einer Software weiter zu generalisieren und unabhängiger vom Entwicklungssystem zu machen. Die GNU Initiative entwickelte zu diesem Zweck das GNU Buildsystem. Diese Werkzeuge abstrahieren eine Buildumgebung auf die Features und Funktionen, welche zur Verfügung stehen. Dadurch wird es möglich selbst in vollkommen unbe-

kannten Buildumgebungen (wie beispielsweise einer GNU/Linux Umgebung mit Headern und Bibliotheken für Windows CE) verwertbare Buildkonfigurationen zu erstellen, so dass eine Software, die portabel geschrieben ist, durch Präprozessormakros entsprechend der zur Verfügung stehenden Funktionalitäten in der Buildumgebung konfiguriert und kompiliert werden kann. Das GNU Buildsystem stellt die Freie Software Alternative zu den in 3.2.1 Visual Studio Projektdateien dar, welche es innerhalb von Visual Studio ermöglichen soll, unterschiedliche Plattformkonfigurationen zu verwalten. Im Gegensatz zu Visual Studio, das klar definierte Umgebungen benötigt (z.B.: Visual Studio 2008 mit Windows Mobile 6.5 SDK), macht das GNU Buildsystem keinerlei Annahmen bezüglich der Softwareversion oder den vorhandenen Bibliotheken.

Zur Anwendung des GNU Buildsytems muss ein Entwickler die zwei Dateien configure.ac und Makefile.am erstellen[3]. Configure.ac sollte dabei Informationen darüber enthalten, welche Funktionalitäten, etwa das Vorhandensein einer Bibliothek, überprüft werden sollen, während Makefile.am als Eingabedatei für Automake eine Vorlage für das Makefile der Software definiert. Aus configure.ac kann anschließend ein Konfigurationsscript (configure) generiert werden, welches das System auf die in configure.ac definierten Features hin untersucht und aus der von Automake erstellten Vorlage ein systemspezifisches Makefile generiert. Wird zusätzlich Autoheader verwendet, kann außerdem ein Konfigurationsheader erstellt werden, welcher Präprozessormakros oder Alternativfunktionen definiert, die im Code einer Anwendung verwendet werden können (zum Beispiel HAVE_PTHREAD_H). Anzumerken ist, dass für die Verwendung mit Windows CE keine speziellen Programme des Buildsystems selbst notwendig sind, da diese nur auf dem System ausgeführt werden, wo der Code übersetzt wird. [Gar00]

4.4.3 OpenSSH

Wie bereits in Abschnitt 3.2.2 erläutert, verwendet Visual Studio zur Kommunikation eines Entwicklersystems mit einem Windows CE Gerät das proprietäre “ActiveSync” Protokoll, das es in Verbindung mit der Remote API (RAPI) erlaubt, Dateien auf ein Gerät zu übertragen, zu starten und das System zu manipulieren. [Mici] Eine Freie Software, die diese Funktionen zur Datenübertragung und Kommunikation mit Konsolenanwendungen zur Ver-

[3] Die Erstellung von configure.ac kann dabei mit dem Werkzeug Autoscan weitestgehend automatisiert werden

fügung stellt, ist CESSH[4]. [Micg] Hierbei handelt es sich um einen Port des weit verbreiteten OpenSSH4.4.3 Servers für Windows CE. Diesen von Microsoft Entwicklern für den in 2.3.1 bereits erwähnten “Plattform-Builder” entwickelten Port kann man mit ein wenig Konfigurationsaufwand auch auf Windows Mobile Systemen verwenden. Dadurch, dass OpenSSH eine eigene Shell implementiert, kann es auch (wie der Visual Studio Debugger) dazu verwendet werden, Konsolenein- und Ausgaben auf einem entfernten Gerät zu überwachen und zu tätigen. Insbesondere mangels einer funktionstüchtigen Konsolenapplikation für Windows CE ist diese Fernsteuerung des Systems eine wichtige Funktionalität, die man auch als Entwickler mit Freier Software benötigt. Im Unterschied zu Visual Studio wird dafür nicht einmal eine lokale Verbindung (USB/Bluetooth) gebraucht; jede TCP Verbindung ist ausreichend. Ein Installationsarchiv, welches die Konfiguration und Einrichtung des Entwicklungssystems mit freien Werkzeugen unterstützen soll, wurde im Rahmen dieser Arbeit erstellt und befindet sich auf der beigelegten CD.

4.4.4 Paketierung mit Freier Software

Um das in Abschnitt 3.2.3 beschriebene Verfahren der Softwarepaketierung für Windows CE mit freien Werkzeugen umsetzen zu können, ist es nötig die _setup.xml, welche die Informationen enthält, durch die ein Cabinet-Archiv als Cabinet-Installer verwendet werden kann, zu generieren. Zu diesem Zweck gibt es das Freie Software Werkzeug pocketpc-cab. Dieses ist als Debian Paket für aktuelle Debian Distributionen zwar noch verfügbar, wird allerdings nicht mehr aktiv weiterentwickelt und unterstützt nur Windows CE Versionen bis einschließlich Version 4. Auf der beigelegten CD befindet sich eine modifizierte Version dieses Skripts, welches die Erstellung von Paketen auch für die aktuellsten Windows CE Versionen ermöglicht und darüber hinaus noch weitere Features (etwa das Anlegen von Registry Einträgen) enthält

Um Cabinet-Archive zu erstellen, verwendet pocketpc-cab die Software LCAB [Rie], welche es ermöglicht, die Konfigurationsdatei zusammen mit den Daten, die man auf dem Gerät installieren möchte, in ein Archiv zu verpacken.

[4]CESSH steht unter der Microsoft Limited Permissive License, die eigentlich proprietäre Einschränkungen vorsieht, welche aber bei CESSH herausgenommen wurden.

4.4.4.1 Signieren von Paketen mit Freier Software

Auch mit Freien Software Werkzeugen muss man auf die Möglichkeit seine Pakete zu signieren nicht verzichten. Wie bereits in Abschnitt 3.2.4 eingeführt, benötigt man ein X.509 Zertifikat, welches von einer vertrauenswürdigen Zertifizierungsstelle ausgestellt werden muss. X.509 ist dabei ein gängiger Standard und auch die Basis der S/MIME E-Mail Verschlüsselung. Zum Erlangen eines solchen Zertifikats kann man sich entweder an eine der von Windows CE als vertrauenswürdig angesehenen Zertifizierungsstellen wenden oder auch mit etablierten freien Werkzeugen, wie etwa TinyCA [Tin], sich selbst ein Zertifikat erstellen und dies auf einem Windows System Gerät installieren.

Obwohl die Code Signatur mit signtool die proprietäre Microsoft Crypto API verwendet, gibt es eine Freie Software Lösung, die die gleiche Funktionalität auf GNU/Linux Systemen bietet. Da die Signatur auf offenen Standards wie etwa PKCS #7 basiert, konnte dazu die OpenSSL Bibliothek für die kryptographischen Berechnungen verwendet werden. Das daraus resultierende Werkzeug, das insbesondere die Einbettung der Signatur in ausführbare Daten so unterstützt, dass Windows diese auch anerkennt, nennt sich osslsigncode. Es kann man in der gleichen Weise verwenden wie signtool. Um etwa einen CAB-Installer zu signieren reicht es, wenn man Folgendes ausführt:

```
osslsigncode -spc meinzertifikat.spc -key meinprivaterschluessel.pvk \
        -n "Mein Anwendungsname" -i http://www.meinewebseite.com/ \
        -in meininstaller.cab -out meinsignierterinstaller.cab
```

4.5 Einrichten einer CEGCC Entwicklungsumgebung

Als kleines Projekt, welches sich vorwiegend an Software Entwickler richtet und infolge des allgemein geringen Interesses an der Freien Software-Entwicklung für Windows CE, hat es die CEGCC Initiative bisher nicht geschafft in eine GNU/Linux Distribution aufgenommen zu werden. Auch lassen sich die Entwickler des Projektes für neue “Release” Versionen einige Zeit. So existiert beispielsweise in der zum jetzigen Zeitpunkt (05.04.2010) als Paket veröffentlichten Version 0.59 noch ein kritischer Fehler im Relocator, welcher das Laden von

Bibliotheken in Windows CE Versionen ab 5.0 verhindert. Der Fehler wurde zwar bereits im Januar behoben, aber da seitdem keine neuen Pakete veröffentlicht wurden, bleibt nur die Option, die instabilen Entwicklerversionen aus dem Quellcodeverwaltungssystem zu verwenden. Dem Code liegen jedoch bash Skripte bei, welche die Erstellung der Werkzeugkette erleichtern. Sind alle Abhängigkeiten (insbesondere bison, flex und gmp) erfüllt, reichen die einfachen Kommandos:

```
svn co https://cegcc.svn.sourceforge.net/svnroot/cegcc cegcc
mkdir cegcc-build
cd cegcc-build
sudo ../cegcc/src/scripts/build-mingw32ce.sh
```

die ausreichend sind, um die CEGCC Werkzeuge im Verzeichnis /opt/mingw32ce zu installieren. Dabei ist es auch möglich mit dem –prefix Argument das Installationsverzeichnis zu ändern und auf die privilegierte Ausführung mittels sudo zu verzichten.

4.6 Beispielhafter Entwicklungsablauf mit Freie Software Werkzeugen

In dieser Sektion wird beispielhaft aufgezeigt, wie man eine “Hallo Welt” Applikation unter Verwendung der WinAPI für Windows CE mit freien Werkzeugen erstellen, übersetzen, konfigurieren, testen und debuggen kann.

4.6.1 Entwicklung

Auf Freie Software Editoren zur Entwicklung soll an dieser Stelle nicht weiter eingegangen werden, da sie nicht speziell für Windows CE vorhanden sind, und auch wenn sie als Beispiele für hochqualitative und hervorragende Freie Software dienen könnten, würde eine genauere Betrachtung unterschiedlicher Editoren den Rahmen dieser Arbeit sprengen. Als direkte Alternative zur Visual Studio Entwicklungsumgebung mit integriertem Debugger könnte man allenfalls den GNU Emacs als Editor des GNU Betriebssystems erwähnen, welcher beispielsweise eine Integration mit dem GNU Debugger bietet.

Als simpelstes Beispiel einer WinAPI Anwendung, welche ein Fenster zeichnet, soll das folgende Programm dienen:

```
#include <windows.h>
int WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR
    lpCmdLine, int nCmdShow)
{
    MessageBox(0, L"Hallo Windows CE!", L"Dieses Programm sagt:", MB_OK)
        ;
}
```

Dieses Programm soll ein Nachrichtenfenster öffnen, welches einen OK Knopf anbietet, um es wieder zu schließen. Bemerkenswert ist die Verwendung von wchar_t und dem L Makro zur Umwandlung von Zeichenketten in das Unicode Format. In ihrer ASCII Variante wären diese Funktionen in der Windows CE API nicht enthalten.

4.6.2 Konfiguration

In diesem Abschnitt soll anhand des Beispiels erläutert werden, wie man das GNU Buildsystem zur Konfiguration eines Softwarepaketes verwenden kann.

4.6.2.1 Basiskonfiguration für Windows CE

Um das GNU Buildsystem zu verwenden, lässt sich beginnend mit dem Programm Autoscan ein Gerüst für die Konfigurationsdatei erstellen. Dieses Gerüst wird unter dem Namen configure.scan in dem Ordner, in welchem man Autoscan ausführt, erstellt und muss anschließend weiter editiert werden, um etwa das Auswählen unterschiedlicher Werkzeugketten möglich zu machen. Um diese Funktion bei der Konfiguration zu ermöglichen, ist es ausreichend die Zeile:

```
AC_CANONICAL_HOST
```

in die configure.ac einzufügen. AC_CANONICAL_HOST ermöglicht es dann, dass man die zum Kompilieren zu verwendende Werkzeugkette etwa mit –host=arm-mingw32ce angeben kann, um für Windows CE zu kompilieren, und macht es weiterhin möglich, ohne Änderungen an den Konfigurationsscripten mit –host=i586-mingw-msvc andere Werkzeuge, etwa für

NT Systeme, auszuwählen. Editiert man nun noch das Makro AC_INIT in einer Form, dass es die korrekte Version und den Namen des Programms beschreibt, ist man bereits in der Lage Autoconf zu verwenden, um sich ein portables configure Script erstellen zu lassen.
Da in diesem Beispiel das GNU Buildsystem demonstriert werden soll, muss nun noch die Verwendung von Automake konfiguriert werden. Dazu erstellt man eine Makefile.am, welche in diesem simplen Fall nur die zwei Zeilen

```
bin_PROGRAMS = hello_world
hello_world_SOURCES = hello_world.cpp
```

enthalten muss. Um autoconf mitzuteilen, dass automake auch verwendet werden soll, dienen die zwei Zeilen:

```
AM_INIT_AUTOMAKE([foreign])
AC_CONFIG_FILES([Makefile])
```

Nun kann man Automake mit dem Parameter –add-missing aufrufen, um weitere Dateien zu generieren, etwa eine Installationsanleitung, Readme und Changelog. Mit einem weiteren Aufruf von Automake wird ein Makefile erzeugt, welches alle Ziele enthält, die man von GNU Software erwartet; das Archivieren des Quellcodes mit make dist, das Installieren mit make install oder das Deinstallieren mit make uninstall.

4.6.2.2 Portable Konfiguration

Mit der soeben vorgestellten Basiskonfiguration ist es bereits möglich, Software sowohl für Windows NT als auch für Windows CE Systeme auf einem GNU System zu erstellen. Dadurch, dass die Konfiguration auf jedem System dynamisch neu erstellt wird, ist diese schon deutlich robuster und besser zur Distribution geeignet als etwa Visual Studio Projektdateien. Ein weiterer Vorteil gegenüber den von Microsoft bereitgestellten Werkzeugen wurde dabei jedoch noch nicht beachtet, nämlich die Möglichkeit portablen Code zu konfigurieren. Durch die hohe Verbreitung der GCC auf verschiedenen Plattformen kann man auch das GNU Buildsystem für viele Plattformen verwenden und so Software entwickeln, die sich auch für exotische Systeme übersetzen lässt. Integriert man etwa:

```
```

```
AC_CHECK_HEADER(windows.h,[AC_DEFINE([HAVE_WINDOWS_H],[1],["windows.h
    found"])])
AC_CHECK_HEADER(stdio.h,[AC_DEFINE([HAVE_STDIO_H],[1],["stdio.h found"])
    ])
if test ${host} = "arm-mingw32ce"; then
     AC_DEFINE(HAVE_W32CE_SYSTEM,1,[Defined for  WindowsCE])
fi
```

hat man die Möglichkeit Autoheader zu verwenden, um eine Vorlage einer config.h zu erstellen, die beim Aufruf von configure mit Werten gefüllt wird, welche Definitionen festlegen, die bei der Kompilierung dazu verwendet werden können, um bestimmten Code nur unter bestimmten Bedingungen zu übersetzen.

Ein portables Programm, welches den bekannten Hallo Welt Code ausführt, könnte etwa folgendermaßen aussehen:

```
#include <config.h>
#ifdef HAVE_WINDOWS_H
    #include <windows.h>
    #ifdef HAVE_W32CE_SYSTEM
        #define MAIN int WinMain(HINSTANCE hI, HINSTANCE hPI, LPWSTR
            CmdLine, int CS)
        #define SAYHELLO MessageBox(0, L"Windows CE!", L"Hallo:", MB_OK);
    #else
        #define MAIN int WinMain(HINSTANCE hI, HINSTANCE hPI, LPSTR
            CmdLine, int CS)
        #define SAYHELLO MessageBox(0, "Windows NT!", "Hallo:", MB_OK);
    #endif // HAVE_W32CE_SYSTEM
#else
    #ifdef HAVE_STDIO_H
        #include <stdio.h>
        #define MAIN int main()
        #define SAYHELLO printf("Hallo Welt!\n");
    #else
        #error "I do not know how to say Hello"
    #endif // HAVE_STDIO_H
#endif // HAVE_WINDOWS_H
MAIN
{
```

```
    SAYHELLO;
}
```

Auch wenn dieses Programm zu Demonstrationszwecken exzessiv vom Präprozessor Gebrauch macht, veranschaulicht es dennoch, wie die vom configure Skript erzeugten Informationen in einer Anwendung verwendet werden können, um dieses für verschiedene Systeme zu übersetzen. In der Praxis wird dies häufig dazu verwendet, um Funktionen plattformabhängig zu verwenden, etwa dass man MKDIR an einer Stelle definiert und dann, je nach Plattform, für die man kompiliert, eine entsprechende Implementierung definiert.

4.6.3 Kompilierung

Zur Kompilierung mit den für Windows CE angepassten GNU Werkzeugen ist sicherzustellen, dass diese vom System auffindbar sind und wiederum auch die von ihnen verwendete Software (wie etwa Header-Dateien) finden. Dazu müssen die entsprechenden Umgebungsvariablen, in denen bestimmte Suchpfade definiert werden können, gesetzt werden. So ist zu gewährleisten, dass die Variablen INCLUDE, PATH und LIB auf die Pfade der Bauumgebung verweisen, welche man verwenden möchte. Wenn die Bauumgebung korrekt eingerichtet ist, kann man die Werkzeuge in der Art verwenden, wie es auch für andere Plattformen, beziehungsweise für natives Übersetzen einer Anwendung üblich ist. Verwendet man bei der Konfiguration ”automake –add-missing” sind in der Datei INSTALL Installationsanweisungen enthalten. Entsprechend können weitere unterstützte Optionen mit dem Aufruf von configure –help abgefragt werden. Ist alles korrekt konfiguriert und portiert, sollte durch den Aufruf von

```
./configure --host=arm-mingw32ce \
--prefix=/tmp/arm-mingw32ce && make && make install
```

eine Software für Windows CE unter Verwendung der MinGWce Header erstellt werden können. Binärdaten sowie eventuelle weitere Header-Dateien werden in dieser Konfiguration in das Verzeichnis /tmp/arm-mingw32ce installiert und können, sofern die INCLUDE und LIB Pfade entsprechend gesetzt sind, die Bauumgebung für davon abhängige Software erweitern.

4.6.4 Bereitstellung, Paketierung

Zur direkten Bereitstellung nach dem Übersetzen kann man den in Abschnitt 4.4.3 beschriebenen SSH Server verwenden. Aufgrund der Möglichkeit mit diesem eine sichere Verbindung über TCP/IP mit dem Zielgerät aufzubauen ist so keine direkte physische Verbindung, wie es etwa das USB verwendende Active Sync voraussetzt, nötig. Mithilfe des in OpenSSH vorhandenen Secure File Ftransfer Protocol (SFTP) Server und des Secure Copy (SCP) Werkzeugs ist es möglich, Dateien unmittelbar über das Netzwerk auf ein Zielgerät zu übertragen.

Sollen Pakete der Software bereitgestellt werden, kann man dies mithilfe der in 4.4.4 eingeführten Werkzeuge pocketpc-cab und lcab. Im Beispiel würde die Befehlsfolge:

```
cat > hello.files <<EOF
      hello_world.exe %CE1%/Hallo "Hallo Welt!" %CE11%
      EOF
perl pocketpc-cab.pl -p "AHeinecke" -a "Hallo Welt" hallo.files hallo.cab
```

einen CAB-Installer mit Namen hallo.cab erzeugen, welcher einen Eintrag im Startmenü als Hallo Welt! erstellt und hello_world.exe nach /Programme/Hallo installiert. Um einen kompletten, beispielsweise von make install erzeugten Verzeichnisbaum, etwa das Installationsverzeichnis /tmp/arm-mingw32ce, zu paketieren, befindet sich auf der beigelegten CD das Python Programm geninputfiles.py, das diesen traversiert und eine für pocketpc-cab-taugliche Eingabedatei erzeugt.

Man besitzt also auch mit Freier Software Möglichkeiten zur Bereitstellung und Paketierung, welche im Gegensatz zu Visual Studio Projektdateien genereller gehalten werden können und nicht spezifisch auf Projektbasis angepasst werden müssen.

4.6.5 Testen, Debuggen

Vorausgesetzt, eine SSH Verbindung mit dem Endgerät ist aufgebaut und man hat einen entsprechenden gdbserver und die Anwendung GDB wie in Abschnitt 4.4.1 beschrieben für Windows CE erstellt, hat man die Möglichkeit, von einem entfernten System seine Software zu starten und zu debuggen. Über SSH kann man Anwendungen auf dem Zielgerät ausfüh-

ren und etwa Konsolenausgaben nachverfolgen, beziehungsweise Konsoleneingaben tätigen. Man benötigt so weder ein physisch an den Computer angeschlossenes Zielgerät, noch muss man zeitaufwändig das Gerät manuell bedienen.

Auch den gdbserver kann man so etwa über SSH ausführen, um dann mit dem GNU Debugger auf diesen zu verbinden und den Programmablauf auf dem Gerät zu kontrollieren.

Dabei benötigt man auf dem Gerät nicht einmal Debug Symbole; diese werden nur auf dem System benötigt, auf welchem der GDB ausgeführt wird. Gerade bei größeren Anwendungen, die mit ihren Debug Symbolen das Speicherlimit von Windows CE 5 überschreiten würden, ist dies wichtig. Mit den Befehlen:

```
arm-mingw32ce-strip hello_world.exe
scp hello_world.exe smartphone:/
gdbserver :9999 /hello_world.exe
```

würde man auf einen (in der ssh Konfiguration als Smartphone eingetragenen) Host das Programm hello_world.exe kopieren und im gdbserver starten. Dieser wartet an Port 9999 auf eine eingehende TCP Verbindung eines GDB. Startet man nun auf dem Hostsystem den GDB mit:

```
arm-mingw32ce-gdb hello_world.exe
```

und befiehlt diesem, sich mit dem Gerät zu verbinden,

```
gdb> target remote 10.42.7.112
```

kann man mit dem Debuggen, wie man es von GDB kennt, beginnen.

5 Software-Portierung mit freien Werkzeugen

Dieses Kapitel soll eine Einführung darüber geben, wie mithilfe von Freier Software die Portierbarkeit eines Programms erhöht werden kann, insbesondere in Bezug auf Windows CE. Dazu werden mit WCELIBCEX und WCECOMPAT zwei unterschiedliche Bibliotheken vorgestellt, die die Portierung von Windows NT-basierten Anwendungen auf Windows CE erleichtern sollen. Anschließend wird beschrieben, wie man das Qt Framework als eine umfassende Bibliothek zur Plattformabstraktion mithilfe freier Werkzeuge erstellen kann, um so Qt Anwendungen für Windows CE erstellen zu können.

5.1 Kompatibilitätsbibliotheken

Durch die in Abschnitt 2.4.1 erläuterte Limitation des Funktionsumfangs der WinAPI für Windows CE entstehen bei der Portierung von Anwendungen, unabhängig von den verwendeten Werkzeugen, Probleme. Arbeitet man mit den freien Werkzeugen des CEGCC Projektes, ist dies nur ein geringeres Problem, da, wie bereits in Abschnitt 4.2.2 erwähnt, die Bibliothek libmingwex viele dieser Funktionen bereitstellt. Es gibt allerdings noch weitere Werkzeuge, welche insbesondere zur Erweiterung der von Microsoft bereitgestellten Bibliotheken konzipiert wurden. Diese implementieren weitaus weniger, aber dafür teilweise andere Funktionen als die libmingwex und sollen von daher, auch wenn man sie bei der Arbeit mit komplett freien Werkzeugen nicht unbedingt benötigt, als Freie Software-Hilfsmittel für die Windows CE Entwicklung an dieser Stelle kurz erwähnt und eingeführt werden.

5.1.1 WCECOMPAT

WCECOMPAT wurde von dem Software-Unternehmen Essmer entwickelt. Auf ihrer Homepage [Ess] wirbt die Firma:

> *WCECOMPAT plugs the holes in the eMbedded Visual C++ C Runtime Library, making it more compatible with ANSI C.*

Mit dem Ziel die Löcher in der WinAPI zu stopfen werden also für eine Anwendung, die WCECOMPAT einbindet, Funktionen unter dem gleichen Namen zur Verfügung gestellt, wie sie von der WinAPI für NT Systeme bekannt sind. Dadurch eignet sich WCECOMPAT besonders da, wo man keinerlei Änderungen am Code einer Software vornehmen möchte, um diese für Windows CE zu kompilieren. Alles, was man als Entwickler machen muss, ist dem Compiler mitzuteilen, dass die Header-Dateien dieser Bibliothek als erste durchsucht werden sollen. WCECOMPAT ist unter anderem eine Abhängigkeit von OpenSSL für Windows CE. Neben klassischen Funktionen, die unter Windows CE fehlen, implementiert WCECOMPAT außerdem grundlegende Funktionalitäten für das Umleiten von Aus- und Eingaben, so genannte Pipes.

5.1.2 WCELIBCEX

Die Bibliothek WCELIBCEX ist mit nur 61 Funktionen deutlich kleiner als WCECOMPAT, wird aber dennoch in einigen Projekten verwendet und bietet durch eine Implementierung von Verzeichnis- (getcwd, mkdir,..), Umgebungs- (setenv, getenv,..) und weiteren häufig verwendeten Funktionen (errno) eine Basis der wichtigsten Funktionen, welche bei der Portierung zu Windows CE fehlen. Als Prefix wird bei Funktionen überwiegend wceex_ verwendet, um so Namenskonflikte zu verringern. Da man die Funktionsnamen jedoch zumeist umdefiniert, um Änderungen an den Quelltexten einer zu portierenden Anwendung gering zu halten, hat dies keinen Effekt bei der Symbolauflösung des Linkers. Zur Implementierung von Umgebungsvariablen wird in dieser Bibliothek die Windows Registry verwendet. Unter “HKLM/Software/WCELIBCEX/Environment” werden Umgebungsvariablen permanent abgelegt. Entwickelt wurde die Bibliothek zur Portierung von Software vom Unternehmen Taxus SI Ltd., welches sie 2005 als Freie Software unter einer nicht schützenden Lizenz

veröffentlichte.

5.1.3 Probleme mit Kompatibilitätsbibliotheken

Kompatibilitätsbibliotheken haben das konzeptionelle Problem, dass sie Funktionen übernehmen, die eigentlich vom System verwaltet werden sollten. Dies kann zu Konflikten führen, wenn man Funktionen kombiniert, die gegen unterschiedliche Bibliotheken verknüpft wurden, etwa wenn ein Befehl die Variable errno aus einer der Bibliotheken setzt, aber die Auswertung, beziehungsweise Überprüfung des Fehlers auf einer anderen Variablen erfolgt. Das Verwenden von unterschiedlichen Kompatibilitätsbibliotheken ist aufgrund von Überschneidungen der Symbolnamen auch nicht möglich, selbst wenn dies teilweise wünschenswert wäre. Zudem wird durch die einfache Ersetzung von Systemfunktionen durch eigene Implementierungen nicht immer exakt das gleiche Verhalten dieser erreicht werden. So kann man leicht Fehler einführen, welche zwar das Kompilieren und Linken einer Software nicht behindern, aber bei der Ausführung auftreten.

Mittelfristig kann es daher sinnvoller sein, Software für Windows CE aufwändiger, aber kontrollierter, nur unter Verwendung der WinAPI Funktionen zu portieren.

5.1.3.1 Newlib und Windows CE

Ist es mithilfe der Newlib möglich, mit vergleichsweise geringem Aufwand POSIX-konforme Software für Windows CE zu portieren, gibt es allerdings auch dabei die bereits im vorherigen Abschnitt erwähnten Probleme. Darüberhinaus verwendet Newlib nicht die C-Aufrufe aus der WinAPI[1] (enthalten in Coredll), so dass es nicht möglich ist WinAPI Aufrufe beim Linken der Software verknüpfen zu lassen. Man kann nicht einmal die Funktionen LoadLibrary und GetProcAddress verwenden, um explizit nur bestimmte Funktionen zur Laufzeit aufzulösen. Nutzt man also Newlib muss man komplett auf die WinAPI verzichten (etwa um Fenster anzuzeigen). Dies ist bei einigen kleinen Dienstprogrammen, welche im Hintergrund ihre Arbeit verrichten, vielleicht noch möglich, aber für die professionelle Software-Entwicklung für Windows CE untauglich. Und selbst wenn man Funktionen aus Bibliothe-

[1] Wie bereits erwähnt, sind Windows API Funktionen von CE nicht wie bei Windows NT von der C-Laufzeitumgebung getrennt.

ken, die mit Newlib verbunden wurden, aus anderen Anwendungen heraus aufrufen möchte, können durch die Vermischung der Namensräume dabei Konflikte und Fehler entstehen, die im Endeffekt die Portierung aufwändiger machen können, als hätte wenn man von Beginn an ganz auf Newlib verzichtet.

5.2 Das Qt Framework

Das Qt Framework ist weniger eine Kompatibilitätsbibliothek als eine eigene Plattform. So bietet es eine eigene Ebene, auf der Software aufbauen kann, ohne von Standard C/C++ Funktionen Gebrauch machen zu müssen. Als umfangreiche Klassenbibliothek abstrahiert dabei Qt nahezu sämtliche Systemaufrufe, wodurch es möglich sein soll, Qt Anwendungen für alle Systeme, die unterstützt werden, ohne Änderungen zu erstellen.
Unter der LGPL lizenziert, stellt Qt die Basis für viele Freie Software-Initiativen (beispielsweise KDE) zur Verfügung und wird im großen Stil professionell angewandt. Vom Handyhersteller Nokia vertrieben, [2] wird insbesondere Wert auf die Möglichkeit gelegt, Desktop-Anwendungen mit vergleichsweise geringem Aufwand durch Verwendung von Qt auf Mobilfunkgeräte (etwa Windows Mobile Handys) portieren zu können. Von daher soll Qt an dieser Stelle sowohl als ein Beispiel einer komplexen Software für Windows CE dienen, an welcher man durch Anwendung der freien Werkzeuge eventuelle Schwächen oder Probleme aufdecken kann, als auch als Freie Software-Lösung zur Portierung von Software für Windows CE. Dadurch, dass Qt bereits für Windows CE portiert wurde, können dabei die Schwierigkeiten herausgearbeitet werden, welche entstehen, sollte man nicht den offiziell unterstützten Visual Studio Compiler verwenden, sondern freie Werkzeuge.

Wäre man in der Lage, eine für den professionellen Einsatz taugliche Version des Qt Frameworks mit den Werkzeugen der CEGCC Initiative zu erstellen, würde dies einen großen Fortschritt für die Software-Entwicklung mit freien Werkzeugen für Windows CE bedeuten, da Software, die das Qt Framework verwendet, um Plattformunabhängigkeit zu erreichen, damit auch für Windows CE erstellt werden könnte.

[2] Qt wurde von der norwegischen Firma Trolltech entwickelt, welche 2008 von Nokia aufgekauft wurde.

5.2.1 Erstellung von Qt mithilfe von CEGCC

Für Qt 4.6.0 stellte Maurice Kalinowski, ein Entwickler bei Nokia, bereis eine Version online, mit der es möglich war, mithilfe der CEGCC Bibliotheken zumindest eine rudimentäre Version von Qt zu erstellen. [Kal] Im Folgenden wird ein Weg beschrieben, die aktuelle Entwicklungsversion von Qt (576cb12 vom 28.4.10) mit CEGCC Werkzeugen für Windows Mobile 6.5 zu erstellen und dabei nach Möglichkeit den Quelltext von Qt unverändert zu erhalten. Der von Kalinowski erstellte Patch wurde dabei zwar zur Anschauung verwendet, aber nicht angewandt und die Lösungen unterscheiden sich stark, wie man erkennen kann, wenn man die im Anhang befindlichen Patches mit denen von [Kal] vergleicht.

5.2.1.1 Erstellung der Buildkonfiguration

Für jede Plattform, auf der man Qt erstellen möchte, benötigt man so genannte mkspecs; dies sind Konfigurationsdateien, die den Build kontrollieren. Basierend auf der Buildkonfiguration für MinGW kann man eine MinGWce qmake.conf erstellen. (siehe Anhang A.1 In dieser Datei werden vorwiegend Definitionen gesetzt, welche beim Kompilieren und Linken verwendet werden, sowie die passenden Werkzeuge für die arm-mingw32ce Plattform definiert. In den Linkeroptionen muss beispielsweise abgeändert werden, dass anstelle von kernel32.lib coredll.lib verwendet werden muss. Als weitere plattformspezifische Datei benötigt man die Header-Datei qplatformdefs.h . Diese wird zumindest indirekt in alle Codedateien eingebunden und dient dazu, Funktionen, die nicht in der Windows CE API enthalten sind, so zu definieren, dass diese dennoch gefunden und verwendet werden können. Dazu bringt Qt eine eigene Kompatibilitätsbibliothek mit, welche etwa Funktionen wie “stat” oder “open” implementiert. Auch wenn die Mingw32ce API diese Funktionen als Erweiterung der öffiziellenÄPI anbietet, sollte man diese Definitionen besser möglichst unverändert von den für die Microsoft Werkzeuge geschriebenen Spezifikationen übernehmen, da sie zwar eventuell ein anderes Verhalten zeigen, aber in der von den Qt Entwicklern verfassten Variante erprobt funktionieren. Der Einfachheit halber wurden in die Plattformdefinitionen zusätzlich einige Definitionen eingefügt, welche in den Headern der WinAPI für Windows CE enthalten seien sollten, aber noch fehlten. Zur Anschauung befindet sich diese Datei in Anhang A.2

5.2.2 Qt Werkzeuge auf der Hostplattform

Das Qt Framework verwendet eine Reihe von Werkzeugen, die das Qt-Buildsystem bilden. Um Qt zu erstellen, werden diese passend zur verwendeten Version/Plattform/Konfiguration benötigt, mit der Dateien für die Zielplattform erstellt werden sollen. Sie enthalten bei der Kompilierung festgelegte Konfigurationsoptionen, welche es nötigt machen, sie bei Änderungen an der Konfiguration jeweils neu zu erstellen. Die Werkzeuge dienen, wie bereits erwähnt, zur Erweiterung des Buildsystems und müssen entsprechend für das System übersetzt werden, auf dem der Build durchgeführt werden soll (im Beispiel Linux-g++).
Notwendig sind mindestens:

- **QMake** Ein Makefile Generator, um Qt Projekte zu konfigurieren.
- **MOC** Der Meta-Objekt Compiler erzeugt meta-object Code, welcher dazu notwendig ist, den C++ Sprachumfang, etwa um Signale, zu erweitern.
- **RCC** Ressource Compiler, benötigt, um Ressource-Dateien in Qt Applikationen einzubinden (etwa Bilder).
- **UIC** Benutzer Interface Compiler, erzeugt aus den vom Qt Designer erstellten XML Dateien C++ Dateien.

Das folgende Skript konfiguriert Qt mit der Konfiguration aus Anhang **??**, die als mkspec für wincewm65professional-g++ hinterlegt ist, und erstellt anschließend die Host-Werkzeuge für ein GNU/Linux System.

```
#!/bin/bash

./configure -platform linux-g++ -xplatform wincewm65professional-g++ -qt-
    zlib \
        -no-qt3support -nomake examples -nomake demos -nomake tools -no-
            largefile \
        -no-iconv -no-webkit -no-openssl -opensource -little-endian -arch
             windowsce \
        -static -D QT_NO_DIRECT3D -embedded --prefix=/opt/mingw32ce/arm-
            mingw32ce

QTCEDIR=$(pwd)"/"$(dirname "$0")
```

```
OLDPATH=$PATH
export PATH=QTCEDIR/bin;$PATH
hosttools="bootstrap moc uic rcc"

for f in $hosttools; do
    cd $QTCEDIR/src/tools/$f && \
    qmake -spec $QTCEDIR/mkspecs/linux-g++ && \
    make
    cd $QTCEDIR
done
export PATH=$OLDPATH
```

Aufgrund der Annahme des Konfigurationsskripts, dass bei der Konfiguration für Windows CE ein Windows NT System als Buildplattform verwendet wird, kann es vorkommen, dass das qmake an der falschen Stelle gesucht wird. In diesem Fall sollte man einen symbolischen Link zu qmake in den Pfad mit aufnehmen, welcher keine Pfadseparatoren enthält.

Sind die Host-Werkzeuge erzeugt, kann man nun daran gehen die eigentlichen Qt Bibliotheken zu erstellen.

5.2.3 Die Qt-Core Bibliothek

Das Qt Framework ist modular aufgebaut. Es gibt zwar wechselseitige Abhängigkeiten der einzelnen Module (so benötigen die meisten Qt-Core), man muss jedoch nicht den kompletten Umfang von Qt erstellen. Das Modul Qt-Core bildet dabei die Basis und enthält grundlegende Objekte wie QObject oder QString.

Beginnt man mit dem Übersetzen von Qt-Core mit der arm-mingw32ce Werkzeugkette der CEGGC Initiative, wird man früh einen Fehler erkennen, der dadurch entsteht, dass Qt zwei Header, altcecrt.h und ceconfig.h, nicht finden kann. Die Datei ceconfig.h ist ein Header, der den Umgang mit selbst erstellten SDKs ermöglichen soll, indem er Definitionen enthält, welche Module im Betriebssystem vorhanden sind. Er enthält keinerlei Copyright und wird mit den Betriebssystemen, etwa auch Windows Mobile, ausgeliefert. So kann man diese Header-Datei von einem Windows Mobile 6.5 Professional Gerät passend zu den mkspecs

kopieren und einbinden.

5.2.3.1 Exkurs altcecrt.h

Da es an dieser Stelle passend ist, noch einmal zu verdeutlichen, warum es schwer, beziehungsweise nahezu unmöglich ist, mit Freier Software die gleichen Header zur Verfügung zu stellen wie mit proprietären Werkzeugen, ohne dabei das Copyright zu Verletzen, wird hier kurz vom Thema des Kapitels abgewichen.

Ist es bei nahezu allen Header-Dateien, die für Windows CE bereitgestellt werden, möglich, durch eine Internetsuche Informationen darüber zu finden, welche Deklarationen oder Definitionen diese enthalten, findet man für altcecrt.h keinerlei hilfreiche Daten. Das einzige, was man erfährt, ist, dass diese in den “vorkompilierten Header” eingebunden wird, wenn man im Visual Studio ein Projekt erstellt, welches die Advanced Template Library (ATL) von Microsoft einbindet. Auch die offizielle Dokumentation (MSDN) bietet keine weiteren Informationen.

Ein Blick in den Header, welcher nicht in den SDKs enthalten ist, sondern bei Visual Studio mitgeliefert wird, offenbart, warum dies der Fall ist. Laut eines Kommentars zu Beginn enthält dieser Header Definitionen und Deklarationen für “time” Funktionen. Tatsächlich wird in ihm aber viel mehr definiert. Mit den Error-Definitionen, welche eigentlich in errno.h gehören, über Maximalgrößen aus limits.h, bis hin zu abgesicherten Funktionen, wie memcpy_s, herrscht in dieser Header-Datei ein vollständiges Durcheinander. Man kann sich bei Betrachtung von altcecrt.h nicht des Eindrucks erwehren, dass ein Entwickler von Microsoft, beauftragt damit, die ATL auf Windows CE zu portieren, die Informationen aus anderen Dateien zusammenkopiert hat, um so eine Art eigene Kompatibilitätsbibliothek zu erschaffen. Anstatt dabei die WinAPI für Windows CE um komplette Bereiche, wie etwa die sicheren Funktionen, zu erweitern, entschied man sich dem Anschein nach, nur die gerade benötigten Funktionen einzufügen und intern zu verwenden, was wiederum zu Namenskonflikten führt, wenn man eine “ordentliche Kompatibilitätsbibliothek” einbindet, die gleiche Bereiche vollständiger implementiert. (WCECOMPAT kann nicht eingebunden werden, wenn man altcecrt.h in seiner Software verwendet.)
Durch diese Arbeitsweise von Microsoft wird erreicht, dass diese Funktionen zwar unter

Windows CE in Bibliotheken enthalten sind, aber Entwickler davon keinen Gebrauch machen können, es sei denn, sie suchen explizit danach, wo denn die Funktionen, welche von den Laufzeitbibliotheken zur Verfügung gestellt werden, deklariert sind. Absichtlich oder unabsichtlich führt dies dazu, dass es für freie Initiativen nahezu unmöglich ist, derartige Dateien so zu reproduzieren, dass sie exakt die gleichen Informationen enthalten.

5.2.4 Erstellen von Qt-Core

Um nach diesem Exkurs mit der Kompilierung fortzufahren, kann man eine leere Datei mit Namen altcecrt.h, damit der Compiler mit dem Übersetzen fortfahren kann, und abwarten, welche Deklarationen fehlen, die der altcecrt Header geliefert hätte. Die Abwesenheit von Fehlerwerten aus errno.h führt als nächstes zu einem Abbruch der Kompilierung. Da es sich dabei um Standardwerte handelt, lassen sie sich einfach, etwa von der MinGW WinAPI von Windows NT, kopieren und in den Qplatformdefs Header einfügen. Man kann nun Fehler für Fehler weiter durch den Kompilierungsvorgang schreiten und diese beheben.

Dabei fällt auf, dass weniger die unterschiedlichen Compiler und Bibliotheken Probleme mit der Kreuz-Kompilerung von Qt von einem GNU/Linux Buildsystem zu einem Windows CE System erzeugen, sondern dass hauptsächlich Fehler dadurch entstehen, dass diese Art der Kompilierung bei Qt nie vorgesehen wurde. So sind zwar konditionale Definitionen für nahezu alle plattformspezifischen Funktionen vorhanden, allerdings wird bei der Überprüfung teilweise nicht daraufhin geprüft, ob man für Windows CE übersetzt, sondern nur, ob man den Visual Studio Compiler verwendet. Anweisungen sind schlicht nicht dazu ausgelegt mit den CEGCC Werkzeugen zu funktionieren. So finden sich etwa Überprüfungen der Form:

```
#if (defined(_MSC_VER) && (_MSC_VER > 600))
#if defined(_WIN32_WCE)
```

welche nur überprüfen, ob man für die Plattform Windows CE (WIN32_WCE) übersetzt, wenn der verwendete Compiler als Microsoft Compiler definiert ist.

Aufgrund solcher Stellen reicht es nicht passende qmake specs (Compiler Anweisungen) und Qplatformdefs zu schreiben, sondern man muss Code von Qt selbst bearbeiten. Glücklicherweise sind diese Stellen durch die saubere Architektur des Qt Frameworks nur selten

vorhanden. Zusammen mit den Konfigurationsdateien reichen die folgenden Änderungen, um Qt-Core sowohl als statisches Objekt, als auch als geteilte Bibliothek zu erstellen.

```
diff --git a/src/corelib/global/qglobal.cpp b/src/corelib/global/qglobal.
    cpp
index dfa2c17..f1d7542 100644
--- a/src/corelib/global/qglobal.cpp
+++ b/src/corelib/global/qglobal.cpp
@@ -2487,7 +2487,7 @@ void qFatal(const char *msg, ...)
 // makes use of the new secure getenv function.
 QByteArray qgetenv(const char *varName)
 {
-#if defined(_MSC_VER) && _MSC_VER >= 1400
+#if (defined(_MSC_VER) && _MSC_VER >= 1400) || (defined(Q_OS_WINCE) &&
    defined(Q_CC_MINGW))
     size_t requiredSize = 0;
     QByteArray buffer;
     getenv_s(&requiredSize, 0, 0, varName);
@@ -2506,7 +2506,7 @@ QByteArray qgetenv(const char *varName)

 bool qputenv(const char *varName, const QByteArray& value)
 {
-#if defined(_MSC_VER) && _MSC_VER >= 1400
+#if (defined(_MSC_VER) && _MSC_VER >= 1400) || (defined(Q_OS_WINCE) &&
    defined(Q_CC_MINGW))
     return _putenv_s(varName, value.constData()) == 0;
 #else
     QByteArray buffer(varName);
diff --git a/src/corelib/kernel/qfunctions_wince.h b/src/corelib/kernel/
    qfunctions_wince.h
index 35cda04..8d6f358 100644
--- a/src/corelib/kernel/qfunctions_wince.h
+++ b/src/corelib/kernel/qfunctions_wince.h
@@ -42,6 +42,7 @@
 #ifndef QFUNCTIONS_WCE_H
 #define QFUNCTIONS_WCE_H
 #ifdef Q_OS_WINCE
+#include <tchar.h>
 #include <stdio.h>
```

```
 #include <stdlib.h>
 #include <windows.h>
@@ -261,7 +262,9 @@ void *qt_wince_bsearch(const void *key,
 typedef unsigned long time_t;
 #define _TIME_T_DEFINED
 #endif
+#ifndef Q_CC_MINGW
 typedef HANDLE HDROP;
+#endif

 #ifndef WS_THICKFRAME
 #define WS_THICKFRAME    WS_DLGFRAME
```

Zwei weitere Probleme sind hier erkennbar; das eine ist die Definition des Typs HDROP in qfunctions_wince.h, erneut ein Punkt, in dem sich die Header von MinGW32ce und den Windows Mobile SDKs unterscheiden. HDROP ist einer der Windows Datentypen. Diese sind übersichtlich und immer mit dem zugehörigen Header, in welchem sie definiert sind, dokumentiert. [Mick]

Beispiel:

```
HDROP
Handle to an internal drop structure.
This type is declared in ShellApi.h as follows:

typedef HANDLE HDROP;
```

Entsprechend wurde HDROP in den CEGCC Headern auch definiert. Die ShellApi.h aus den Windows Mobile SDKs definiert diesen jedoch nicht, so dass Qt den Typ selbst definiert, was dann bei Verwendung der MinGW32ce API zu einem Namenskonflikt führt.
In der letzten Modifikation war das Einbinden des tchar.h Headers als erstes notwendig. Da die Reihenfolge, in welcher Include Anweisungen aus anderen Headern heraus (etwa windows.h) erfolgen, nicht dokumentiert ist, existieren auch hier Unterschiede, welche aber beispielsweise durch ein explizites Einbinden behoben werden können. Alle weiteren Übersetzungsprobleme konnten bereits durch die Qmakespecs und Qplatformdefs Dateien behoben werden.

Anmerkung: Beim Übersetzen von Qt-Core müssen auch noch einige Fehler in den Abhängigkeiten, etwa zlib, behoben werden. Da diese sich aber mit den soeben vorgestellten Fehlerklassen decken, werden sie hier nicht gesondert erwähnt.

5.2.5 Die Benutzeroberfläche Qt-GUI

Um minimale Qt Anwendungen mit einer grafischen Benutzeroberfläche für Windows CE erstellen zu können, benötigt man neben Qt-Core außerdem noch das Modul für grafische Benutzeroberflächen Qt-GUI. Wie bereits in Abschnitt 5.2.3 erwähnt, erwartet Qt nicht unter einem GNU/Linux System für Windows CE konfiguriert zu werden. Das Shell Skript, welches die Konfiguration erzeugt, nimmt daher an, dass man Qt für embedded Linux erstellt, und definiert Q_WS_QWS, also den “Qt Windows Server” als Fenstersystem anstelle des unter Windows CE eigentlich zu verwendenden Q_WS_WINCE (Was nichts anderes bedeutet, als dass die Windows API verwendet werden soll, um Fenster zu zeichnen). Eine Modifikation des Konfigurationsskripts ist also vonnöten:

```
diff --git a/configure b/configure
index 63d8f2d..dabf8d7 100755
--- a/configure
+++ b/configure
@@ -7356,9 +7356,12 @@ fi

 # Embedded compile time options
 if [ "$PLATFORM_QWS" = "yes" ]; then
-    # Add QWS to config.h
-    QCONFIG_FLAGS="$QCONFIG_FLAGS Q_WS_QWS"
-
+    # Add QWS to Config.h
+    if [ "$XPLATFORM" = "wincewm65professional-g++" ]; then
+        QCONFIG_FLAGS="$QCONFIG_FLAGS"
+    else
+        QCONFIG_FLAGS="$QCONFIG_FLAGS Q_WS_QWS"
+    fi
     # Add excluded decorations to $QCONFIG_FLAGS
     decors=`grep '^decorations *= ' "$QMAKE_VARS_FILE" | ${AWK} '{print
         $3}'`
```

```
    for decor in $decors; do
```

Diese fügt die Bedingung ein, dass für die erstellte Plattformkonfiguration nicht Q_WS_QWS definiert ist.

Ähnliche Probleme finden sich in den Projektdateien der einzelnen Qt Module, welche nur überprüfen, ob man für Windows CE kompiliert, wenn man die Konfiguration unter Windows aufruft. Diese Probleme zu beheben erfordert sehr viel manuelle Arbeit, da das Qt-GUI Modul selbst noch modular aufgebaut ist und man beinahe jede der 24 Projektdateien, welche Teile des Moduls sind, an mehreren Stellen editieren muss.

Bei der weiteren Übersetzung treten erneut Fehler auf. So definiert Qt beispielsweise an einer Stelle etwas neu, was schon vorher in den MinGWce Headern definiert wurde, oder es gibt bestimmte Präprozessor Bedingungen, die nicht für die CEGCC Werkzeuge gedacht sind. Da diese Art Fehler bereits in 5.2.3 ausführlich erläutert wurden, soll hier nicht weiter darauf eingegangen werden. Bei Interesse können in Anhang A.3 die kompletten Änderungen, welche nötig waren, um Qt-GUI zu kompilieren, nachgelesen werden. Hat man alle Fehler behoben und Qt-Gui erstellt, ist das nächste Ziel Applikationen zu erstellen, welche Qt verwenden, wobei man beim Schritt des Linkens feststellen wird, dass zur Erstellung ausführbarer Dateien nicht ausreichende Symboldefinitionen vorhanden sind. Aufgrund fehlender Referenzen auf Funktionen zur Verwaltung von Schriften gelingt das Linken der Anwendung nicht. Nicht vorhanden sind zum Beispiel Symbole für GetOutlineTextMetricsW, GetFontData sowie EnumFontFamiliesExW und GetIconInfo. Diese Funktionen werden in dem Header Wingdi definiert und sind auf Windows NT Systemen in der gdi32.dll enthalten. Windows CE besitzt gdi32.dll jedoch nicht, sondern enthält die Definitionen dieser Funktionen in der Bibliothek coredll.dll. So erklärt es sich, dass die Header (welche von Win32 übernommen werden konnten) die Funktionen deklarieren, diese aber nicht in den Exportdefinitionsdateien vorhanden sind.

5.2.5.1 Praktisches Erweitern der Exportdefinitionsdateien

Um trotzdem mit den fehlenden Symbolen arbeiten zu können, kann man das in Abschnitt **??** erläuterte Verfahren zur Erweiterung der Exportdefinitionen der MinGWce Bibliotheknutzen. Welche Funktion in welcher Datei enthalten ist, gibt die Microsoft Dokumentation

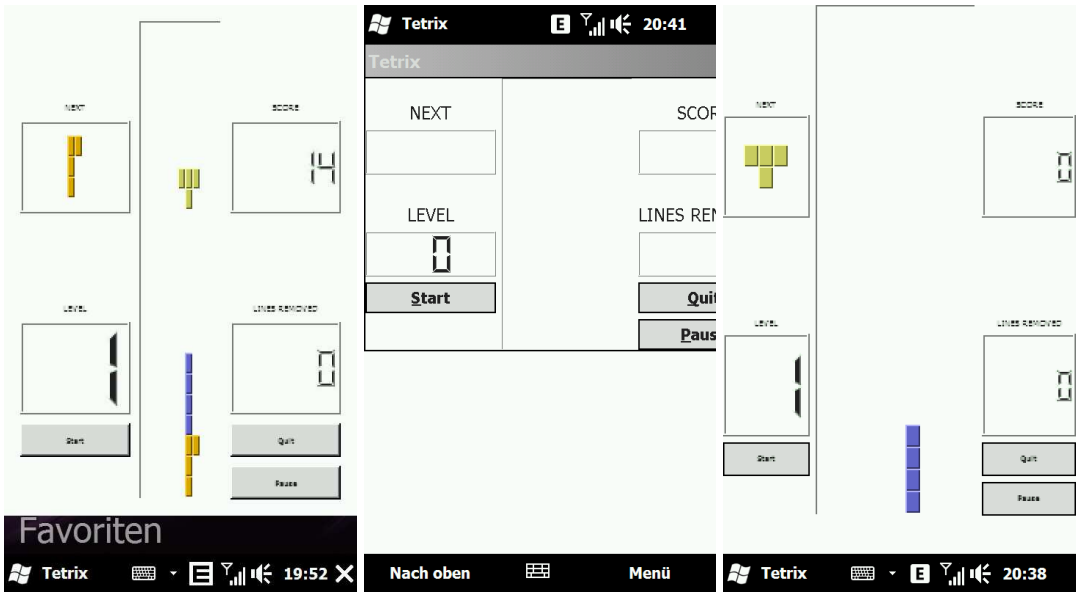
an; [Micd] verweist auf eine derartige Webseite, auf welcher beschrieben ist, dass die Funktion GetOutlineTextMetrics in coredll vorhanden ist. Da coredll jedoch in den MinGWce Bibliotheken nicht definiert ist, muss man den Symbolnamen noch in die passende Definitionsdatei (coredll.def) in den CEGCC Quellen eintragen. Ist dies geschehen, kann das dlltool diese in das passende Format umwandeln, welches dem Linker ermöglicht das Symbol zu referenzieren. Dies kann durch die Buildkonfiguration der MinGWce API automatisiert werden und es ist dazu ausreichend die folgenden Befehle auszuführen:

```
./configure --host=arm-mingw32ce --prefix=/opt/mingw32ce
make && make install
```

Anmerkung: Diese Funktionen wurden erst mit Version 5.0 von Windows Mobile implementiert, wodurch sich erklären lässt, warum gerade sie nicht definiert wurden.

5.2.6 Vergleich der Qt Varianten

Vergleicht man in Fig. 5.1 die Qt Anwendungen, welche mit CEGCC Werkzeugen erstellt wurden, mit denen, deren Erstellung mit Visual Studio durchgeführt wurde, erkennt man, dass eine unterschiedliche Integration in die Gerätedarstellung vorliegt. Mit der in dieser Arbeit beschriebenen Konfiguration gelingt die Integration etwas besser; so ist es etwa möglich, die Auflösung vom Hochformat ins Querformat zu ändern, die Systemkonfiguration der Schriften wird aber auch hierbei nicht korrekt verwendet. Für eine genauere Analyse, warum dies der Fall ist, wäre ein tieferes Verständnis des Qt Quellcodes erforderlich. Vermutlich liegen die Probleme darin, dass Qt trotz der Änderungen an einigen Stellen Code für die Embedded Linux Plattform anstelle der Windows CE Implementierung verwendet. Die Verwendung der Header und Werkzeuge der CEGCC Initiative sollte darauf keinen Einfluss haben, da unter MinGWce die gleichen Funktionen verknüpft werden, die auch Visual Studio verwenden würde. So waren keinerlei Stubs (Funktionen ohne Funktionalität) nötig und es wurden keine Teile des Codes bewusst für die Übersetzung mit freien Werkzeugen entfernt. Jedes Symbol, das nicht definiert war, konnte definiert und korrekt verbunden werden. Eine Ausführung von Qt Anwendungen auf Windows CE Systemen wäre ansonsten nicht möglich gewesen.



(a) Qt 4.6 mit CEGCC (b) Visual Studio (c) Qt 4.7 mit CEGCC

Abbildung 5.1: Vergleich der unterschiedlichen Qt Versionen

Was in den Bildern der Qt Anwendung Tetrix (Fig. 5.1) nicht zu erkennen ist, aber einen schwerwiegenden technischen Unterschied zwischen der Visual Studio Version und der mit der GCC erstellten Variante darstellt, ist, dass die GCC Version statisch verknüpft ist. So war es im Laufe dieser Arbeit nicht möglich, eine DLL-Datei von Qt-GUI zu erstellen, die dynamisch geladen werden kann und mehreren Programmen die GUI Funktionen von Qt zur Verfügung stellt. Zwar war es möglich Qt-Core als dynamische Bibliothek zu erstellen, bei Qt-GUI trat jedoch an mehreren Stellen GCC Bug 42871 auf. Dieser erzeugt unter bestimmten Bedingungen (welche bei statischem Linken nicht auftraten) einen internen Compilerfehler bei der Auflösung der Instruktionsreihenfolge für ARM Prozessoren und wird in CEGCC erst dann behoben werden, wenn eine neue Version des GCC als Basis verwendet wird. Da dies jedoch eine Anpassung der Patches der CEGCC Initiative bedeutet, ist dies ein aufwändiger und mühsamer Prozess.

6 Fazit und Ausblick

6.1 Fazit: Einsatzreife Freier Software Werkzeuge zur professionellen Entwicklung

Die in Kapitel 5 bei der Erstellung des Qt Frameworks aufgezeigten Probleme sind symptomatisch für den Stand der Entwicklung freier Werkzeuge, mit deren Hilfe technisch alle Anforderungen an eine Werkzeugkette zur Software-Entwicklung erfüllt werden können, teilweise sogar besser als mit den vom Hersteller vertriebenen. Da vorhandene Software für Windows CE jedoch überwiegend für und mit Visual Studio erstellt und konfiguriert wurde, fehlt zumeist die Unterstützung für freie Werkzeuge.

Wenn man sich für die Verwendung freier Werkzeuge zur Software-Entwicklung für Windows CE entscheidet, geht man leider immer noch, insbesondere bei den hoch komplexen Werkzeugen Compiler und Linker, ein nicht geringes Risiko ein, da diese aufgrund ihrer mangelnden Verbreitung nicht ausreichend Anwendung finden, um kritische Fehler aufzudecken. Wenngleich nahezu die komplette Codebasis für diese freien Werkzeuge von der GCC Initiative auf einem hohen Niveau qualitätsgesichert wird, sorgt eben die geringe Anzahl von Nutzern, die die GCC speziell zum Erstellen von ARM Anweisungen für Windows CE verwenden, dafür, dass es immer möglich ist, dass Fehler genau für diese Plattform auftreten.

Durch die Unterschiede zwischen der Windows API, welche basierend auf der MSDN Dokumentation bei MinGW zum Einsatz kommt, und der mit Visual Studio verwendeten, kann es vorkommen, das sich die Notwendigkeit ergibt, Fehler in der API eigenständig beheben zu müssen, wenn man speziell diese API benötigt. Schätzt man die Kosten für eventuelle Änderungen an der Windows API geringer als den Produktivitätsgewinn, den man durch die

Verwendung etablierter Verfahren und Werkzeuge des GNU Systems erzielt, sollte man freie Werkzeuge zur Software-Entwicklung für Windows CE verwenden.

6.2 Ausblick: Die Zukunft Freier Software auf Windows CE

Als Betriebssystem im Smartphone hat Windows CE 5, welches in der vorliegenden Arbeit vor allem thematisiert wurde, bald das Ende seines Lebenszyklus‘ erreicht. Windows CE wird dennoch nicht so bald obsolet werden, da das Betriebssystem neben Smartphones noch in vielen weiteren eingebetteten Systemen, etwa innerhalb von Automaten, zum Einsatz kommt. Auch wird es weiterhin Interesse daran geben, die vorhandenen Freien Software Werkzeuge für Windows CE weiter zu entwickeln. Da mit dem Einsatz sowohl die Nachfrage nach stabiler und erprobter Software für dieses System anhalten wird, als auch die Herausforderung Freier Software-Entwicklung für diese Plattform bestehen bleibt.

Für den Nachfolger, das auf Windows CE 6 basierende Windows Phone 7 genannte Betriebssystem, hat Microsoft bereits angekündigt, die Ausführung von nativer Software komplett verhindern zu wollen. Mit dem Kernel 6.0 und einer neuen Speicherarchitektur ermöglicht Microsoft zwar zukünftig, deutlich mehr und leistungsfähigere Software für diese Plattform zu erstellen, zwingt jedoch Entwickler dazu, ihre Software speziell für die .NET oder Silverlight Laufzeitumgebungen zu schreiben und so an Microsoft zu binden. Auf diese Weise wird zwar die Möglichkeit des Auftretens von Problemen bei der Speicher- und Prozessverwaltung vermindert, aber native Anwendungen, die von allen Systemressourcen oder der Echtzeitfähigkeit des Systems Gebrauch machen, werden grundsätzlich verhindert.

Unternehmen, wie etwa Nokia, welche unter enormem Aufwand ihre Software (Qt) für Windows CE portiert haben, werden von Microsofts neuer Plattform einfach ausgeschlossen. Als abschreckendes Beispiel illustriert dies, wie schnell man als Software Hersteller durch Verwendung proprietärer Software in Abhängigkeit eines anderen Unternehmens geraten kann.

7 Verzeichnisse

Literaturverzeichnis

[Bac] BACKX, DANNY: *Building Windows CE applications with the GNU toolset*. `http://wince-xcompile.sourceforge.net/build-process.html`. Zugriff: 18.05.2010.

[Bol] BOLING, DOUGLAS: *Windows CE .NET Advanced Memory Management*. `http://msdn.microsoft.com/en-us/library/ms836325.aspx`. Zugriff: 17.04.2010.

[Bol01] BOLING, DOUGLAS: *Programming Microsoft Windows CE*. Microsoft Press, 2001.

[Can] CANALYS: *Smartphone Market Share*. `http://www.canalys.com/pr/2010/r2010021.html`. Zugriff: 18.04.2010.

[CEG] CEGCC INITIATIVE: *The CEGCC Project*. `http://cegcc.sourceforge.net/`. Zugriff: 18.05.2010.

[Ess] ESSEMER: *WCECOMPAT Homepage*. `http://sites.google.com/a/essemer.com.au/www/windowsce`. Zugriff: 14.05.2010.

[Frea] FREE SOFTWARE FOUNDATION EUROPE: *Was ist Freie Software*. `http://fsfe.org/about/basics/freesoftware.de.html`. Zugriff: 13.04.2010.

[Freb] FREE SOFTWARE FOUNDATION EUROPE: *Wir sprechen von Freier Software*. `http://www.fsfe.org/documents/whyfs.de.html`. Zugriff: 13.04.2010.

[Frec] FREE SOFTWARE FOUNDATION, INC: *GCC Steering Committee*. `http://gcc.gnu.org/steering.html`. Zugriff: 14.05.2010.

[Fred] FREE SOFTWARE FOUNDATION, INC: *GNU Binutils*. `http://www.gnu.org/software/binutils/`. Zugriff: 11.05.2010.

[Free] FREE SOFTWARE FOUNDATION, INC: *GNU General Public License*. `http://www.gnu.org/copyleft/gpl.txt`. Zugriff: 25.04.2010.

[Fref] FREE SOFTWARE FOUNDATION, INC: *What is Free Software?* `http://www.gnu.org`. Zugriff: 14.04.2010.

[Freg] FREE SOFTWARE FOUNDATION, INC: *Why you shouldn't use the Lesser GPL for your next library*. `http://www.gnu.org/philosophy/why-not-lgpl.html`. Zugriff: 25.04.2010.

[Gar00] GARY V. VAUGHAN, BEN ELLISTON, TOM TROMEY AND IAN LANCE TAYLOR: *GNU Autoconf, Automake and Libtool*. New Riders publishing, 2000.

[Gat] GATES, BILL: *1989 Bill Gates Talk on Microsoft*. `http://csclub.uwaterloo.ca/media/1989%20Bill%20Gates%20Talk%20on%20Microsoft.html`. Zugriff: 13.05.2010.

[Gat02] GATLIFF, BILL: *Embedding with GNU: Newlib*. Embedded Systems Programming, Vol.15 No.1, 2002.

[Gou04] GOUGH, BRIAN J.: *An Introduction to GCC*. Network Theory, 2004.

[Gra02] GRASSMUCK, VOLKER: *Freie Software*. Bundeszentrale für politische Bildung, 2002.

[Gro02] GROVER, SANDEEP: *Linkers and Loaders*. Linux Journal, 2002.

[Kal] KALINOWSKI, MAURICE: *Qt-CEGCC*.
`http://qt.gitorious.org/+qt-developers/qt/ce-gcc/commits/4.6`.
Zugriff: 14.05.2010.

[Lan] LANGRIDGE, JASON: *Welcome to Windows Mobile 6*.
`http://blogs.msdn.com/jasonlan/attachment/1904298.ashx`.
Zugriff: 10.05.2010.

[Mica] MICROSOFT: *Authenticode*.
`http://msdn.microsoft.com/en-us/library/ms537359%28VS.85%29.aspx`.
Zugriff: 10.05.2010.

[Micb] MICROSOFT: *Funktionsdokumentation: CreateProcess*.
`http://msdn.microsoft.com/en-us/library/ms885182.aspx`.
Zugriff: 20.04.2010.

[Micc] MICROSOFT: *Funktionsdokumentation: IsProcessorFeaturePresent*.
`http://msdn.microsoft.com/en-us/library/ms886726.aspx`.
Zugriff: 20.04.2010.

[Micd] MICROSOFT: *Funktoion GetOutlineTextMetrics*.
`http://msdn.microsoft.com/en-us/library/ms933897.aspx`.
Zugriff: 10.05.2010.

[Mice] MICROSOFT: *Installieren von Stammzertifikaten in Windows Mobile*.
`http://support.microsoft.com/kb/915840`.
Zugriff: 10.05.2010.

[Micf] MICROSOFT: *Microsoft Developer Network*.
`http://msdn.microsoft.com`.
Zugriff: 10.05.2010.

[Micg] MICROSOFT: *Open SSH for Windows CE*.
http://www.codeplex.com/wikipage?projectname=CESSH.
Zugriff: 14.05.2010.

[Mich] MICROSOFT: *Overview of the Windows API*.
http://msdn.microsoft.com/en-us/library/Aa383723.
Zugriff: 20.04.2010.

[Mici] MICROSOFT: *Remote API(RAPI)*.
http://msdn.microsoft.com/en-us/library/aa920177.aspx.
Zugriff: 10.05.2010.

[Micj] MICROSOFT: *Security Features in the CRT*.
http://msdn.microsoft.com/en-us/library/8ef0s5kh.aspx.
Zugriff: 10.05.2010.

[Mick] MICROSOFT: *Windows Data Types*.
http://msdn.microsoft.com/en-us/library/aa383751%28VS.85%29.aspx.
Zugriff: 14.05.2010.

[Mina] MINGW.ORG: *MinGW | History*.
http://www.mingw.org/history.
Zugriff: 18.05.2010.

[Minb] MINGW.ORG: *MinGW | MinGW*.
http://www.mingw.org/node/19/revisions/571/view.
Zugriff: 18.05.2010.

[Red] RED HAT: *The Newlib Homepage*.
http://sourceware.org/newlib.
Zugriff: 18.04.2010.

[Rei04] REITER, BERNHARD: *Wandel der IT: Mehr als 20 Jahre Freie Software*. HMD, Heft 238, 2004.

[RG06] RISHAB GHOSH, PHILIPPE AIGRAIN: *The Impact of Free/Libre Open Source Software on Innovation and competitveness of the European Unition*.

http://flossimpact.eu/, 2006.
Zugriff: 18.04.2010.

[Rie] RIEN: *lcab: Cabinet File Creation Tool*.
http://ohnobinki.u.ohnopublishing.net/~ohnobinki/lcab.
Zugriff: 14.05.2010.

[RSA] RSA LABORATORIES: *PKCS 7 Cryptographic Message Syntax Standard*.
http://www.rsa.com/rsalabs/node.asp?id=2129.
Zugriff: 10.05.2010.

[Sta] STALLMAN, RICHARD: *Copyleft: Pragmatischer Idealismus*.
http://www.gnu.org/philosophy/pragmatic.de.html.
Zugriff: 13.04.2010.

[Sve] SVEN: *GCC flowchart*.
http://de.wikipedia.org/w/index.php?title=Datei:GCC_Schema.svg&filetimestamp=20060901231602.
Zugriff: 12.04.2010.

[Tin] TINYCA INITIATIVE: *TinyCA*.
http://tinyca.sm-zone.net.
Zugriff: 14.05.2010.

[Unb] UNBEKANNT: *Windows CE programming with GCC*.
http://win-ce.voxware.com:28575/Development%20Tools/gnuwince.html.
Zugriff: 18.05.2010.

[X C] X CONSORTIUM: *X11 Lizenz*.
http://www.xfree86.org/3.3.6/COPYRIGHT2.html#3.
Zugriff: 14.04.2010.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GNU GNU is not UNIX

UNIX Uniplexed Information and Computing System

GCC GNU Compiler Collection

GPL GNU General Public License

LGPL Lesser General Public License

API Application Programming Interface

WinAPI Windows Application Programming Interface

MinGW Minimalist GNU for Windows

SDK Software Development Kit

ROM Read Only Memory

IDE Integrated Development Envrionment

DLL Dynamic Link Library

PDA Personal Data Assistant

USB Universal Serial Bus

A Anhang

A.1 qmake.conf für die CEGCC Werkzeugkette

Listing A.1: QMake Konfigurationsdatei für die CEGCC Werkzeugkette

```
#
# qmake configuration for wince-g++
#
# Written for arm mingw32ce from the CEGCC Project
# http://cegcc.sourceforge.net
#

MAKEFILE_GENERATOR      = MINGW
TEMPLATE                = app
CONFIG                  += qt warn_on release link_prl copy_dir_files
   debug_and_release debug_and_release_target precompile_header win32
   wince
QT                      += core gui
# Defines for Windows CE
DEFINES                 += _NO_OLDNAMES _WIN32_WCE=0x502 \
_WIN32_IE=0x502 $$CE_ARCH _ARMV4I_ armv4i _ARM_ __arm__ Q_OS_WINCE_WM
   UNDER_CE\
 WINCE _WINDOWS _UNICODE UNICODE _WIN32 QT_NO_PRINTER QT_NO_PRINTERDIALOG
     #\
#QT_NO_TEXTCODEC
DEFINES += QT_NO_CONCURRENT
QMAKE_COMPILER_DEFINES  += __GNUC__ WIN32

QMAKE_EXT_OBJ           = .o
QMAKE_EXT_RES           = _res.o

QMAKE = qmake
```

```
QMAKE_CC                = arm-mingw32ce-gcc
QMAKE_LEX               = flex
QMAKE_LEXFLAGS          =
QMAKE_YACC              = byacc
QMAKE_YACCFLAGS         = -d
QMAKE_CFLAGS            =
QMAKE_CFLAGS_DEPS       = -M
QMAKE_CFLAGS_WARN_ON    = -w
QMAKE_CFLAGS_WARN_OFF   = -w
QMAKE_CFLAGS_RELEASE    = -O2
QMAKE_CFLAGS_DEBUG      = -g
QMAKE_CFLAGS_YACC       = -Wno-unused -Wno-parentheses

QMAKE_CXX               = arm-mingw32ce-g++
QMAKE_CXXFLAGS          = $$QMAKE_CFLAGS
QMAKE_CXXFLAGS_DEPS     = $$QMAKE_CFLAGS_DEPS
QMAKE_CXXFLAGS_WARN_ON  = $$QMAKE_CFLAGS_WARN_ON
QMAKE_CXXFLAGS_WARN_OFF = $$QMAKE_CFLAGS_WARN_OFF
QMAKE_CXXFLAGS_RELEASE  = $$QMAKE_CFLAGS_RELEASE
QMAKE_CXXFLAGS_DEBUG    = $$QMAKE_CFLAGS_DEBUG
QMAKE_CXXFLAGS_YACC     = $$QMAKE_CFLAGS_YACC
QMAKE_CXXFLAGS_THREAD   = $$QMAKE_CFLAGS_THREAD
QMAKE_CXXFLAGS_RTTI_ON  = -frtti
QMAKE_CXXFLAGS_RTTI_OFF = -fno-rtti
QMAKE_CXXFLAGS_EXCEPTIONS_ON = -fexceptions -mthreads
QMAKE_CXXFLAGS_EXCEPTIONS_OFF = -fno-exceptions

QMAKE_INCDIR            =
QMAKE_INCDIR_QT         = $$[QT_INSTALL_HEADERS]
QMAKE_LIBDIR_QT         = $$[QT_INSTALL_LIBS]

QMAKE_RUN_CC            = $(CC) -c $(CFLAGS) $(INCPATH) -o $obj $src
QMAKE_RUN_CC_IMP        = $(CC) -c $(CFLAGS) $(INCPATH) -o $@ $<
QMAKE_RUN_CXX           = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $obj $src
QMAKE_RUN_CXX_IMP       = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

QMAKE_LINK              = arm-mingw32ce-g++
QMAKE_LINK_C            = arm-mingw32ce-gcc
QMAKE_LFLAGS            = -enable-stdcall-fixup -Wl,-enable-auto-import -
   Wl,-enable-runtime-pseudo-reloc
QMAKE_LFLAGS_EXCEPTIONS_ON = -mthreads -Wl
QMAKE_LFLAGS_EXCEPTIONS_OFF =
```

```
QMAKE_LFLAGS_RELEASE    = -Wl,-s
QMAKE_LFLAGS_DEBUG      =
QMAKE_LFLAGS_CONSOLE    = -Wl,-subsystem,console
QMAKE_LFLAGS_WINDOWS    = -Wl,-subsystem,windows
QMAKE_LFLAGS_DLL        = -shared
QMAKE_LFLAGS_SHLIB      = -shared
QMAKE_LINK_OBJECT_MAX   = 10
QMAKE_LINK_OBJECT_SCRIPT= object_script


QMAKE_LIBS              =
QMAKE_LIBS_CORE         = -lole32 -lws2 -lcoredll
QMAKE_LIBS_GUI          = -loleaut32 -lole32 -luuid -lceshell -lcoredll
QMAKE_LIBS_NETWORK      = -lws2
QMAKE_LIBS_OPENGL       =  # -lopengl32 -lglu32 -lgdi32 -luser32
QMAKE_LIBS_COMPAT       = -laygshell -lceshell -lcomdlg -lcoredll -luuid
   -lws2
QMAKE_LIBS_QT_ENTRY     = -lmingw32 -lqtmain

QMAKE_TAR             = tar -cf
QMAKE_GZIP            = gzip -9f

QMAKE_DIR_SEP         = /
QMAKE_COPY            = cp -f
QMAKE_COPY_FILE       = cp
QMAKE_COPY_DIR        = cp -r
QMAKE_MOVE            = mv -f
QMAKE_DEL_FILE        = rm -f
QMAKE_DEL_DIR         = rmdir
QMAKE_STRIP           = arm-mingw32ce-strip
QMAKE_STRIPFLAGS_LIB += --strip-unneeded
QMAKE_CHK_DIR_EXISTS  = test -d
QMAKE_MKDIR           = || mkdir -p
QMAKE_INSTALL_FILE    = install -m 644 -p
QMAKE_INSTALL_PROGRAM = install -m 755 -p

QMAKE_MOC               = moc
QMAKE_UIC               = uic
QMAKE_IDC               = idc

QMAKE_IDL               = midl
QMAKE_LIB               = arm-mingw32ce-ar -ru
```

```
QMAKE_RC                = arm-mingw32ce-windres
QMAKE_ZIP               = zip -r -9

QMAKE_STRIP             = arm-mingw32ce-strip
QMAKE_STRIPFLAGS_LIB    += --strip-unneeded
QMAKE_INCDIR = /opt/mingw32ce/arm-mingw32ce/include /opt/mingw32ce/
   include /opt/mingw32ce/arm-mingw32ce/include/sys
QMAKE_LIBDIR = /opt/arm-mingw32ce/lib
load(qt_config)
```

A.2 Qt Plattformdefinitionen für CEGCC

Listing A.2: Qt Plattformdefinitionen für die CEGCC Werkzeugkette

```
/*
    ****************************************************************************


**
** Copyright (C) 2010 Nokia Corporation and/or its subsidiary(-ies).
** All rights reserved.
** Contact: Nokia Corporation (qt-info@nokia.com)
**
** This file is part of the qmake spec of the Qt Toolkit.
**
** $QT_BEGIN_LICENSE:LGPL$
** No Commercial Usage
** This file contains pre-release code and may not be distributed.
** You may use this file in accordance with the terms and conditions
** contained in the Technology Preview License Agreement accompanying
** this package.
**
** GNU Lesser General Public License Usage
** Alternatively, this file may be used under the terms of the GNU Lesser
** General Public License version 2.1 as published by the Free Software
** Foundation and appearing in the file LICENSE.LGPL included in the
** packaging of this file.  Please review the following information to
** ensure the GNU Lesser General Public License version 2.1 requirements
** will be met: http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html.
**
** In addition, as a special exception, Nokia gives you certain
   additional
** rights.  These rights are described in the Nokia Qt LGPL Exception
** version 1.1, included in the file LGPL_EXCEPTION.txt in this package.
**
** If you have questions regarding the use of this file, please contact
** Nokia at qt-info@nokia.com.
**
**
```

```
**
**
**
**
**
**
** $QT_END_LICENSE$
**
****************************************************************************
    */

#ifndef QPLATFORMDEFS_H
#define QPLATFORMDEFS_H

#ifdef UNICODE
#ifndef _UNICODE
#define _UNICODE
#endif
#endif

// http://msdn.microsoft.com/en-us/library/ms886726.aspx

#define PF_ARM_INTEL_WMMX 0x8001003
#define _O_BINARY 0x8000
// http://msdn.microsoft.com/en-us/library/bb431750.aspx

// These Should be defined in Winuser.h
#define VK_TBACK 0x08
#define MOD_KEYUP 0x1000
#define WM_IME_REQUEST 0x0288
#define INI_INTL 0x01
#define IM_SPELL 0
#define WS_EX_NOANIMATION 0x04000000L

// These Should be defined in TCPSHELL
#define IME_ESC_SET_MODE 0x0800

#define getpid _getpid
```

```
// This is really ugly
#include "/opt/mingw32ce/arm-mingw32ce/include/float.h"
//#define _clear87 _clearfp
#define _control87 _controlfp_s
// For QLocale.cpp
inline bool isascii(int c)
{
     return (c >= 0 && c <=127);
}
typedef int errno_t;
// Get Qt defines/settings
#include "qglobal.h"
#include "qfunctions_wince.h"
#include <windef.h>
#define _POSIX_
#include <limits.h>
#undef _POSIX_
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>
#undef QT_NO_ICONV
#include <windows.h>
// Workaround for _NO_OLDNAMES definition taken from sys/stat.h of the
// Mingw32ce API
struct stat
{
    _dev_t   st_dev;/* Equivalent to drive number 0=A 1=B */
    _ino_t   st_ino;     /* Always zero ? */
    mode_t  st_mode;    /* See above constants */
    short   st_nlink;   /* Number of links  */
    short   st_uid;     /* User: Maybe significant on NT ? */
    short   st_gid;     /* Group: Ditto */
    _dev_t   st_rdev;    /* Seems useless (not even filled in) */
    _off_t   st_size;    /* File size in bytes */
    time_t  st_atime;   /* Accessed date (always 00:00 hrs local on FAT)
        */
    time_t  st_mtime;   /* Modified time */
    time_t  st_ctime;   /* Creation time */
```

```
};

#define wcsdup _wcsdup
#define Q_FS_FAT
#ifdef QT_LARGEFILE_SUPPORT
#define QT_STATBUF              struct _stati64
#define QT_STATBUF4TSTAT        struct _stati64
#define QT_STAT                 ::_stati64
#define QT_FSTAT                ::_fstati64
#else
#define QT_STATBUF struct stat
#define QT_STATBUF4TSTAT        struct stat
#define QT_STAT                 ::qt_wince_stat
#define QT_FSTAT                ::qt_wince__fstat
#endif
#define QT_STAT_REG             _S_IFREG
#define QT_STAT_DIR             _S_IFDIR
#define QT_STAT_MASK            _S_IFMT
#if defined(_S_IFLNK)
#  define QT_STAT_LNK           _S_IFLNK
#endif
#define QT_FILENO               ::qt_wince___fileno
#define QT_OPEN                 ::qt_wince_open
#define QT_CLOSE                ::qt_wince__close
#ifdef QT_LARGEFILE_SUPPORT
#define QT_LSEEK                ::_lseeki64
#define QT_TSTAT                ::_tstati64
#else
#define QT_LSEEK                ::qt_wince__lseek
#define QT_TSTAT                ::_tstat
#endif
#define QT_READ                 ::qt_wince__read
#define QT_WRITE                ::qt_wince__write
#define QT_ACCESS               ::qt_wince__access
#define QT_GETCWD               ::_getcwd
#define QT_CHDIR                ::_chdir
#define QT_MKDIR                ::qt_wince__mkdir
#define QT_RMDIR                ::qt_wince__rmdir
```

```
#define QT_OPEN_LARGEFILE       0
#define QT_OPEN_RDONLY          _O_RDONLY
#define QT_OPEN_WRONLY          _O_WRONLY
#define QT_OPEN_RDWR            _O_RDWR
#define QT_OPEN_CREAT           _O_CREAT
#define QT_OPEN_TRUNC           _O_TRUNC
#define QT_OPEN_APPEND          _O_APPEND
# define QT_OPEN_TEXT           _O_TEXT
# define QT_OPEN_BINARY         _O_BINARY

#define QT_FOPEN                ::fopen
#define QT_FSEEK                ::fseek
#define QT_FTELL                ::ftell
#define QT_FGETPOS              ::fgetpos
#define QT_FSETPOS              ::fsetpos
#define QT_MMAP                 ::mmap
#define QT_FPOS_T               fpos_t
#define QT_OFF_T                long

#define QT_SIGNAL_ARGS          int

#define QT_VSNPRINTF(buffer, count, format, arg) \
    _vsnprintf(buffer, count, format, arg)

#define QT_SNPRINTF             ::_snprintf

# define F_OK   0
# define X_OK   1
# define W_OK   2
# define R_OK   4

#endif // QPLATFORMDEFS_H
```

A.3 Änderungen am Qt-GUI Modul zur Erstellung mit CEGCC Werkzeugen

Listing A.3: Diff des Qt-GUI Moduls gegen Version 576cb12c

```
diff --git a/src/gui/gui.pro b/src/gui/gui.pro
index a6370b2..8e8dd0f 100644
--- a/src/gui/gui.pro
+++ b/src/gui/gui.pro
@@ -1,7 +1,7 @@
 TARGET      = QtGui
 QPRO_PWD    = $$PWD
 QT = core
-DEFINES    += QT_BUILD_GUI_LIB QT_NO_USING_NAMESPACE
+DEFINES    += QT_STYLE_WINDOWSMOBILE QT_BUILD_GUI_LIB
    QT_NO_USING_NAMESPACE
 win32-msvc*|win32-icc:QMAKE_LFLAGS += /BASE:0x65000000

 !win32:!embedded:!mac:!symbian:CONFIG       += x11
@@ -15,8 +15,7 @@ contains(QT_CONFIG, x11sm):CONFIG += x11sm
 #platforms
 x11:include(kernel/x11.pri)
 mac:include(kernel/mac.pri)
-win32:include(kernel/win.pri)
-embedded:include(embedded/embedded.pri)
+win32|wince*:include(kernel/win.pri)
 symbian {
     include(kernel/symbian.pri)
     include(s60framework/s60framework.pri)
diff --git a/src/gui/image/image.pri b/src/gui/image/image.pri
index 8d75fdd..96331e4 100644
--- a/src/gui/image/image.pri
+++ b/src/gui/image/image.pri
@@ -55,10 +55,10 @@ SOURCES += \
         image/qimagepixmapcleanuphooks.cpp \
```

```
-win32 {
+win32|wince* {
     SOURCES += image/qpixmap_win.cpp
 }
-embedded {
+embedded:!wince* {
     SOURCES += image/qpixmap_qws.cpp
 }
 x11 {
diff --git a/src/gui/inputmethod/inputmethod.pri b/src/gui/inputmethod/
    inputmethod.pri
index 02e3e57..510c13f 100644
--- a/src/gui/inputmethod/inputmethod.pri
+++ b/src/gui/inputmethod/inputmethod.pri
@@ -11,11 +11,11 @@ x11 {
     HEADERS += inputmethod/qximinputcontext_p.h
     SOURCES += inputmethod/qximinputcontext_x11.cpp
 }
-win32 {
+win32|wince* {
     HEADERS += inputmethod/qwininputcontext_p.h
     SOURCES += inputmethod/qwininputcontext_win.cpp
 }
-embedded {
+embedded:!wince* {
     HEADERS += inputmethod/qwsinputcontext_p.h
     SOURCES += inputmethod/qwsinputcontext_qws.cpp
 }
diff --git a/src/gui/kernel/kernel.pri b/src/gui/kernel/kernel.pri
index 6fd45ad..aad092e 100644
--- a/src/gui/kernel/kernel.pri
+++ b/src/gui/kernel/kernel.pri
@@ -3,7 +3,6 @@
 # Only used on platforms with CONFIG += precompile_header
 PRECOMPILED_HEADER = kernel/qt_gui_pch.h

-
 KERNEL_P= kernel
```

```
 HEADERS += \
         kernel/qaction.h \
@@ -135,7 +134,7 @@ symbian {
 }



-unix:x11 {
+unix:!wince*:x11 {
         INCLUDEPATH += ../3rdparty/xorg
         HEADERS += \
                 kernel/qx11embed_x11.h \
@@ -171,7 +170,7 @@ unix:x11 {
                  kernel/qeventdispatcher_x11_p.h
 }

-embedded {
+embedded:!wince* {
         HEADERS += \
                 kernel/qeventdispatcher_qws_p.h

@@ -219,8 +218,7 @@ embedded {
                  kernel/qcursor_mac.mm \
                  kernel/qdnd_mac.mm \
                  kernel/qsound_mac.mm  \
-                 kernel/qapplication_mac.mm \
-                 kernel/qwidget_mac.mm \
+
                  kernel/qcocoapanel_mac.mm \
                  kernel/qcocoaview_mac.mm \
                  kernel/qcocoawindow_mac.mm \
diff --git a/src/gui/kernel/qclipboard.h b/src/gui/kernel/qclipboard.h
index edaa260..51cb567 100644
--- a/src/gui/kernel/qclipboard.h
+++ b/src/gui/kernel/qclipboard.h
@@ -41,7 +41,7 @@

 #ifndef QCLIPBOARD_H
 #define QCLIPBOARD_H
```

```
_
+#include <wchar.h>
 #include <QtCore/qobject.h>

 QT_BEGIN_HEADER
diff --git a/src/gui/kernel/qguifunctions_wince.h b/src/gui/kernel/
    qguifunctions_wince.h
index 932ee90..fa7b4e6 100644
--- a/src/gui/kernel/qguifunctions_wince.h
+++ b/src/gui/kernel/qguifunctions_wince.h
@@ -44,7 +44,6 @@
 #include <QtCore/qfunctions_wince.h>
 #define UNDER_NT
 #include <wingdi.h>
_
 #ifdef QT_BUILD_GUI_LIB
 QT_BEGIN_HEADER
 QT_BEGIN_NAMESPACE
@@ -77,6 +76,7 @@ int qt_wince_GetDIBits(HDC, HBITMAP, uint, uint, void*,
     LPBITMAPINFO, uint);
 #define ALTERNATE 0
 #define WINDING 1

+#ifndef Q_CC_MINGW
 // QFontEngine
 typedef struct _FIXED {
   WORD  fract;
@@ -116,7 +116,7 @@ typedef struct tagTTPOLYCURVE
     WORD    cpfx;
     POINTFX apfx[1];
 } TTPOLYCURVE;
_
+#endif
 #define GGO_NATIVE 2
 #define GGO_GLYPH_INDEX 0x0080
 #define TT_PRIM_LINE 1
diff --git a/src/gui/kernel/qkeymapper_win.cpp b/src/gui/kernel/
    qkeymapper_win.cpp
```

```
index f84b902..ad5d8c3 100644
--- a/src/gui/kernel/qkeymapper_win.cpp
+++ b/src/gui/kernel/qkeymapper_win.cpp
@@ -76,6 +76,12 @@ extern Q_CORE_EXPORT QLocale qt_localeFromLCID(LCID id
    );
 #endif

 #if defined(Q_OS_WINCE)
+// Resolve Naming conflicts, use Qt_prefix for functions
+// that may be declaed by mingw for wince from here on
+#define ToAscii Qt_ToAscii
+#define ToUnicode Qt_ToUnicode
+#define GetKeyboardState Qt_GetKeyboardState
+
 bool GetKeyboardState(unsigned char* kbuffer)
 {
     for (int i=0; i< 256; ++i)
diff --git a/src/gui/kernel/qwindowdefs_win.h b/src/gui/kernel/
   qwindowdefs_win.h
index edcb1db..992a97b 100644
--- a/src/gui/kernel/qwindowdefs_win.h
+++ b/src/gui/kernel/qwindowdefs_win.h
@@ -110,6 +110,7 @@ Q_DECLARE_HANDLE(HRGN);
 #ifndef HMONITOR
 Q_DECLARE_HANDLE(HMONITOR);
 #endif
+
 #ifndef HRESULT
 typedef long HRESULT;
 #endif
diff --git a/src/gui/painting/painting.pri b/src/gui/painting/painting.
   pri
index ed8ee76..e635167 100644
--- a/src/gui/painting/painting.pri
+++ b/src/gui/painting/painting.pri
@@ -96,7 +96,7 @@ SOURCES += \
                 painting/qrasterdefs_p.h                \
                 painting/qgrayraster_p.h
```

```
-win32 {
+win32|wince* {
        HEADERS += painting/qprintengine_win_p.h

        SOURCES += \
@@ -108,7 +108,7 @@ win32 {
        !win32-borland:!wince*:LIBS += -lmsimg32
 }

-embedded {
+embedded:!wince* {
     HEADERS += \
        painting/qgraphicssystem_qws_p.h \

@@ -167,7 +167,7 @@ win32|x11|mac|embedded|symbian {
        HEADERS += painting/qbackingstore_p.h
 }

-embedded {
+embedded:!wince* {
        contains(QT_CONFIG,qtopia) {
                DEFINES += QTOPIA_PRINTENGINE
                HEADERS += painting/qprintengine_qws_p.h
@@ -189,7 +189,7 @@ symbian {
                 painting/qpaintengine_s60_p.h
 }

-x11|embedded {
+x11|embedded:!wince {
        contains(QT_CONFIG,qtopia) {
            DEFINES += QT_NO_CUPS QT_NO_LPR
        } else {
@@ -360,7 +360,7 @@ mac {
        SOURCES += painting/qwindowsurface_mac.cpp
 }

-embedded {
```

```
+embedded:!wince* {
         HEADERS += painting/qwindowsurface_qws_p.h
         SOURCES += painting/qwindowsurface_qws.cpp
 }
diff --git a/src/gui/styles/qwindowsstyle.cpp b/src/gui/styles/
   qwindowsstyle.cpp
index 1653baa..0ad76ff 100644
--- a/src/gui/styles/qwindowsstyle.cpp
+++ b/src/gui/styles/qwindowsstyle.cpp
@@ -38,7 +38,6 @@
 ** $QT_END_LICENSE$
 **
 ****************************************************************************/

-
 #include "qwindowsstyle.h"
 #include "qwindowsstyle_p.h"
 #include <private/qstylehelper_p.h>
diff --git a/src/gui/styles/styles.pri b/src/gui/styles/styles.pri
index 5084442..836e87c 100644
--- a/src/gui/styles/styles.pri
+++ b/src/gui/styles/styles.pri
@@ -156,7 +156,7 @@ contains( styles, windowsce ) {
         DEFINES += QT_NO_STYLE_WINDOWSCE
 }

-contains( styles, windowsmobile ) {
+wince*:{
         HEADERS += styles/qwindowsmobilestyle.h
         SOURCES += styles/qwindowsmobilestyle.cpp
 } else {
diff --git a/src/gui/text/text.pri b/src/gui/text/text.pri
index 9ec3142..02a0599 100644
--- a/src/gui/text/text.pri
+++ b/src/gui/text/text.pri
@@ -71,11 +71,12 @@ SOURCES += \
         text/qtextodfwriter.cpp \
         text/qstatictext.cpp
```

```
-win32 {
+win32|wince* {
        SOURCES += \
                text/qfont_win.cpp \
                text/qfontengine_win.cpp
        HEADERS += text/qfontengine_win_p.h
+     DEFINES += QT_NO_FONTCONFIG
 }

 unix:x11 {
@@ -95,7 +96,7 @@ unix:x11 {
         OBJECTIVE_SOURCES += text/qfontengine_mac.mm
 }

-embedded {
+embedded:!wince* {
        SOURCES += \
                text/qfont_qws.cpp \
                text/qfontengine_qws.cpp \
@@ -174,7 +175,7 @@ contains(QT_CONFIG, freetype) {
            ../3rdparty/freetype/src/autofit/afloader.c\
            ../3rdparty/freetype/src/autofit/autofit.c

-     symbian {
+     symbian|wince* {
          SOURCES += \
              ../3rdparty/freetype/src/base/ftsystem.c
      } else {
diff --git a/src/gui/widgets/qmenu.h b/src/gui/widgets/qmenu.h
index a040afa..60af094 100644
--- a/src/gui/widgets/qmenu.h
+++ b/src/gui/widgets/qmenu.h
@@ -46,11 +46,10 @@
 #include <QtCore/qstring.h>
 #include <QtGui/qicon.h>
 #include <QtGui/qaction.h>
-
```

```
 #ifdef QT3_SUPPORT
 #include <QtGui/qpixmap.h>
 #endif
-
+#include <windef.h>
 QT_BEGIN_HEADER

 QT_BEGIN_NAMESPACE
diff --git a/src/gui/widgets/qmenubar_p.h b/src/gui/widgets/qmenubar_p.h
index 82070fe..d7adb32 100644
--- a/src/gui/widgets/qmenubar_p.h
+++ b/src/gui/widgets/qmenubar_p.h
@@ -208,7 +208,7 @@ public:
     struct QWceMenuBarPrivate {
         QList<QWceMenuAction*> actionItems;
         QList<QWceMenuAction*> actionItemsLeftButton;
-        QList<QList<QWceMenuAction*>> actionItemsClassic;
+        QList<QList<QWceMenuAction*> > actionItemsClassic;
         HMENU menuHandle;
         HMENU leftButtonMenuHandle;
         HWND menubarHandle;
```

A.4 CD Inhalt

Ort	Dateityp	Beschreibung
/Bachelorarbeit.pdf	PDF Datei	Gesamttext im Portable Document Format
/tex	Verzeichnis	Verwendete Latex Quelldateien
/pocketpc-cab.pl	Perl Skript	Werkzeug zum Erstellen von CAB-Installern
/geninputfiles.py	Python Skript	Hilfsmittel zum Erstellen von CAB-Installern
/OpenSSH.cab	CAB-Installer	CE-Installationsarchiv für OpenSSH Server
/verschiedenes/wcecompat.txt	ASCII Text	Exportdefinitionen von WCECOMPAT
/verschiedenes/wceex_functions.txt	ASCII Text	Exportdefinitionen von WCELIBCEX
/verschiedenes/mingwdumps	Verzeichnis	Exportdefinitionen von MinGW32ce
/verschiedenes/msvcdumps	Verzeichnis	Exportdefinitionen der Windows CE API
/images	Verzeichnis	In der Arbeit verwendete Bilder
/hello_world	Verzeichnis	Dateien des Hello World Beispiels
/Anhang	ASCII text	Dateien aus Anhang 1-3

Erklärung

Hiermit versichere ich, dass ich meine Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)