

1 Grundlagen

Das erste Kapitel erklärt die Grundlagen der Anwendungsentwicklung für das Symbian-Betriebssystem. Zuerst wird kurz auf die Bedeutung des Begriffes »Smartphone« eingegangen und erörtert, was das eigentlich ist. Ein Betriebssystem für Smartphones ist kein gewöhnliches Betriebssystem. Es muss bestimmte Voraussetzungen erfüllen. Welche dies sind und wie das Symbian-Betriebssystem entstanden ist, wird in Kapitel 1.1.2 besprochen, wobei kurz auf die verschiedenen Anwendungen von Symbian eingegangen wird. Welche Endgeräte werden unterstützt? Kapitel 1.2 erklärt die Grundlagen der Entwicklungsumgebung. Welche Anforderungen gibt es an ein SDK und welche Tools helfen bei der Softwareentwicklung? Das Kapitel über die Grundlagen endet mit der Erstellung der ersten eigenen Anwendung mithilfe des Application Wizard. Die Anwendung wird im Emulator ausgeführt. Schneller als man denkt, hat man eine eigene Anwendung entwickelt und könnte sich als Symbian-OS-Programmierer bezeichnen!

1.1 Betriebssysteme für Smartphones

1.1.1 Was ist ein Smartphone?

Schlägt man den Begriff »smart« im Wörterbuch einmal nach, so findet man Übersetzungen wie »elegant«, »fesch«, »gerissen«, »geschickt« oder »pfiffig«. Man fragt sich also, was ein »Smartphone« wohl ist. Handelt es sich dabei etwa einfach um ein besonders »fesches« Telefon? Oder vielleicht sogar um ein besonders »gerissenes«, das automatisch den günstigsten Tarif wählt oder sogar von selbst weiß, wen man anrufen möchte und automatisch die richtige Nummer wählt?

Nützliche kleine Helfer

Leider nein. Aber dennoch kann sich ein Smartphone als nützlicher, kleiner Helfer erweisen. Es hilft beim Verwalten von Terminen und Kontakten, erlaubt es, mobil im Internet zu surfen, dient als Gedächtnisstütze durch Sprachmemos, Aufgabenlisten und vieles mehr. Es ist also wie ein kleiner Taschencomputer mit Telefonfunktion. Nun könnte man aber behaupten, dass ein normales Mobiltelefon diese Möglichkeiten auch bietet. Warum wird dieses also nicht auch als Smartphone bezeichnet? Der größte Unterschied zwischen einem »normalen« Mobiltelefon und einem Smartphone liegt vor allem im Betriebssystem. Eines dieser Betriebssysteme für Smartphones ist das Symbian OS. Symbian OS ist allerdings nicht das Einzige dieser Art. Microsoft macht mit seinem Smartphone-Betriebssystem Konkurrenz [Microsoft] und auch Linux wurde schon auf einem Mobiltelefon gesichtet. Seit kurzem ist auch Palm Source auf den Smartphone-Markt aufmerksam geworden und bietet ein entsprechendes Betriebssystem an [Palm]. Allerdings ist Symbian OS mit allein im Jahr 2003 ca. acht Millionen verkauften Smartphones das derzeit erfolgreichste Betriebssystem dieser Sparte und deckt ca. 80 Prozent des weltweiten Mobilfunkmarktes ab [Teltarif].

Telefon und mehr

Was macht ein Smartphone außer dem Betriebssystem nun wirklich aus? Es gibt keine einzelne Eigenschaft, die ein Smartphone auszeichnet. Klar ist aber, dass ein Smartphone bestimmte Fähigkeiten besitzt:

- Größe und Form eines Mobiltelefons
- Grafikfähiges Farbdisplay
- Anwendungen wie E-Mail, Kalender, Adressbuch usw.
- Erweiterbarkeit durch Installation neuer Anwendungen

Die wichtigste Eigenschaft eines Smartphones ist wahrscheinlich, dass es im Gegensatz zu »normalen« Mobiltelefonen möglich ist, eigene Anwendungen zu entwickeln und diese dann auf dem Gerät zu installieren. Dies eröffnet unglaublich viele Möglichkeiten. Manch einer wird nun argumentieren, dass sein Mobiltelefon auch erweiterbar ist, und zwar mit Java-Anwendungen. Hierzu ist aber anzumerken, dass diese Anwendungen sehr stark in ihren Möglichkeiten eingeschränkt sind. Oftmals kann man nur sehr kleine (ca. 60 KB) Anwendungen installieren und diese laufen dann in einer vom eigentlichen Betriebssystem abgeschotteten »Sandbox«, vergleichbar mit den Java Applets im Webbrowser.

Grenzenlos erweiterbar

Diese »Sandbox« verhindert aus Sicherheitsgründen den Zugriff einer Java-Anwendung (einem so genannten MIDlet) auf die meisten Betriebssystemfunktionen und andere Anwendungen. Dies ist ja vom

Prinzip her auch gut so, nur schränkt es einen Softwareentwickler sehr ein, und es mangelt daher auch oft an sinnvollen Java-Anwendungen für Mobiltelefone. Mit Symbian OS dagegen hat man die Möglichkeit, als Softwareentwickler auf alle Betriebssystemfunktionen zuzugreifen. Man ist frei und kann im Prinzip fast alles damit realisieren. Das Symbian-Konsortium bietet also für ihr gleichnamiges Betriebssystem eine Softwareentwicklungsumgebung an, ein so genanntes Software Development Kit (SDK), das in diesem Buch näher betrachtet wird.

1.1.2 Betriebssystemanforderungen

Nach der groben Definition dessen, was ein Smartphone »smart« macht (nämlich die Möglichkeit zum Laden von Anwendungen), ist die nächste Frage, welche Anforderungen an ein Betriebssystem für Smartphones gestellt werden. Ein Smartphone ist wie andere Mobiltelefone auch ein so genanntes »embedded system«. Für ein Betriebssystem gelten in diesem Zusammenhang bestimmte Anforderungen. Wie ein »normales« Mobiltelefon ist ein Smartphone ein Massenprodukt. Die Geräte dürfen also nicht allzu teuer sein, um die entsprechenden Stückzahlen zu erreichen. Da es sich um Massenprodukte handelt, wäre es extrem teuer, ein Problem zu beheben, das erst nach der Auslieferung auffällt. Das bedeutet wiederum, dass das Gerät und damit auch das Betriebssystem insgesamt qualitativ hochwertig und vielfach getestet sein muss. Alle verkauften Geräte nachträglich mit einer neuen Betriebssystemversion auszustatten, wäre ein schwieriges und teures Unterfangen.

Zusätzlich sollte das Betriebssystem sehr verlässlich und stabil laufen. Anders als bei PC-Betriebssystemen, bei denen man Abstürze und Probleme in Kauf nimmt, wäre ein regelmäßiger Systemabsturz auf einem Smartphone fatal für die Akzeptanz des Gerätes. Ein Mobiltelefon soll ja immer und überall (solange eine Netzverbindung besteht) funktionieren. Wie würden Sie reagieren, wenn Ihnen das Betriebssystem mitten in einem Gespräch abstürzt?

Stabilität und Sicherheit

Eine weitere Einschränkung besteht darin, dass mobile Endgeräte nur wenig Strom verbrauchen dürfen, damit sie nicht ständig aufgeladen werden müssen. Sie haben meist wenig Speicherplatz im Vergleich zu z. B. PCs, da sie keine Festplatte besitzen und daher alle Daten in relativ teuren Flash-RAM-Bausteinen persistent gehalten werden müssen. Smartphones, mehr noch als andere mobile Endgeräte wie PDAs, müssen sehr schonend mit ihren Ressourcen umgehen. Das betrifft natürlich insbesondere für Betriebssystem, da es hauptsächlich für die Verwaltung der Ressourcen zuständig ist. Aber nicht nur das Betriebs-

Stromverbrauch

system selbst ist für einen schonenden Umgang mit den Ressourcen zuständig, auch der Anwendungsentwickler muss sich darüber Gedanken machen. Vor allem dann, wenn seine Anwendung intensive Berechnungen durchführt und damit die Auslastung der CPU und der Stromverbrauch steigt.

*Anforderungen wie an ein
Echtzeitbetriebssystem*

Zusätzlich zu diesen Einschränkungen, mit denen auch Betriebssysteme für PDAs kämpfen müssen, muss ein Betriebssystem für Smartphones noch den Dienst als Mobiltelefon leisten. Mobilfunknetze erfordern aber von Endgeräten vorausbestimmbare und möglichst schnelle Antwortzeiten. Daher werden in diesem Bereich schnelle und vorausbestimmbare Antwortzeiten auch vom Betriebssystem des Smartphones gefordert, welche denen eines Echtzeitbetriebssystems ähneln.

1.1.3 Die Geschichte von Symbian

Symbian ist verglichen mit anderen aktuellen Systemen ein relativ junges Betriebssystem. Das liegt natürlich vor allem an der rasanten Entwicklung der letzten Jahre im Bereich der mobilen Endgeräte. Aber im Gegensatz zu anderen Betriebssystemen für mobile Endgeräte basiert Symbian auf den Erfahrungen von fast zwei Jahrzehnten im Bereich der Entwicklung mobiler Endgeräte. Seinen Ursprung fand Symbian im Jahr 1981, als David Potter, ein Physiker und Dozent am Imperial College in London, die Firma Psion gründete. Psion ist abgeleitet von Potter Scientific Investment (PSI). Da dieser Name aber leider schon vergeben war, fügte Potter einfach noch zwei Buchstaben hinzu. Psion klingt für einen Physiker aufregend, wie ein neu entdecktes subatomares Teilchen. In den ersten drei Jahren hatte sich die Firma der Softwareentwicklung für Sinclair Rechner verschrieben. 1984 aber entwickelte Psion die erste eigene Hardware und brachte im Oktober 1984 den weltweit ersten Handheld Computer heraus. Der Psion Organizer I war mit ganzen 2 KB Hauptspeicher (ausbaubar bis 16 KB) ausgestattet und besaß einen Hitachi-6301-Prozessor mit 0,92 MHz. Das Display war monochrom mit einer Zeile, die 16 Buchstaben fassen konnte (siehe Abb. 1–1).

*Die Entwicklung des
Symbian OS beginnt 1994*

Weitere Geräte folgten. Im Jahr 1991 waren die Psion Organizer schon in der dritten Generation und gegenüber Apple, Sharp und HP in Führung. Diese Geräte liefen mit dem Betriebssystem SIBO («Sixteen-Bit-Organizer» oder »single-board») und waren ein durchschlagender Erfolg. Nach einer weniger erfolgreichen Exkursion von Psion auf dem Laptop-Markt entschied man sich 1994 für die Entwicklung eines komplett neuen Betriebssystems mit dem Codenamen »Protea«.

Grund dafür waren Einschränkungen des 16-Bit-SIBO-Systems, wie z. B. die fehlende Unterstützung für Zeigegeräte (Maus oder Stift). Auch war das alte Betriebssystem auf die x86-Prozessor-Architektur beschränkt und es fehlte die Unterstützung für Unicode.



Abb. 1-1

Psion Organizer I

Im Jahr 1997 wurde EPOC (steht für Epoche) veröffentlicht und lief als Erstes auf Psions 5mx. Weitere Geräte, die mit diesem Betriebssystem liefen und noch betrieben werden, sind Psions netBook, Ericssons MC218, Psions Series 7 und Revo. Um das neue Betriebssystem zu vermarkten, entwickelte Psion ein Lizenzierungsmodell für EPOC. Obwohl sie damit recht erfolgreich waren, hatten die Marketingverantwortlichen bei Psion eine bessere Idee, um die weite Verbreitung des neuen Betriebssystems zu garantieren: Im Juni 1998 wurde die von der Psion Group unabhängige Firma Symbian gegründet, an der neben Psion die vier größten Mobiltelefonhersteller beteiligt waren – Nokia, Ericsson, Motorola und Panasonic [Symbian].

Diese Allianz der größten Hersteller mobiler Endgeräte treibt bis heute die Entwicklung des nun Symbian bzw. Symbian OS genannten Betriebssystems voran. Mittlerweile traten dieser Allianz einige weitere Firmen wie Siemens, Samsung und Sony-Ericsson bei. Weitere Firmen wie Benq lizenzierten das Produkt. Motorola entschied sich allerdings Mitte 2003, seine Anteile an Symbian zu verkaufen und das Betriebssystem nur noch zu lizenzieren. Motorola bietet nun neben Symbian-Geräten auch Smartphones mit dem Konkurrenzprodukt von Microsoft und mit dem freien Linux-Betriebssystem an. Das erste Endgerät mit dem neuen offiziellen Symbian OS war der Nokia 9210 Communicator im Jahr 2001.

*Breite Akzeptanz bei
den Herstellern*

1.1.4 Symbian-Anwendungen

Flexibles Look&Feel

Eine große Stärke von Symbian ist seine Flexibilität. Es kann für die unterschiedlichsten Endgeräteklassen eingesetzt werden. Dabei ist die Basis des Betriebssystems und die der API des Betriebssystems immer die gleiche. Der einzige Unterschied zwischen den verschiedenen Versionen für die unterschiedlichen Geräteklassen besteht in den Teilen, die für die Darstellung der Oberfläche zuständig sind. Dies ermöglicht einem Hersteller mobiler Endgeräte, auf einfache Weise das Aussehen und die Bedienung (das so genannte Look&Feel) seiner Geräte völlig individuell zu gestalten. Ein wichtiges Argument, wenn es darum geht, sich von der Konkurrenz abzusetzen. Der Anwender merkt nicht, dass es sich überall um dasselbe Betriebssystem handelt und nur die Oberfläche anders aussieht. Abbildung 1–2 zeigt die aktuellen drei Betriebssystemversionen und ihre Anwendungen.

Abb. 1–2
Symbian-Geräteklassen
und ihre Anwendungen



Series 60

Das Series 60 (Pearl) ist die Version, die momentan am weitesten verbreitet ist und zur Zeit auf ca. sechs Geräten von verschiedenen Herstellern eingesetzt wird. Dabei handelt es sich meistens um Smartphones, die der ersten Geräteklasse entsprechen. Das sind Geräte, die äußerlich nicht viel von einem »normalen« Mobiltelefon unterscheidet. Das heißt, sie besitzen ein Farbdisplay und »nur« eine normale Telefonatatur. Man nennt diese Geräte sprachzentrisch mit erweiterten Informations-Funktionen, da sie vor allem zum Telefonieren gedacht sind,

aber trotzdem alle Möglichkeiten eines Organizers bieten. Sie sind mit einer Hand bedienbar, besitzen einen integrierten Webbrowser und einfache, aber gute Adress- und Kalender-Funktionen.

Die zweite Geräteklasse entspricht mehr einem Organizer, wie man sie von den Psion-Geräten kennt. Es handelt sich dabei um so genannte informationszentrische Geräte mit Telefonie-Funktion. Sie besitzen meist eine voll funktionsfähige Tastatur, ein relativ großes Farbdisplay (ideal zum Betrachten und Erstellen von Daten) und einen professionellen E-Mail Client und Webbrowser. Hier gibt es zur Zeit eigentlich nur einen Vertreter auf dem Markt: das Nokia 9210. Da diese Version des Betriebssystems so selten eingesetzt wird, wird in diesem Buch auch nicht gesondert darauf eingegangen. Die Basis der verschiedenen Betriebssystemversionen ist ohnehin identisch und daher ist es auch keine große Umstellung von Series 60 auf Series 80. Falls man also Anwendungen für das Nokia 9210 entwickeln möchte, bekommt man in diesem Buch alle grundsätzlich notwendigen Informationen dafür.

Series 80

Die dritte Geräteklasse erinnert mehr an eine Mischung aus heutigen PDAs und Mobiltelefonen. Diese Klasse entspricht am meisten dem, was man im Allgemeinen unter einem »Smartphone« versteht. Hierbei handelt es sich ebenfalls um informationszentrische Geräte mit Telefonie-Funktion. Diese besitzen einen Touchscreen und werden mit einem Stift bedient. Manche Geräte wie das P800 besitzen allerdings zusätzlich noch eine Telefontastatur, die abnehmbar ist. Wie die anderen Geräteklassen auch besitzen diese Geräte Kalender, Adressbuch, E-Mail und Webbrowser. Die Daten- und Telefoniefunktionen sind hierbei also voll integriert. Als dieses Buch geschrieben wurde, waren drei Smartphones mit dieser Betriebssystemversion auf dem Markt vertreten: das Sony Ericsson P800, das Motorola A920 und das Benq P30. Insgesamt waren 11 Geräte mit dem Symbian-OS-Betriebssystem auf dem Markt und drei weitere kurz vor dem Erscheinen.

UIQ

Während der Niederschrift dieses Buches erschien eine ganz neue GUI-Version des Symbian-Betriebssystems: das Series 90. Auf Basis der Symbian-Version 7.0 aufgebaut, ist diese Version für PDA-ähnliche mobile Medien-Geräte gedacht. Das erste dieser Klasse ist das Nokia 7700, welches einen Touchscreen mit einer Auflösung von 640×320 Pixel besitzt und nur mit einem Stift bedient wird. Als neuestes Feature erlaubt diese Plattform-Version das Anlegen von benutzerdefinierten Skins, um die Benutzeroberfläche nach Belieben jederzeit umgestalten zu können.

1.2 Entwicklungsumgebungen: Series-60- und UIQ-7.0-SDK

Dieses Buch beschäftigt sich hauptsächlich mit zwei Versionen des Symbian-Betriebssystems: dem Series 60 (Version 6.1) und dem UIQ (Version 7.0), da diese am häufigsten eingesetzt werden. Trotzdem bietet dieses Buch eine gute Grundlage für das Erlernen der Programmierung anderer Versionen des Betriebssystems, wie z. B. der Series 80 (Version 6.0) oder Series 90 (Version 7.0), da sich diese im Prinzip nur in den Bereichen unterscheiden, die für die grafische Oberfläche zuständig sind.

*Jeder kann Software
für sein Mobiltelefon
entwickeln*

Symbian OS ist eine offene, d. h. für jeden zugängliche Plattform. Dies ermöglicht erstmals die Entwicklung von Software für Mobiltelefone durch jedermann. Symbian bietet dazu den kostenlosen Download der verschiedenen Software Development Kits (SDKs) für die verschiedenen Betriebssystemversionen an [SymbianSDK]. Software für Symbian wird hauptsächlich in C++ geschrieben. Das Betriebssystem unterstützt aber grundsätzlich auch Java und die Sprache OPL, die der Sprache Basic ähnlich ist. Es besteht sogar die Möglichkeit, in Visual Basic Anwendungen für die Nokia Communicator der 9200er Serie zu entwickeln. Wir beschränken uns hier aber auf die Entwicklung von Anwendungen in C++. Mit C++ unterliegt man keinen Einschränkungen einer Runtime Engine wie bei Java oder OPL und es gibt hier auch entsprechend gute Unterstützung durch Integrated Development Environments (IDEs) wie z. B. Microsoft Visual Studio 6.0 oder CodeWarrior von Metrowerks.

1.2.1 Anforderungen an ein SDK

*Modularität ist
entscheidend*

Neben den Anforderungen an das Betriebssystem, wie hohe Stabilität, gute Performance, kleiner Speicherverbrauch usw., gibt es auch Anforderung aus Sicht eines Entwicklers, die für den Erfolg eines Betriebssystems von entscheidender Bedeutung sind. Eine Anwendung, die modular aufgebaut ist, ist leicht erweiterbar und Fehler können einfacher gefunden und behoben werden. Dasselbe gilt auch für ein SDK. Der modulare Aufbau einer Programmierschnittstelle bzw. API ist entscheidend für die schnelle Entwicklung und Erweiterbarkeit der Anwendungen, die auf diese Schnittstelle aufsetzen. Möchte man als Anwendungsentwickler bestimmte Funktionalitäten einer API verwenden, so braucht man nur entsprechende Module einbinden. Soll eine Funktionalität hinzukommen, wird einfach zusätzlich das entsprechende Modul eingebunden.

Die grafische Oberfläche (GUI) sollte individuell anpassbar sein. Dies dient vor allem auch der Abgrenzung der einzelnen Hardwarehersteller voneinander. Es ist nämlich nicht mehr unbedingt die Hardware, die ausschlaggebend für den Kauf eines Gerätes ist, sondern eher die individuelle Software. Hierzu ist Symbian sehr gut geeignet, da die GUI völlig unabhängig vom Betriebssystem ist. Es gibt – wie schon gesehen – verschiedene GUI-Varianten: Crystal, Quarz und Pearl. Selbstverständlich sollten die Application Programming Interfaces (APIs) gut dokumentiert und übersichtlich sein.

Ganz wichtig ist die Unterstützung durch Entwicklungs-Tools, damit der Softwareentwickler nicht immer nur auf der Kommandozeile und mit dem Text-Editor arbeiten muss. Tools, welche die Anwendungsentwicklung unter Symbian OS erleichtern, werden im folgenden Abschnitt beschrieben.

1.2.2 Tools

Verschiedene Werkzeuge (Tools) erleichtern die Arbeit bei der Entwicklung von Software für Symbian-OS-Geräte. Diese Helfer treten in verschiedenen Varianten und Erscheinungsformen auf. Es gibt Tools, welche uns Arbeit bei der Erstellung von Basis-Sourcecode abnehmen, Tools, die den Entwicklungszyklus (Build-Prozess) vereinfachen, sowie Tools zur Erstellung der Installationsdateien für das Endgerät. Die wichtigsten benötigten Tools sind die folgenden:

- Microsoft Visual Studio
- Metrowerks CodeWarrior
- Makmake
- Bldmake
- Abl
- Makesis
- Emulator

Das Microsoft Visual Studio (MSVS) C++ sowie der CodeWarrior von Metrowerks sind die von Symbian vorgesehenen IDEs für die Entwicklung von Anwendungen mit dem Series-60-SDK. Diese IDEs erleichtern das Schreiben der Anwendungen durch bekannte Funktionen wie Syntax Highlighting, die Darstellung von Klassen, Methoden und Members in einer Baumstruktur oder die Kompilation des Sourcecodes auf Knopfdruck. Auch das Starten des Emulators direkt aus der Entwicklungsumgebung heraus und das Erstellen neuer Projekte mithilfe des Application Wizard wird von diesen IDEs unterstützt. Makmake unterstützt neben der Generierung von Makefiles für die Win-

*Microsoft Visual Studio
C++ und CodeWarrior*

dows- und ARM-Plattform auch die Erstellung von Projektdateien für die MSVS C++-IDE. Der CodeWarrior benötigt diese Funktionalität von makmake nicht, da er selber aus den ».mmp«-Dateien seine Projektdateien generieren kann.

Bldmake ist ein Tool, welches anhand einer speziellen Informationsdatei (»bld.inf«) die Konfiguration der Build-Skripte (bzw. DOS-Batch-Dateien) vornimmt. In einem Series-60-Projekt wird mit Bldmake eine Datei namens »abld.bat« generiert. Durch Starten dieser »abld.bat«-Datei kann dann der Build-Prozess angestoßen werden. Die »abld.bat«-Datei stößt beim Aufruf dabei wiederum das so genannte ABLD (Automatic Build)-Tool an, welches die Makefiles erzeugt und gleich das Projekt kompiliert.

Es ist auch möglich, die Makefiles mit dem Makmake-Tool zu generieren und danach, wie C-Programmierer es gewohnt sind, mit nmake das Projekt zu kompilieren.

Makesis ist ein Tool, das mithilfe so genannter Package-Dateien (mit der Dateiendung ».pkg«) eines Series-60-Projekts eine Installationsdatei (Dateiendung ».sis«) erzeugt. Die Package-Datei gibt dabei an, wo diese Dateien auf dem Entwicklungssystem liegen und wo sie auf dem Zielgerät installiert werden sollen. Die Installationsdateien müssen nach deren Generierung nur noch auf das Endgerät übertragen werden, um die Anwendung zu installieren. In den ».sis«-Dateien sind die Ressourcdateien und Binärdateien der Anwendung enthalten.

*Emulator zum Testen
und Debuggen*

Der Emulator dient zum Testen und Debuggen der entwickelten Software bevor man sie auf das Endgerät überspielt. Abbildung 1–3 zeigt den Emulator in seiner Standardausführung. Der Emulator ist in zwei Varianten beim Series-60-SDK vorhanden. Einmal als Unicode Debug und einmal als Unicode-Release-Version. Die Debug-Version unterstützt, wie der Name schon sagt, das Debuggen der Programme während der Laufzeit im Emulator. Gestartet wird der Emulator entweder direkt aus der IDE heraus oder durch Aufruf der entsprechenden »epoc.exe«-Datei.

*Basis-Quellcode wird mit
Application Wizard erstellt*

Zum Kennenlernen der oben genannten Tools dienen die in Kapitel 1.3 folgenden Übungen. Weitere Tools findet man nach der Installation des SDK im Verzeichnis »Series60Tools«: Darunter befinden sich der Application Wizard, EpocSwitch, EpocToolbar, MBM-Viewer sowie MmpClick. Der Application Wizard wird in das MS Visual Studio C++-IDE integriert und ermöglicht die einfache Erstellung eines Projekts mit den notwendigsten Klassen und Funktionen. Ähnlich wie die MFC Wizards für die Entwicklung von Windows-Software kann man sich hiermit eine Umgebung für die eigene Anwendung schnell zusammenstricken und dann darauf aufbauend seine Programmlogik implementieren. Abbildung 1–4 zeigt die Oberfläche des Application Wizard.



Abb. 1-3
Der Emulator



Abb. 1-4
Der Application Wizard

EpoSwitch erlaubt das Wechseln zwischen verschiedenen SDK-Versionen. Wenn man also unter Series60 (Pearl) und Series80 (Crystal) entwickelt, so kann man hiermit einfach zwischen den Kompilierungsumgebungen wechseln. Leider wird UIQ (Quartz) von diesem Tool nicht unterstützt, da es im Gegensatz zu den anderen beiden nicht von Nokia entwickelt wurde. EpoToolbar wird wie der Application Wizard auch in das Visual Studio integriert und erweitert dieses um eine Menüleiste, mit der es dann möglich ist, z. B. eine EPOC-Klasse automatisch zu erstellen. Auch ist es damit möglich, direkt für das Zielsystem zu kompilieren oder eine ».pkg«-Datei für das Erstellen

Zahlreiche Tools erleichtern die Arbeit

einer Installationsdatei (».sis«) zu generieren. Der MBMViewer erlaubt es, so genannte ».mbm«-Dateien zu betrachten. ».mbm«-Dateien enthalten die Icons, die auf dem Smartphone für eine Anwendung angezeigt werden. MmpClick ist eine Erweiterung für den Windows Explorer, die es unter anderem erlaubt, mit Rechts-Klick auf eine ».mmp«-Datei ein Projekt für die gewünschte Plattform zu kompilieren.

1.2.3 Quellcode-Kompatibilität

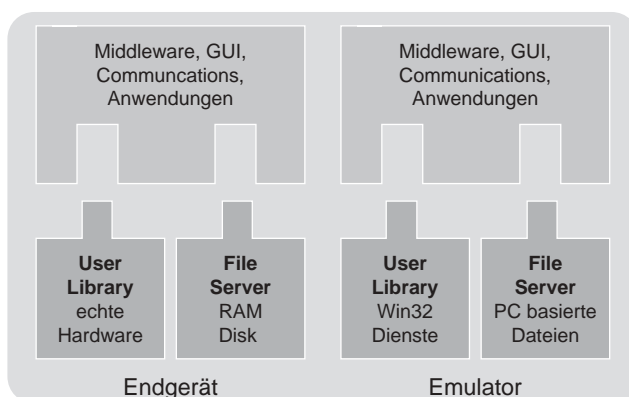
Symbian-Anwendungen werden meist auf einem PC unter Windows oder Linux entwickelt, das Zielsystem ist aber ein Smartphone mit dem Betriebssystem Symbian OS. Wie wird es also ermöglicht, dass derselbe Quellcode auf dem Emulator und auf dem Endgerät läuft, obwohl die Betriebssysteme eigentlich komplett unterschiedlich sind? Warum sind keine Anpassungen mehr zu machen, wenn man das Programm einmal auf dem Emulator und einmal auf dem Endgerät laufen lassen möchte? Wie wird also der Quellcode für die verschiedenen Systeme kompatibel?

Kompatibilität

Die Lösung dieser Fragen ist recht einfach. Die Quellcode-Kompatibilität wird erreicht, indem den Anwendungsprogrammen die Basisdienste des Betriebssystems über die gleichen APIs zur Verfügung gestellt werden. Auf der »echten« EPOC-Maschine sind diese Basisdienste im Kernel, Fileserver und in den Gerätetreibern implementiert und verwenden die echte Hardware. Im Emulator verwenden die Basisdienste die Windows (Win32)- bzw. Linux-APIs, PC-Hardware und das PC-Filesystem. Auf Quellcodeebene sind die APIs aber immer dieselben. Abbildung 1–5 zeigt schematisch das Prinzip der Quellcode-Kompatibilität.

Abb. 1–5

Quellcode-Kompatibilität (nach [Task00])



Die Quellcode-Kompatibilität wird für den größten Teil von EPOC gewährleistet. Anwendungen und Middleware-Bibliotheken sind komplett Quellcode-kompatibel und müssen nur neu kompiliert werden, um sie vom Emulator auf das Zielsystem übertragen zu können. Manche Dinge müssen nicht einmal neu kompiliert werden. Dies ist sehr wichtig für die Flexibilität einer Anwendung. Daten sollten immer implementierungsunabhängig sein. Dies kann man durch die Verwendung von Symbians Stream- und Store-APIs sicherstellen.

Auch die Benutzeroberfläche (GUI) einer Anwendung und die Texte der Anwendung, die ja in verschiedenen Sprachen vorkommen können, sollten nie fest im Quellcode angegeben werden. Dies würde dazu führen, dass bei einer Änderung der Sprache oder der Benutzeroberfläche das Programm neu übersetzt werden müsste. Das ist zeitaufwändig und macht die Anwendung unflexibel. Symbian OS hat für diesen Zweck die Ressourcdateien eingeführt. Wer schon einmal für Windows eine Anwendung mit grafischer Benutzerschnittstelle (GUI) entwickelt hat, kennt das Prinzip. Die Ressourcdateien enthalten alle Informationen über den Aufbau der GUI und die Texte der Anwendung. Sie können in verschiedenen Sprachen vorliegen und müssen nicht neu übersetzt werden, sondern werden nur bei Bedarf von der Anwendung zur Laufzeit geladen. Ressourcdateien sind ein wichtiger Aspekt bei Symbian, auf die in Abschnitt 4.1 noch detaillierter eingegangen wird.

*Ressourcdateien machen
Anwendungen flexibel*

Was auch nicht neu kompiliert werden muss, sind interpretierte Programme wie z. B. Java- oder OPL-Anwendungen. Diese Anwendungen werden von einem so genannten Interpreter bzw. einer Java Virtual Machine ausgeführt. Ein Interpreter ist dabei eine Anwendung, die den Quellcode bzw. den Bytecode bei Java zur Laufzeit interpretiert und ausführt.

1.2.4 Laufwerke

Das Symbian OS ist ein vollständiges Betriebssystem. Wie bei seinen großen Brüdern, den PC-Betriebssystemen, gibt es unter Symbian ein Dateisystem. Wie sieht dieses Dateisystem aus? Wer die Arbeit mit Windows gewohnt ist, wird sich schnell unter Symbian OS orientieren können. Wie unter Windows auch, gibt es hier eine Unterteilung in Laufwerke. Das mag einem auf den ersten Blick etwas komisch vorkommen. Wieso Laufwerke, wenn das Smartphone doch gar keine Festplatte oder kein Diskettenlaufwerk besitzt?

Der Grund für diese Unterteilung ist, dass die EPOC-Entwickler damals offenbar etwas bei Windows abgekupfert haben. Das mag

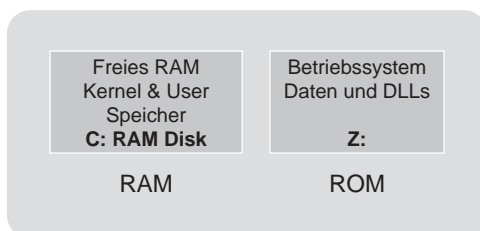
wohl daran liegen, dass sie selbst meist unter Windows entwickelt haben und sich nicht an ein völlig anderes System gewöhnen wollten. Zusätzlich hat diese Unterteilung auch einen praktischen Nutzen. Sie erlaubt eine klare Trennung von verschiedenen Speichermedien in verschiedene Laufwerke. Die Abbildung 1–6 zeigt die Unterteilung der Laufwerke unter Symbian OS.

*Laufwerke trennen
verschiedene
Speichermedien*

Grundsätzlich gibt es unter Symbian OS zwei Laufwerke: das »C:«- und das »Z:«-Laufwerk. Das »Z:«-Laufwerk erlaubt den Zugriff auf das Read Only Memory (ROM). Es enthält einen bootstrap loader und alle ».exe«, ».dll« und andere Dateien, um Symbian OS zu booten und Anwendungen laufen zu lassen. Alle Dateien auf »Z:« sind schreibgeschützt. Programmdateien werden direkt aus dem ROM gestartet.

Abb. 1–6

*Laufwerke
(nach [Task00])*



*RAM ist schnell, aber
auch flüchtig*

Dies ist etwas außergewöhnlich, da bei anderen Betriebssystemen die Anwendungen erst in das Random Access Memory (RAM) geladen und dann ausgeführt werden. Hier sieht man deutlich einen grundsätzlichen Unterschied zu PC-Betriebssystemen, denn diese besitzen im Gegensatz zu einem Smartphone eine Festplatte, auf der die Anwendungsdaten gespeichert sind. Nun sind Festplatten im Vergleich zu RAM-Speichern sehr langsam, haben aber einen entscheidenden Vorteil: Sie behalten die Daten auch dann noch, wenn der Strom wegbleibt. RAM ist ein flüchtiges Speichermedium. Um aber mit den Daten auf einer Festplatte sinnvoll arbeiten zu können, müssen diese aufgrund der Langsamkeit der Festplatte zuerst in das RAM geladen werden.

Mit Anwendungsdaten, die auf dem Smartphone im sehr schnellen ROM liegen, kann dagegen sofort gearbeitet werden. Was Symbian OS ebenfalls von anderen Betriebssystemen unterscheidet, ist der Zugriff auf den RAM-Speicher. Auf das RAM kann unter anderem auch wie auf ein Laufwerk zugegriffen werden. Der Grund dafür ist derselbe wie beim ROM. Es gibt ja keine Festplatte, aber Daten, die während der Laufzeit entstanden sind, wollen irgendwo abgelegt werden. Dies geschieht demnach im RAM.

Der RAM-Speicher ist über das »C:«-Laufwerk zugreifbar und dieses ist das Schreib-Lese-Laufwerk unter Symbian. »C:« enthält Anwendungsdaten, Anwendungs- und System (».ini«)-Dateien und Anwendungen, die vom Benutzer installiert wurden. Früher, bei den ersten EPOC Geräten, war das RAM eine heikle Sache, da es ständig mit Strom versorgt werden musste, um die vorhandenen Daten nicht zu verlieren. Der Strombedarf war zwar sehr gering, aber der Akku des EPOC-Geräts durfte niemals ganz leer laufen. Heutzutage wird dieses Problem mit so genannten Flash-RAM-Bausteinen gelöst. Sie erlauben das Lesen und Schreiben von Daten, kommen aber ohne Strom aus, um diese persistent zu halten. Diese nicht flüchtigen RAM-Bausteine waren früher extrem teuer, sind aber heutzutage Massenware und dadurch sehr günstig.

Eine Festplatte gibt es nicht

Die meisten aktuellen Symbian-Smartphones besitzen daher als Speicher ein ROM, ein RAM und ein Flash-RAM. Im ROM liegen dabei die Betriebssystemdaten und die werkseitig installierten Anwendungen, im Flash-RAM die Benutzerdaten und im RAM werden die Anwendungen ausgeführt. Manche Symbian-Smartphones besitzen die Möglichkeit, den Speicher durch Compact Flash Cards oder Sony Memory Sticks zu erweitern. Wird solch eine Speichererweiterung verwendet, so wird dafür ein weiteres Laufwerk pro Speicherkarte angelegt. Die erste Speicherkarte ist dann über den Laufwerksbuchstaben »D:« erreichbar, die nächste über »E:« usw. Die Laufwerke »Z:« und »C:« werden für den Emulator auf Unterverzeichnisse des Laufwerks gemapped, auf dem das Symbian-SDK installiert wurde. Dies ist aus Sicherheitsgründen so, damit EPOC-Programme nicht einfach irgendwo auf die Windows-»C:«-Platte zugreifen können und weil die meisten PCs sowieso kein »Z:«-Laufwerk besitzen.

Z: und C: werden auf Unterverzeichnisse im Emulator abgebildet

1.3 Übungen: SDK, Application Wizard, Emulator

1.3.1 Allgemeines

In dieser ersten Übung gehen wir näher auf die wichtigsten Tools ein, die das Symbian-SDK mitbringt. Ziel ist es, die Entwicklungsumgebung aufzusetzen, mit dem Application Wizard ein Grundgerüst für eine Anwendung zu erstellen, diese zu kompilieren und im Emulator auszuführen. Damit haben wir dann sehr schnell unsere erste Symbian-OS-Anwendung entwickelt.

Für diese und alle weiteren Übungen sollte das Betriebssystem Windows NT, 2000 oder XP auf Ihrem PC installiert sein. Bei der Installation des SDK unter Windows 2000 oder XP könnte es vorkom-

Windows 98 und älter werden nicht mehr unterstützt

men, dass Sie vom Installationsprogramm gewarnt werden, das SDK sei nicht unter diesem Betriebssystem getestet. Ignorieren Sie diese Meldung. Bis auf ein paar Java-Tools, die sowieso nicht benötigt werden, funktioniert es. Neben dem Windows-Betriebssystem benötigen Sie zusätzlich folgende Software:

- Microsoft Visual Studio 6.0 oder .NET
- für die UIQ-Übungen den Metrowerks Codewarrior
- das Series-60-SDK in der Version 1.0 für die Symbian-OS-Version 6.1 (Nokia 7650, 3650 und N-Gage)
- für die UIQ-Übungen das UIQ-2.0-SDK für die Symbian-OS-Version 7.0 (P800 und P900).

Die IDEs Microsoft Visual Studio und Metrowerks CodeWarrior sind kostenpflichtige Software, die Sie über Ihren Händler beziehen können. Grundsätzlich benötigt man diese Software allerdings nicht, um Anwendungen für Symbian OS zu entwickeln. Sie ist aber recht hilfreich und man sollte nicht darauf verzichten, um sinnvoll arbeiten zu können.

Die meisten Übungen in diesem Buch werden mit dem Microsoft Visual Studio erklärt. Der Grund für diese Entscheidung ist, dass es im Gegensatz zum CodeWarrior viel weiter verbreitet ist und die meisten Leser vermutlich schon eine Version auf ihrem Rechner laufen haben. Der CodeWarrior bietet bessere Integrationsmöglichkeiten für das Symbian-SDK. Beispielsweise unterstützt er direkt das Erstellen einer Projektdatei aus einer Symbian-»`.mmp`«-Datei (siehe Kapitel 2.1.1). Wir werden in diesem Buch aus diesen Gründen auch auf die CodeWarrior-IDE eingehen.

SDKs kosten nichts

Prinzipiell ist es aber gleich, welche IDE Sie verwenden. Die eine bietet nur etwas mehr Komfort als die andere, aber die Übungen laufen in beiden. Die Version 1.0 des Series-60-SDKs gibt es allerdings nur mit Unterstützung der Microsoft-Visual-Studio-IDE [Microsoft]. Die Version 1.2 bietet auch schon Unterstützung für die Metrowerks CodeWarrior-IDE [Metrowerks]. Das Series-60-SDK bekommt man über das Nokia-Forum [NokiaSDK]. Alle Übungsbeispiele finden sich auf der Webseite dieses Buches [SymbÜbg]. Das UIQ-SDK ist über Symbian direkt erhältlich. Alle SDKs werden kostenlos bereitgestellt.

1.3.2 Installation des Series-60-SDK

Bevor wir mit der ersten Übung beginnen, sollten Sie am besten schon das Microsoft Visual Studio 6.0 installiert haben. Falls das nicht der Fall ist, können Sie auch die Teile überblättern, die Visual-Studio-spezifisch sind, und mit einer der späteren Übungen in Kapitel 2 beginnen.

Laden Sie nun zuerst, falls noch nicht geschehen, das Series-60-SDK Version 1.0 herunter, entpacken Sie die gezippte Datei in ein Verzeichnis und starten Sie die Setup.exe. Zum Installieren sollten Sie übrigens Administratorrechte besitzen, da es sonst zu Fehlern bei der Installation kommen kann.

Stimmen Sie bei der Installation dem »Licence Agreement« zu, so werden Sie im nächsten Dialog nach den zu installierenden Komponenten gefragt. Hier wählen Sie alle Symbian-SDK-Dateien aus. Bei den 3rd-Party-Tools brauchen Sie auf jeden Fall, falls noch nicht installiert, Perl.

Einige Bemerkungen zur Java-Laufzeitumgebung (Java Runtime): Java ist nicht unbedingt notwendig, da die Java-Tools, die im SDK mitgeliefert werden, meistens nicht so gut funktionieren. Diese verlangen nämlich unbedingt eine ganz bestimmte Version der Java-Laufzeitumgebung. Ist noch eine andere Java-Laufzeitumgebung auf dem System installiert, kann es Probleme geben.

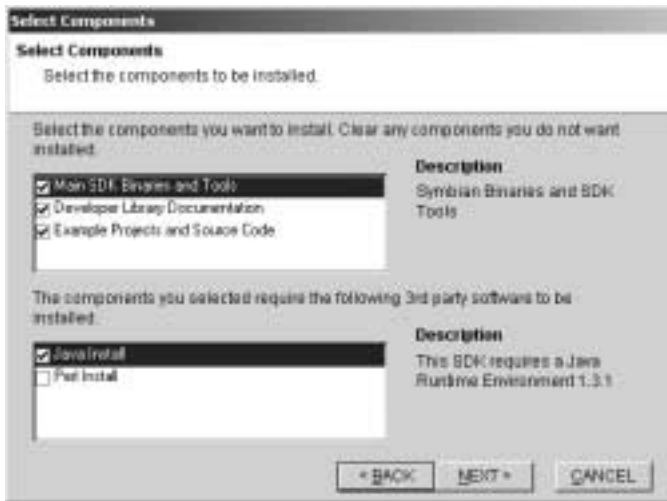


Abb. 1-7
Komponenten-
Installation

Abbildung 1-7 zeigt die Installationsroutine. Für den Installationspfad wählen Sie am besten die Standardeinstellungen. Dies vereinfacht das Nachvollziehen der Übungen und garantiert, dass keine überflüssigen Komplikationen beim Kompilieren auftreten (siehe Abb. 1-8).

Nach der Installation starten Sie den Rechner neu. Die Installationsroutine sollte nun unter »C:\Symbian\6.1\Series60« die in Abbildung 1-9 gezeigten Verzeichnisse angelegt haben.

In den Verzeichnissen »Epo32Ex« und »Series60Ex« findet man die Beispiele des SDK. Die Dokumentation zu den SDK-Beispielen und zum SDK selbst findet man unter »Series60Doc«. Diese werden wir uns später kurz anschauen.

Abb. 1-8
Installationspfad

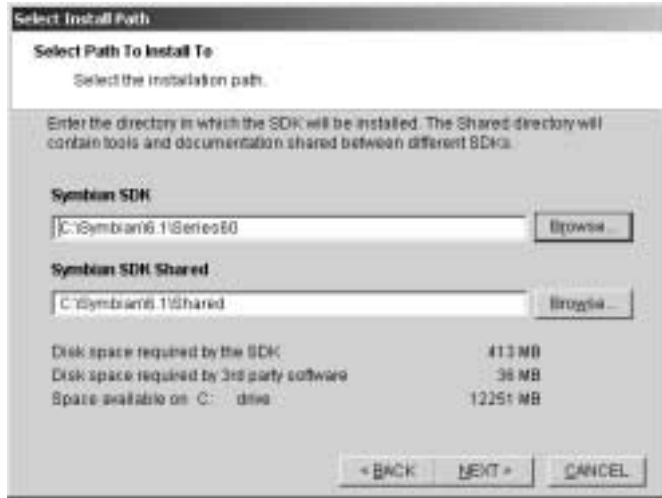


Abb. 1-9
Verzeichnisse



Im »Series60Tools«-Verzeichnis findet man einige nützliche Programme, die in Abschnitt 1.2.2 beschrieben wurden und die die Arbeit zusätzlich erleichtern. Das »Epoc32«-Verzeichnis beinhaltet die SDK-Binärdateien, Konfigurationsdateien und die Emulator-Verzeichnisse, in denen dann die Anwendungen für den Emulator abgelegt werden, d. h. die emulierten »C:«- und »Z:«-Laufwerke. Abbildung 1-10 zeigt die wichtigsten Emulator-Verzeichnisse in der Übersicht.

Da wir später mit dem Application Wizard unsere erste Anwendung erstellen und diese dann gleich im Emulator starten wollen, müssen wir diesen zuerst in das Visual Studio integrieren. Um dies zu tun, begeben Sie sich mit dem Windows Explorer in das Verzeichnis, in dem der Application Wizard liegt: »C:\Symbian\6.1\Series60\Series60Tools\ApplicationWizard«. Hier kopieren Sie die Dateien »AvkonAppWiz.awx« und »AvkonAppWiz.hlp« in das Visual-C++-Template-Verzeichnis: »C:\Programme\Microsoft Visual Studio\Common\MSDev98\Template«.

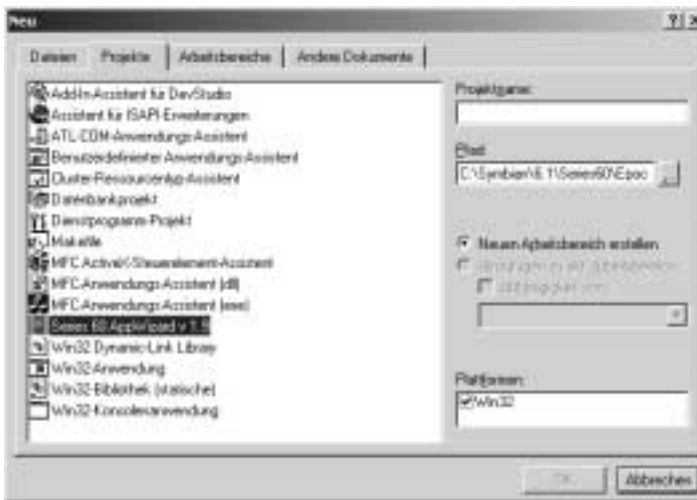
<code>\data</code>	Emulator-Konfigurations-Verzeichnis	Epoc.ini (die Initialisierungsparameter für den Emulator). Epoc.bmp (das Bitmap für den Rahmen und das Hintergrundbild des Emulators).
<code>\Release\wins\UDEB</code>	Emulator-Startverzeichnis	Epoc.exe, das eigentliche Emulator-Programm und alle Shared Libraries (DLLs).
<code>\Release\wins\UDEB\Z\SYSTEM</code>	Emuliertes »Z:«-Laufwerk	Alles, was das EPOC-»Z:«-Laufwerk beinhalten sollte, außer Shared Libraries (DLLs).
<code>Wins\c</code>	Das emulierte »C:«-Laufwerk	Alle Daten. Keine kompilierten C++-Programme, die sind alle auf »Z:«.

Abb. 1–10

Emulator-Verzeichnisse

1.3.3 Anwendungen mit dem Application Wizard

Nun starten sie das Visual-Studio-IDE. Über die Menüpunkte »Datei« und »Neu« gelangen Sie zur Projektauswahl zur Erstellung eines neuen Projektes, wie in Abbildung 1–11 gezeigt.

**Abb. 1–11**

Projektauswahl

Klicken Sie hierzu auf den Reiter »Projekte« und wählen Sie dann das Icon mit der Bezeichnung »Series-60-AppWizard v 1.9«. Geben Sie als Projektnamen »Uebung1« an und wählen Sie für den Pfad folgendes Verzeichnis: »C:\SYMBIAN\6.1\SERIES60\EPOC32EX«.

Unser Projekt wird also in dem Verzeichnis erstellt, in dem die Symbian-Beispiele liegen. Neue Projekte sollten Sie am besten auch immer dort anlegen, zumindest aber in einem Unterverzeichnis von »C:\Symbian\6.1\Series60«, um beim Kompilieren Probleme zu ver-

Neue Projekte im
EPOC32EX- oder
SERIES60EX-Verzeichnis

meiden. Haben Sie nun den Namen und den Pfad angegeben, so klicken Sie auf »OK«. Nun öffnet sich der Application Wizard, wie in Abbildung 1–12 gezeigt.

Das Erstellen einer einfachen Symbian-OS-Anwendung mit grafischer Oberfläche geschieht hier in vier Schritten. Im ersten Schritt geben Sie den Namen Ihrer Anwendung unter »*Application Title*« an. Geben Sie hier z. B. »Uebung1« an. In diesem Dialog kann auch festgelegt werden, was für ein Typ von grafischer Oberfläche erstellt werden soll. Das Simpelste ist die Standardeinstellung *EIKON Control*. Diese unterscheidet sich von einer dialogbasierten in den Klassen, die vom Application Wizard generiert werden. Bei »EIKON Control« wird eine Container-Klasse, welche von der Klasse `CCoeControl` erbt, für die Darstellung der Oberfläche verwendet, bei »Dialog based« wird die Klasse `CEikDialog` als Basisklasse verwendet.

Abb. 1–12

Der Application Wizard für
Schritt 1 von 4



Zu Dialogen und Containern kommen wir aber in Kapitel 4.3. Die Unicode-UID wird Ihnen praktischerweise vom Application Wizard generiert. Diese UID ist sehr wichtig für jede Anwendung, dazu aber später mehr. Belassen Sie das Feld »Unicode UID« einfach wie es ist. Die Unterstützung für INI und Dokumentdateien benötigen wir zuerst auch nicht, machen Sie hier also keinen Haken. Mit INI-Dateien können Anwendungskonfigurationen festgelegt werden. Dokumentdateien dienen dem persistenten Ablegen von Anwendungsdaten. Klicken Sie nun auf »Weiter«. Im folgenden Dialog können Sie noch Copyright und Autor der Anwendung angeben. Wenn Sie erneut auf

»Weiter« klicken, sehen Sie einen Dialog, der auflistet, welche Klassen nun generiert werden sollen (siehe Abb. 1–13).



Abb. 1–13

Der Application Wizard für Schritt 3 von 4

Die einfachste Anwendung mit einer grafischen Oberfläche unter Symbian OS besitzt immer vier Klassen. Eine Anwendungsklasse, über die unsere Anwendung gestartet wird (CUebung1App), zwei Klassen, die für die grafische Oberfläche zuständig sind (CUebung1AppUi und CUebung1Container) und eine Dokument-Klasse (CUebung1Document), die das Datenmodell der Symbian-Anwendung repräsentiert und für die Erstellung der Objekte des User-Interface zuständig ist. Dazu in Kapitel 4.2.4 mehr.

Minimal vier Klassen bei einer Anwendung mit grafischer Oberfläche

Im vierten und letzten Dialog sehen Sie die Verzeichnisse, die automatisch angelegt werden. Das »Include«-Verzeichnis beinhaltet alle Header-Dateien, »Source« enthält den Sourcecode. Das »Group«-Verzeichnis enthält die Symbian-Projektdateien und das »Data«-Verzeichnis enthält die Ressourcen. Das »Test sources«-Verzeichnis ist nur für Testzwecke und wird im Normalfall nicht angelegt (siehe Abb. 1–14).

Es werden noch zwei zusätzliche Verzeichnisse im Projektordner angelegt: »Aif« und »Install«. »Aif« beinhaltet die Anwendungs-Icons und das »Install«-Verzeichnis enthält die ».pkg«-Datei für das Erstellen einer Installationsdatei mit der Endung ».sis«. Nachdem Sie im Application Wizard auf »Fertigstellen« geklickt haben, sollten nun die generierten Klassen im Visual Studio sichtbar sein. Abbildung 1–15 zeigt diese Ansicht.

Abb. 1-14

Der Application Wizard für
Schritt 4 von 4

**Abb. 1-15**

Ansicht der neu
generierten Klassen



1.3.4 Kompilieren und Ausführen der Anwendung

Nun werden wir das Projekt kompilieren. Klicken Sie dazu im oberen Menü auf den »Erstellen«-Button, oder drücken Sie die F7 Taste. Im unteren »Erstellen«-Fenster des Visual Studio sollte sich jetzt etwas bewegen. Am Schluss sollte die Anwendung »UEBUNG1.APP« mit null Fehlern und zwei Warnungen durchkompiliert worden sein. Die zwei Warnungen können Sie getrost ignorieren, da diese nichts direkt mit dem Projekt, sondern mit einer Debug-Library von Symbian zu tun haben. Klicken Sie nun auf das Ausführen-Symbol in der Menüleiste oder drücken Sie CTRL-F5, um die Anwendung im Emulator zu star-

ten. Nun sollte folgender Dialog erscheinen (siehe Abb. 1–16), in dem sie die ausführbare Datei, in diesem Fall »Epoc.exe« angeben müssen.

**Abb. 1–16***Ausführbares Programm*

»Epoc.exe« ist der Emulator, in dem die Anwendung dann gestartet wird. Hier müssen Sie darauf achten, dass Sie die richtige »Epoc.exe«-Datei wählen. Da als Standard im Debug Mode kompiliert wird, müssen Sie nun die Debug-Version des Emulators starten. Diese liegt in folgendem Verzeichnis: »C:\Symbian\6.1\Series60\Epoc32\Release\wins\UDEB\EPOC.EXE«.

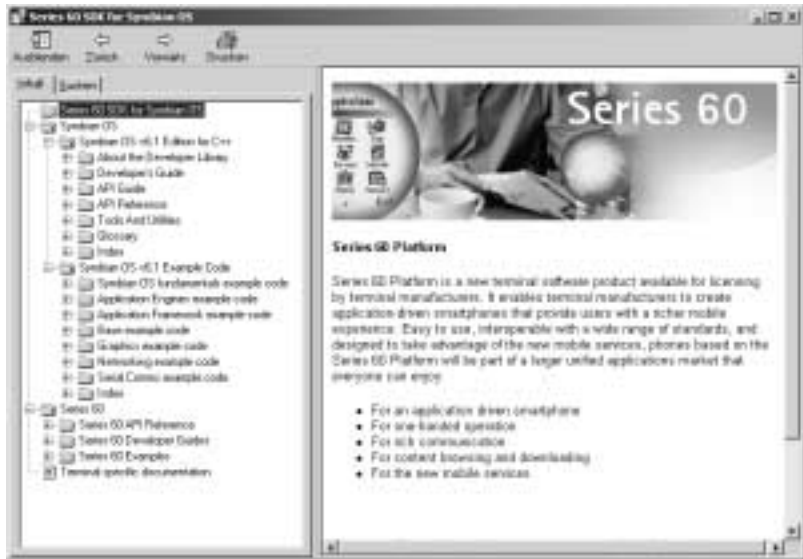
Klicken Sie auf »OK« und der Emulator wird gestartet. Wählen Sie im Emulator aus der Liste der Anwendungen unsere »Uebung1« aus und starten Sie diese. Herzlichen Glückwunsch! Sie haben soeben ihre erste Symbian-OS-Anwendung zum Laufen gebracht! Abbildung 1–17 zeigt das beeindruckende Ergebnis der Arbeit.

**Abb. 1–17***Ergebnis von Übung 1 am Emulator*

1.3.5 Das SDK-Help

Zum Abschluss des ersten Kapitels und der ersten Übung soll noch ein ganz kurzer Überblick über die Hilfe des Symbian-SDKs gegeben werden. Die SDK-Hilfe finden Sie im Startmenü von Windows unter »Programme/Symbian 6.1 SDKs/Series 60/Documentation/SD Help«. Abbildung 1–18 zeigt die SDK-Hilfe nach dem Öffnen.

Abb. 1–18
Die SDK-Hilfe



In der Hilfe gibt es zwei Hauptbereiche: Symbian OS und Series 60. Unter Symbian OS finden Sie die allgemeinen Betriebssystem-API-Beschreibungen und Beispiele. Unter Series 60 finden sie die Beschreibungen der Nokia-Erweiterung von Symbian OS, d.h. der Nokia-spezifischen Klassen. Dort finden Sie auch viele Beispiele zur Programmierung der grafischen Oberfläche und zur Bluetooth-Programmierung. Die Beispiele im Symbian-OS-Bereich der Hilfe beschränken sich dagegen auf die Betriebssystembasis und allgemeine Beispiele zur Grafikprogrammierung.

Schauen Sie sich ruhig einmal in der Hilfe um. Hier finden Sie nützliche Informationen, die über die Beispiele in diesem Buch hinausgehen. Gerade die Series-60-Beispiele sind sehr gut dokumentiert und mit UML-Diagrammen und Screenshots erläutert.