WML und WMLScript
Informationen aufbereiten und präsentieren für WAP-Dienste

Michael Krutwig / Robert Tolksdorf

Reproduktion der Papierausgabe von 2001

Robert Tolksdorf
Bismarckstr. 18
14109 Berlin

Vorwort

Die Definition des *Wireless Application Protocol*, kurz *WAP*, hat die Weichen für das Entstehen eines weltweiten Informationssystems für mobile Endgeräte, insbesondere Handys, gestellt. Ähnlich dem World Wide Web kann man mit WAP Informationen aufbereiten und ansprechend präsentieren. Im Kern der WAP-Protokolle stehen aus Nutzersicht die Auszeichnungssprache *Wireless Markup Language WML* und deren Programmiersprache *WMLScript*. Die weiteren in WAP definierten Dienste zum Transport von Daten, zur Verschlüsselung von Informationen etc. arbeiten hinter den Kulissen und sind eher für Netzanbieter relevant.

WML
WMLScript

In diesem Buch werden Sie in Tiefe WML und WMLScript kennenlernen und einen Überblick auf die dahinterstehende WAP-Technik bekommen. Dabei sind die Beschreibungen für die WAP Version 1.3 gültig – einen kurzen Ausblick auf die Neuerungen in der Folgeversion 1.4 erhalten Sie am Ende des Buchs.

Kein »drahtloses Internet«

Nicht zu verwechseln ist WAP mit einem »drahtlosen Internet« oder »mobilem WWW«. WAP definiert einen eigenen Protokollstapel zur Übertragung von Informationen, der funktional zwar den Internet-Protokollen gleichkommt, aber eben eine andere Technik darstellt. Genauso ist WML keine Untermenge von HTML, sondern eine unabhängig definierte Auszeichnungssprache. WAP sieht allerdings vor, daß über *Gateways* Zugriffe auf die Internet-Welt möglich sind.

WAP Forum

WAP ist die Entwicklung eines Industriekonsortiums, des *WAP Forums*. Es wurde 1997 gegründet und zählt heute ungefähr 500 Mitglieder, die nach eigener Aussage 95% Marktanteil bei mobilen Geräten darstellen. Alle offiziellen Informationen zu WAP einschließlich der technischen Spezifikationsdokumente finden Sie kostenlos im Web unter der Adresse

```
www.wapforum.org
```

Zielgruppe

Dieses Buch wendet sich an Sie als Website-Designer, wenn Sie neue WAP-Seiten erstellen oder bereits vorhandene HTML-Inhalte

für WAP aufbereiten wollen. Es ist aber ebenso gut für Sie geeignet, wenn Sie als privater Nutzer die eigene WAP-Homepage gestalten wollen.

Voraussetzungen

Für dieses Buch setzen wir voraus, daß Sie Erfahrungen mit der Nutzung des Internets haben. Wo es angebracht ist, werden Sie Vergleiche zu den äquivalenten Konzepten und Technologien des Internets im Buch finden.

Aktuelle Informationen

Während ein Buch nicht nachträglich veränderbar ist, lassen sich neuere Entwicklungen und Ergänzungen elektronisch sehr leicht bereitstellen. Schauen Sie daher ab und zu auf der dieser Web-Seite nach:

`www.dpunkt.de/wap`

Sie finden dort interessante Verweise und Ergänzungen zu diesem Buch.

In diesem Buch sind Ausschnitte aus WAP-Seiten oder andere Texte, die in dieser Form in Dateien stehen, in `Schreibmaschinenschrift` dargestellt. Sind Teile davon Platzhalter für andere Texte, wird *`kursive Schreibmaschinenschrift`* verwendet.

Die Abbildungen mit Bildschirmausschnitten wurden zumeist unter Windows auf Handy-Simulatoren erstellt. Wir haben dabei überwiegend das Entwicklungspaket von Nokia verwendet. Es ist durchaus möglich, daß die Beispiele auf Ihrem System etwas von den Abbildungen abweichen.

✘ An vielen Stellen dieses Buchs finden Sie Hinweise, die auf Besonderheiten eingehen. Beispielsweise werden Sie öfter Tips zur Gestaltung Ihrer WML-Seiten antreffen. All diese Stellen sind – wie dieser Absatz – mit dem Zeichen ✘ markiert.

WML 1.2

Die Beschreibung von WML in diesem Buch ist ab der WML Version 1.1 gültig. Die – wenigen – Ergänzungen, die in WML 1.2 hinzukamen, sind wie dieser Absatz markiert.

Am Entstehen eines Buchs sind neben den Autoren viele indirekte Helfer beteiligt. Dank gilt dem Verlag und allen, die durch Bereitstellung von Ressourcen oder durch Hilfestellung und Geduld am Entstehen dieses Buchs mitgewirkt haben.

✘ An dieser Stelle eine Anmerkung zu den in diesem Buch verwendeten Netzadressen, den URLs. Alle diese Verweise wurden vor Drucklegung überprüft und existierten zu dieser Zeit. Es läßt sich aber nicht vermeiden, daß einige Web-Anbieter mit der Zeit ihr Angebot umorganisieren oder einstellen. Dadurch werden natürlich auch URLs ungültig. Ihnen wird dann nichts anderes übrig

bleiben, als ein bestimmtes Programm oder eine Seite mit einer Suchmaschine aufzufinden.

Damit soll es genug der Vorrede sein. Im folgenden Kapitel lernen Sie in einem Schnellstart, wie Sie ein einfaches WAP-Dokument erstellen und online verfügbar machen.

Karlsruhe/Berlin, im Oktober 2000
Michael Krutwig, Robert Tolksdorf

Inhaltsverzeichnis

1 Schnellstart ... 1
1.1 Ein erstes WML-Dokument ... 1
1.2 WML-Dateien veröffentlichen ... 4

I WML 7

2 WML-Grundlagen: Tags und Attribute ... 9
2.1 Tags ... 9
2.2 Attribute ... 12
2.3 Wertetypen für Attribute ... 13

3 Aufbau von WML-Dokumenten ... 15
3.1 Dokumentenvorspann ... 15
3.2 Dokumenteninformationen ... 17
3.3 Karten ... 20

4 Textauszeichnung ... 23
4.1 Zeichendarstellung auf Decks ... 23
4.2 Entitäten ... 24
4.3 Textformatierung ... 26
4.4 Absatzformatierung ... 28
4.5 Links ... 30
4.6 Uninterpretierten Text darstellen ... 35

5 Tabellen ... 37
5.1 Die Tabellenstruktur ... 37

6 Grafiken ... 43
6.1 Grafiken einbinden ... 43
6.2 Grafiken erstellen ... 48
6.3 Vom Web-Logo zum WBMP-Bild ... 49

7 Formulare in WML ... 53
7.1 Auswahlfelder ... 53
7.2 Geschachtelte Auswahllisten ... 58
7.3 Eingabefelder ... 60

8 Ereignisse und Aktionen ... 67
8.1 Steuerelemente ... 67
8.2 Navigationsaktionen ... 70
8.3 Datenfelder an Server übermitteln ... 71
8.4 Übermittlung von Formulareingaben ... 73
8.5 Zurück- und Neudarstellungsaktionen ... 74
8.6 Standardereignisse ... 76
8.7 Zeitereignisse ... 78
8.8 Aktionsschablonen ... 82

9 Variablen in WML-Dokumenten ... 85
9.1 Variablennamen ... 85
9.2 Variablen setzen ... 86
9.3 Variablen verwenden ... 89

10 WML erstellen ... 91
10.1 Entwicklungsumgebung ... 91
10.2 Browser zum Testen ... 94

II WMLScript 99

11 Daten, Datentypen und Ausdrücke ... 101
11.1 Datentypen ... 101
11.1.1 Wahrheitswerte ... 101
11.1.2 Ganzzahlen ... 101
11.1.3 Fließkommazahlen ... 103
11.1.4 Zeichenketten ... 103
11.1.5 Ungültige Werte ... 104
11.2 Typisierung ... 104
11.3 Ausdrücke ... 106
11.3.1 Arithmetische Operatoren ... 107
11.4 Zuweisungen ... 110
11.5 Bindungsregeln ... 110

12 Kontrollstrukturen ... 113
12.1 Anweisungen und Anweisungslisten ... 114
12.2 Zuweisungen ... 115
12.3 Blöcke ... 117

12.4 Kommentare . . . 117
12.5 Variablendeklarationen . . . 117
12.6 Entscheidungen . . . 119
12.7 Schleifen . . . 121

13 Funktionen . . . 127
13.1 Deklaration . . . 127
13.2 Funktionsergebnisse . . . 129
13.3 Funktionsaufruf . . . 129

14 Fehler . . . 133
14.1 Automatisch behobene Fehler . . . 133
14.1.1 Fehler bei arithmetischen Berechnungen . . . 133
14.1.2 Fehler bei Fließkommakonstanten . . . 134
14.1.3 Fehler bei Konvertierungen . . . 135
14.2 Nicht behebbare Fehler . . . 136
14.2.1 Fehler im Programmcode . . . 136
14.2.2 Programmabbrüche . . . 137
14.2.3 Speicherfehler . . . 137

15 Kernbibliotheken in WMLScript . . . 139
15.1 Die `Lang`-Bibliothek . . . 139
15.2 Die `Float`-Bibliothek . . . 145
15.3 Die `String`-Bibliothek . . . 148
15.4 Die `URL`-Bibliothek . . . 156
15.5 Die `WMLBrowser`-Bibliothek . . . 162
15.6 Die `Dialogs`-Bibliothek . . . 164
15.7 Die `Crypto`-Bibliothek . . . 165
15.8 Proprietäre Bibliotheken . . . 166

III WAP 169

16 WAP in der Praxis . . . 171
16.1 Voraussetzungen an WAP-Angebote . . . 171
16.2 WAP-Anwendungen . . . 172
16.2.1 Klassifizierung von WAP-Anwendungen . . . 172
16.2.2 Informationsdienste . . . 174
16.2.3 Auskunftsdienste . . . 174
16.2.4 Finanzdienste . . . 175
16.2.5 m-Commerce . . . 177
16.2.6 Mobilitätsdienste . . . 178

17 Mobilfunknetze – die WAP-Grundlage 181
17.1 Mobilfunknetze, -systeme und Zugriffsverfahren 181
17.1.1 Merkmale von Funknetzen 182
17.1.2 Kanalzugriffsverfahren 184
17.2 Mobilfunk in der ersten Generation 186
17.3 Die zweite Generation des Mobilfunks 187
17.3.1 GSM 188
17.3.2 DECT 192
17.4 Die dritte Generation: UMTS 193

18 Die WAP-Architektur 197
18.1 Was ist WAP? 197
18.2 Das WAP-Forum 199
18.3 Die WAP-Spezifikationen 199
18.4 Die Kommunikation im WAP 200
18.5 Das WAP-Gateway 203
18.6 Push-Dienste 204

19 Die Protokolle des WAP 207
19.1 Grundlegendes zu Protokollschichten 207
19.2 Der WAP-Protokollstapel 210
19.3 Die Transportebene 212
19.3.1 Ausnahmebehandlung durch das WCMP 214
19.4 Sicherheit im WAP 215
19.4.1 Sicherheit auf Transportebene mit WTLS 215
19.4.2 Das Identitätsmodul WIM 221
19.5 Transaktionen im WTP 223
19.5.1 Transaktionsklassen 223
19.5.2 Funktionen des WTP 224
19.5.3 Protokolldienste des WTP 228
19.6 Das Sitzungsmodell 230
19.6.1 WSP als verbindungsloser Dienst 234
19.6.2 Verbindungsorientierte Sitzungen mit dem WSP 236
19.7 Die Anwendungsumgebung WAE 246

20 WAE-Anwendungen 249
20.1 Die Wireless Markup Language WML 249
20.2 Die Skriptsprache WMLScript 251
20.3 Telefoniedienste mittels WTA 256
20.3.1 WTA-Ereignisse 257
20.3.2 WTAI-Aufrufe 259
20.4 Bilder im WBMP-Format 261
20.5 Visitenkarten und Kalender 263

21 Ausblick **265**

A Übersicht der WML-Tags **269**
A.1 Struktur 269
A.2 Inhaltsauszeichnung 269
A.3 Tabellen 270
A.4 Formulare 270
A.5 Tasks 271
A.6 Ereignisse und Variable 271

B Symbolische Werte von Attributen **273**

C Internationale Sprachcodes **275**

D Document Type Definitions **279**
D.1 WML DTD 279
D.2 Channel DTD 287

Literaturverzeichnis **289**

Index **293**

1 Schnellstart

WAP ist eine vergleichsweise neue Technologie. Sie basiert aber auf relativ einfachen Konzepten, so daß man sehr schnell eine erste eigene WML-Seite in das Netz stellen kann. Dieses Kapitel zeigt Ihnen einen solchen »Schnellstart«, für den Sie lediglich einen Texteditor und einen zugänglichen Web-Server benötigen.

1.1 Ein erstes WML-Dokument

Dokumente, die in einem WAP-Gerät dargestellt werden sollen, müssen in einer speziellen Formatierung vorliegen. Sie bestehen aus fortlaufendem Text, in den verschiedene Steueranweisungen – die *Tags* – eingestreut sind. Anders als beispielsweise ein Word-Dokument ist ein solches Dokument in einem reinen ASCII-Format geschrieben.

Falls Sie HTML – die Sprache des Webs ([24]) – schon kennen, wird der Einstieg in WML besonders einfach sein. Die Auszeichnungssprache für WAP-Dokumente benutzt die gleiche syntaktische Struktur mit Tags und Attributen. ✘

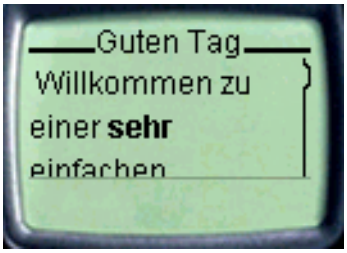
Die Tags umschließen jeweils Abschnitte des fortlaufenden Textes und legen dadurch fest, daß diese Abschnitte beispielsweise besonders dargestellt werden sollen oder eine bestimmte Struktur innerhalb des Dokuments bilden. Die Tags *markieren* also Textteile. Die Menge aller definierten Tags formen zusammen eine Sprache, die *Auszeichnungssprache* (englisch »Markup Language«).

Während man für das Web die Hypertext Markup Language HTML ([24]) zur Auszeichnung verwendet, wurde für den WAP-Dienst eine neue Auszeichnungssprache definiert, die auf die speziellen Gegebenheiten von sehr kleinen und mobilen Geräten wie Handys abgestimmt wurde. Diese Sprache heißt *Wireless Markup Language*, kurz *WML*.

WML wird vom WAP-Forum in einer Spezifikation als Standard festgeschrieben ([44]), die regelmäßig überarbeitet wird. Die

folgenden Kapitel werden Ihnen die Einzelheiten von WML aufzeigen – an dieser Stelle soll ein erstes Beispiel einen Eindruck geben. Abbildung 1.1 zeigt eine sehr einfache Seite, wie sie auf einem Handy dargestellt werden könnte.

Abbildung 1.1
Die Darstellung im Handy



Nicht auf jedem WAP-Gerät muß diese Darstellung identisch sein. So kann die Schriftart von Hersteller zu Hersteller variieren, die Größe des Displays wird unterschiedlich sein und auch bei der Interpretation der Tags gibt es einen gewissen Spielraum für die Darstellung. Abbildung 1.2 zeigt die Beispielseite in einem WAP-Browser auf einem Palm PDA und in Abbildung 1.3 auf der nächsten Seite finden Sie die (offensichtlich fehlerhafte) Darstellung in einem Windows-basierten Microbrowser.

Abbildung 1.2
Die Darstellung auf einem PalmV PDA

In Abbildung 1.4 auf der nächsten Seite sehen Sie den WML-Code für das Dokument. Auf den ersten Blick ist zu erkennen, daß der sichtbare Text »Willkommen zu einer sehr einfachen WML-Seite« von Markierungen in spitzen Klammern umschlossen ist.

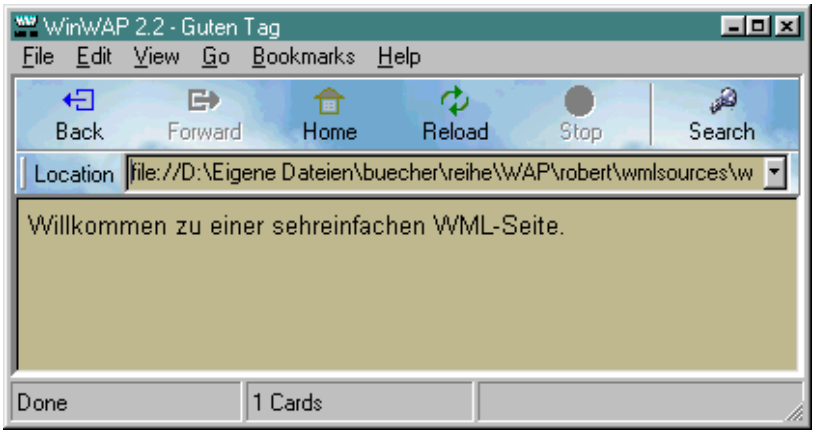


Abbildung 1.3 *Die Darstellung auf einem Windows-basierten Microbrowser*

Das Tag `<em>` (von »emphasized«) zeichnet beispielsweise das Wort »sehr« so aus, daß es in der Darstellung hervorgehoben wird.

```
<?xml version="1.0"?>
<!DOCTYPE wml
 PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="hallo" title="Guten Tag">
    <!-- Nun kommt der eigentliche Text -->
    <p>
      Willkommen zu einer <em>sehr</em>
      einfachen WML-Seite.
    </p>
  </card>
</wml>
```

Abbildung 1.4 *Eine einfache WML-Datei*

Am Beginn des WML-Codes stehen zwei Markierungen (`<?xml` und `<!DOCTYPE`), die zusammen besagen, daß der Inhalt dieser Datei ein WML-Dokument nach der aktuellen Spezifikation des WAP-Forums ist. Es wird danach von der Klammerung `<wml>`... `</wml>` umschlossen.

Jedes WML-Dokument besteht aus einer Reihe von einzelnen Seiten. Die Seiten heißen *Cards* und bilden zusammen als Dokument ein *Deck* (einen »Kartenstapel«). Der Grund für diese Struktur liegt in der vergleichsweise hohen Verzögerung beim Verbin-

dungsaufbau und in der geringen Geschwindigkeit der drahtlosen Datenkommunikation. Um nicht bei jeder Seite eine neue Verbindung aufbauen zu müssen, werden die Cards eines Decks auf einmal und komprimiert zum Handy gefunkt. Dort kann man zwischen ihnen in Ruhe wechseln, ohne weitere Verbindungen aufzubauen.

In dem Beispiel gibt es nur eine einzige Card, die von `<card>` ... `</card>` umschlossen ist. Beim beginnenden `<card>`-Tag finden sich noch weitere Informationen, die in den *Attributen* notiert sind. `id` ist beispielsweise ein Attribut des `<card>`-Tags und enthält als Wert einen eindeutigen Namen, der die Card bezeichnet. Beim `title`-Attribut ist der Wert eine Überschrift der Card, die von den Microbrowsern am Beginn der Darstellung angezeigt wird.

Nach einem Kommentar, der von `<!--...-->` umschlossen ist, folgt der eigentliche Inhalt der Seite: ein einfacher Absatz, der mit dem Tag `<p>` (von »Paragraph«) umschlossen ist. Der kleine Beispielsatz bildet den Inhalt des Absatzes und enthält die oben schon genannte Hervorhebung mit `<em>`.

✘ Mit Vorkenntnissen in HTML zur Auszeichnung von Web-Seiten wird Ihnen das Aussehen von WML-Stapeln nicht ganz ungewohnt erscheinen. Es gibt allerdings eine ganze Reihe subtiler Unterschiede und einige neue Konzepte, so daß Sie WML als eine zweite, neue Sprache ansehen sollten. HTML-Seiten sind nicht als WML-Dokumente verwendbar und können auch nicht direkt konvertiert werden, da sie mehr Möglichkeiten zur Darstellung haben.

1.2 WML-Dateien veröffentlichen

In einem WML-Dokument steht der komplette Inhalt eines Stapels, so wie er von einem WML-Browser verarbeitet werden kann. Tatsächlich erhält ein WAP-Gerät allerdings nicht eine Text-Datei wie oben, sondern binäre Daten, in denen der WML-Text komprimiert dargestellt ist, um Übertragungszeit zu sparen. Die Übersetzung vom textuellen in das binäre Format wird automatisch von Ihrer Entwicklungsumgebung oder dem Betreiber eines WAP-Anschlusses durchgeführt.

Bei einem Web-Server ist in dessen Konfiguration vermerkt, von welcher Art Dokumente mit bestimmten Endungen sind. So gilt üblicherweise, daß Dokument mit den Endungen `.htm` oder

`.html` mit dem sogenannten MIME-Typ `text/html` an einen Browser ausgeliefert oder Bilder mit den Endungen `.jpeg` oder `.jpg` als `image/jpeg` geschickt werden. Mehr zu MIME finden Sie in Abschnitt 20.1 auf Seite 251.

Für WAP-Dienste muß Ihr Web-Server so konfiguriert sein, daß er textuelle WML-Dateien – übliche Endung `.wml` – mit dem Medientypen `text/vnd.wap.wml` und die binäre Form – `.wmlc`-Dateien – als `application/vnd.wap.wmlc` ausliefert. Sollten Bilder im WBMP-Format, das Sie in Kapitel 6.1 auf Seite 43 beschrieben finden, auf Karten enthalten sein, so muß der Server Bilddateien mit der Endung `.wbmp` als `image/vnd.wap.wbmp` übertragen.

Falls Sie einen Web-Server selber betreiben, kennen Sie die dafür notwendigen Einstellungen. Ansonsten sollten Sie zunächst erproben, ob der Server schon so eingestellt ist und ansonsten Ihren Administrator um die notwendige Neukonfiguration bitten.

Ihnen bleibt jetzt nur noch, Ihre WML-Datei auf dem Web-Server abzulegen. Wenn Sie dann im WAP-Gerät die URL wie `http://mein.serv.er/willkommen.wml` angeben, kontaktiert Ihr Gerät über ein Gateway Ihren Web-Server, überträgt den Inhalt der Seite und stellt ihn dar. Ihre erste WML-Seite ist online.

In dieser Form können Sie praktisch alle WML-Konzepte, die Sie in den folgenden Kapiteln dieses ersten Teils des Buchs kennenlernen, umsetzen. Die Darstellung von WML beginnt mit der Beschreibung der Struktur von WML-Stapeln, zeigt Ihnen die Möglichkeiten zur Textauszeichnung und für Verweise auf andere Stapel mit Hyperlinks. Für komplexere Anwendungen lernen Sie die WML-Unterstützung für Grafiken, Tabellen und Formulare kennen. Abschließend geht es um die fortgeschrittenen Konzepte wie den Umgang mit Ereignissen, Aktionen und Variablen. Vor dem Ende des WML-Teils erhalten Sie noch einen Überblick auf die Funktionalitäten von WML-Entwicklungsumgebungen und sehen einige Beispiele.

Teil I

WML

2 WML-Grundlagen: Tags und Attribute

Die *Wireless Markup Language*, WML, ist die Sprache, mit der WAP-Inhalte gestaltet werden. In diesem Kapitel lernen Sie dazu die Grundlagen von Auszeichnungssprachen kennen.

2.1 Tags

Ein Dokument, das Sie mit WAP laden können, ist auf einem Server im Netz gespeichert und wird zum WAP-Gerät mithilfe verschiedener Protokolle geeignet transportiert. WML ist das Format, in dem diese Dokumente gespeichert sind und das ein Informationsanbieter herstellen muß.

Anders als beispielsweise ein Word-Dokument, das nicht als Klartext lesbar ist, verwenden WML-Seiten lediglich einen normalen Textzeichensatz und können so mit einem beliebigen Editor erstellt werden. So könnten Sie selbst mit dem spartanischen `edit` Kommando unter MS-DOS eine WML-Seite erstellen. Tatsächlich wird man natürlich komfortablere Editoren verwenden und für größere Dokumentenbestände eine WML-Entwicklungsumgebung wählen.

WML ist eine Auszeichnungssprache (englisch »Markup Language«), die auf dem XML-Standard ([56]) beruht. Damit gehört sie zu der Sprachenfamilie, die aus der Weiterentwicklung von HTML, der Sprache zur Markierung von Web-Dokumenten ([24]), entstanden ist. XML erlaubt die Definition von anwendungsspezifischen Auszeichnungssprachen und WML ist speziell auf die Verwendung in drahtlosen Netzen mit beschränkter Bandbreite und eingeschränkten Darstellungsmöglichkeiten in den Anzeigegeräten hin konzipiert.

Eine XML-Auszeichnungssprache definiert eine feste Menge von Auszeichnungselementen – die *Tags* oder *Elemente*. Hinzu kommen Regeln über deren Zusammensetzung – die Grammatik

der Sprache. In einem Dokument werden diese Auszeichnungselemente mit dem Inhaltstext kombiniert. Alle Tags sind dazu von spitzen Klammern umschlossen, wie beispielsweise in

```
Kontostand:<br/>3013
```

`<br/>` ist hier das WML-Element, das Wort »Kontostand:« und die Zahl »3013« gehören zum Inhalt. In der WML-Spezifikation wird nun festgelegt, daß ein WAP-Gerät bei der Darstellung für `<br/>` einen Zeilenumbruch einfügen soll.

Tags können auch Inhalt umschließen – in diesem Fall gibt es ein Start-Tag und ein Ende-Tag, die den gleichen Namen tragen. Ein Beispiel ist

```
Kontostand:<br/><u>-120</u>
```

Die Markierung der Zahl beginnt mit dem Start-Tag `<u>` und endet mit dem Ende-Tag `</u>`. Im WML-Standard steht, daß so markierter Text unterstrichen darzustellen ist.

In WML ist eine feste Menge von 35 Tagnamen definiert, die Sie in den nächsten Kapiteln kennenlernen. Dabei ist die Groß- und Kleinschreibung der Namen relevant und alle WML-Tags sind komplett mit Kleinbuchstaben definiert.

✘ Während in HTML beispielsweise die Tags `<Table>`, `<TABLE>` und `<TaBlE>` als identisch angesehen werden und den Beginn einer Tabelle markieren, ist in WML dafür ausschließlich `<table>` definiert.

Umschließt ein Tag-Paar keinerlei Inhalt, kann man dafür die Abkürzung verwenden. So beschreibt `<td>...</td>` beispielsweise eine Tabellenzelle. Wollen Sie eine leere Zelle markieren, können Sie `<td></td>` als `<td/>` abkürzen.

✘ In HTML gibt es diese abgekürzten Schreibweisen nicht – die XML-Sprachen erweitern den Umgang mit Tags an dieser Stelle.

Das obige Beispiel `<br/>` ist allerdings nicht nur eine Abkürzung – welchen Inhalt sollte man auch als Zeilenumbruch markieren? Tags, für die kein Inhalt vorgesehen ist, müssen also immer in der Kurzform notiert werden – `<br></br>` gibt es nicht in WML.

XML-Sprachen kennen den Begriff der *Wohlgeformtheit*, dessen Einhaltung von korrekten WML-Dokumenten gefordert wird. Er umfaßt drei Bedingungen:

1. Zu jedem Start-Tag muß es ein passendes Ende-Tag geben. So muß jeder Absatz in einem WML-Dokument mit `<p>` eingeleitet werden, aber auch von `</p>` abgeschlossen sein.

   In HTML gibt es diese strenge Forderung nicht – `<p>` markiert dort nur den *Beginn* eines neuen Absatzes. ✘

2. Tags müssen korrekt geschachtelt sein. Eine Markierung folgender Art ist nicht erlaubt:

   ```
   <p><u>Bitte beachten Sie die
   Geschäftsbedingungen.</p><p>Die
   Haftung liegt bei Ihnen.</u></p>
   ```

   Um beide Absätze zu unterstreichen, müssen Sie sie jeweils einzeln als unterstrichen markieren:

   ```
   <p><u>Bitte beachten Sie die
   Geschäftsbedingungen.</u></p><p><u>Die
   Haftung liegt bei Ihnen.</u></p>
   ```

   In HTML gilt diese Regel eigentlich auch. Allerdings sind vorhandene HTML-Browser so programmiert, daß sie auch ungültige Dokumente um fast jeden Preis darstellen und auch über eine falsche Schachtelung hinwegsehen. ✘

3. Ein Dokument muß genau ein äußerstes Tag haben – in WML ist dies `<wml>`. Es darf nur einmalig auftreten und von keinem anderen Tag umschlossen sein.

Erfüllt ein WML-Dokument diese Kriterien nicht, kann das WAP-Gerät die Darstellung verweigern.

An den Beispielen erkennen Sie auch, daß die Formatierung des WML-Quelltextes keine Rolle spielt – schließlich entscheidet sich die Darstellung ja erst im WAP-Gerät.

In einem Aspekt ist die Formatierung des Quellcodes allerdings nicht egal. Da ein Zeilenumbruch als ein Leerzeichen gilt, ist es wichtig, ob man nach einem Start-Tag eine neue Zeile im Quellcode beginnt. Schreibt man also: ✘

```
<u>
Bitte beachten Sie...
```

dann befindet sich vor dem »Bitte« ein Leerzeichen und dieses kann bei der Darstellung mit unterstrichen werden: » Bitte«. Um die korrekte Darstellung »Bitte« zu erreichen gibt es zwei Möglichkeiten, `<u>Bitte beachten Sie` ist die naheliegende Variante davon. Muß das Tag unbedingt in einer einzelnen Zeile stehen, fügen Sie den Zeilenumbruch nach dem Tag-Namen ein:

```
<u
>Bitte beachten Sie...
```

Dieses Verfahren nutzt die Regel, daß nach dem Elementnamen und vor dem Tag-Abschluß mit `>` oder `/>` beliebig viel Leerraum, also auch ein Zeilenumbruch, auftreten darf.

`<!-- -->`

Neben den normalen Tags, die der WML-Definition entsprechen müssen, können Sie in Ihre Dokumente auch Kommentare einfügen. Sie haben immer die Form

```
<!-- Kommentartext -->
```

Die Kommentare können sich über mehrere Zeilen erstrecken und auch andere Tags einschließen. Es ist allerdings nicht möglich, Kommentare zu schachteln.

2.2 Attribute

Bei der Markierung von Inhalt durch Tags kann man noch weitere Eigenschaften der gewünschten Darstellung etc. notieren. Dies geschieht in *Attributen*, die innerhalb des Start-Tags nach dem Elementnamen notiert werden. Für die Auszeichnung von Absätzen mit `<p>` ist es beispielsweise möglich, in einem Attribut die Ausrichtung des Absatzes bei der Darstellung festzulegen. Um alle Zeilen zu zentrieren, können Sie folgendes schreiben:

```
<p align="center">Schalterstunden<br>
Mo-Fr: 9-18 Uhr<br>
Am Wochende geschlossen</p>
```

Die Ausrichtung der Darstellung ist im Attribut `align` vermerkt. Im Beispiel enthält das Attribut den Wert `center`, indem man ihn hinter dem Attributnamen von einem `=` getrennt hinschreibt.

Attributwerte müssen in WML – wie bei jeder XML-basierten Auszeichnungssprache – immer von doppelten oder einfachen Hochkommas eingeschlossen sein und jedes Attribut muß immer einen

Wert tragen. Um ein doppeltes Hochkomma in einem Attributwert zu notieren, muß das Attribut von einfachen Anführungszeichen umschlossen sein, also wie bei `title='Eine "tolle" Seite'`. Innerhalb von doppelten Anführungszeichen können Sie ohne Probleme das einfache verwenden: `title="Aristoteles' Schriften"`. In den Beispielen in diesem Buch verwenden wir übrigens immer die Form mit doppelten Hochkommata.

Die WML-Spezifikation definiert, welche Attribute für die einzelnen Tags möglich sind und welche Werte wie bei der Darstellung beachtet werden sollen.

Anders als in HTML ist in WML bei den Attributnamen die Groß- und Kleinschreibung relevant. In WML sind alle Attribute nur in Kleinschreibung definiert. ✘

Zu einem WML-Tag können mehrere Attribute definiert sein, sie werden dann einfach nacheinander geschrieben: `<p align="center" mode="nowrap">`. Es gibt keinerlei Bedingungen für die Reihenfolge dieser Attribute. Bei einigen Attributen kann die WML-Definition ihr Auftreten verlangen; so muß beispielsweise beim Tag `<img>` das die Darstellung einer Grafik bewirkt, immer das Attribut `src` verwendet werden, das die Netzadresse der Grafikdatei enthält. Bei einigen Tags ist in der WML-Spezifikation vermerkt, daß sich die Verwendung bestimmter Attribute gegenseitig ausschließt.

2.3 Wertetypen für Attribute

In WML sind Attribute immer Zeichenketten. Um dennoch genauere Aussagen über die Wertebereiche für Attribute machen zu können, legt man in WML bestimmte Arten von Attributen fest. Die wichtigsten unterscheidbaren Attributtypen in WML sind:

- *Zahlen* sind positive ganze Zahlen wie `50`.
- *Längen* beschreiben Ausmaße von Seitenbestandteilen. Sie werden durch positive ganze Zahlen wie `50` ausgedrückt. Die Einheit dieser Länge ist dann die Anzahl von Bildpunkten auf dem Ausgabegerät. Längen können auch relativ als Prozentwert wie `50%` angegeben werden, dann repräsentieren sie einen Anteil am umschließenden Darstellungselement.
- *URLs* sind Verweise auf andere Ressourcen im Netz. Sie kennen den prinzipiellen Aufbau mit Sicherheit von der Web-

Nutzung, die formale Definition, wie URLs aussehen, finden Sie im RFC-Standard 2396 ([4]).

- *Schalter* enthalten ein logisches wahr oder falsch. Die beiden möglichen Werte sind `true` für wahr und `false` für falsch.
- *Token* sind vordefinierte symbolische Werte, wie beispielweise `center` des `align`-Attributs beim `<p>`-Tag. Je nach Attribut sind andere Werte vorgesehen.

Hinzu kommen normale Zeichenketten, beispielsweise für Seitenüberschriften. Sie enthalten normalen Text ohne weitere Auszeichnungselemente.

✘ Der Grund für die Repräsentation aller Werte als normale Zeichenketten liegt darin, daß WML auf XML-Technologie basiert. Mit ihr ist eine Festlegung wie »Attribut `width` muß eine Ganzzahl enthalten« – also eine Typisierung wie in einer Programmiersprache – nicht möglich.

Mit den Grundkonzepten aus diesem Kapitel ist die syntaktische Basis für WML gelegt. Im folgenden Kapitel lernen Sie die ersten WML-Tags kennen, nämlich die zur Beschreibung der Struktur von WML-Seiten.

3 Aufbau von WML-Dokumenten

Ein WML-Dokument ist eine Sammlung von WML-Seiten, die zusammen an das WAP-Gerät übertragen werden. Jede einzelne Seite nennt man *Card* (»Karte«), das gesamte Dokument *Deck* (»Kartenstapel«). WML legt die Struktur der Auszeichnung von Decks fest und definiert die dabei verwendbaren Tags.

Abbildung 3.1 zeigt die immer gleiche Struktur eines WML-Dokuments. Der Vorspann markiert das Dokument als WML-Text. Es folgen Dokumenteninformationen und Schablonen und schließlich der eigentliche Inhalt, der in einzelne Cards unterteilt ist.

```
<?xml version="1.0"?>
<!DOCTYPE wml
 PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <head>Dokumenteninformationen</head>
  <template>Schablonen</template>
    <card>      <card>      <card>      <card>
```


```
    </card>     </card>     </card>     </card>
</wml>
```

Abbildung 3.1
Die Struktur eines WML-Dokuments

3.1 Dokumentenvorspann

Der Dokumentenvorspann ist bei jedem korrekten Deck notwendig. Er zeichnet es als ein XML-Dokument aus, das der WML-

Spezifikation folgt. Wenn Sie eine WML-Entwicklungsumgebung benutzen, wird er wahrscheinlich automatisch generiert. Die folgenden Zeilen sind am Beginn eines Stapels notwendig:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
```

<!DOCTYPE>

Sie besagen zweierlei: Mit `<?xml` zeigt man an, daß es sich um ein XML-Dokument handelt; mit `<!DOCTYPE>`, daß das Dokument der für WML geltenden Grammatik folgt und das umschließende Tag `<wml>` ist. Diese beiden Zeilen sind immer konstant und können auch automatisch von Ihrer WML-Entwicklungsumgebung erzeugt werden.

<wml>

Direkt nach dieser Einleitung muß das WML-Dokument beginnen. Es wird von dem Tag `<wml>` umschlossen. In einem Dokument kann und muß es genau ein solches Tag geben.

Das Tag `<wml>` kann mehrere Attribute tragen. Zwei davon sind die *Standardattribute*, nämlich `id` und `class`. Sie sind bei jedem WML-Tag verwendbar und sind für die automatische Verarbeitung von WML-Dokumenten und für die Navigation zwischen Karten gedacht. Sie haben keine Wirkung auf die Darstellung von Inhalten im WAP-Gerät oder die Interaktion mit den Nutzern.

`id` (von »identifier«) dient der eindeutigen Identifizierung einer Stelle im WML-Dokument, nämlich des Tags, das das Attribut trägt. Sein Wert ist eine Zeichenkette, die für alle `id`-Attribute im gleichen Kartenstapel eindeutig sein muß.

`class` erlaubt es, Elemente einer Klasse zuzuweisen, indem als Wert ein Name angegeben wird. Damit fassen Klassen logisch verschiedene Bestandteile eines Dokuments zusammen. Ein Element kann zu mehreren Klassen gehören, in diesem Fall enthält das Attribut mehrere Klassennamen, die durch Leerzeichen getrennt sind.

Während diese beiden Standardattribute keinerlei sichtbaren Auswirkungen haben, gibt `xml:lang` an, in welcher Sprache das Dokument geschrieben ist. Der Microbrowser kann daraufhin die Schreibrichtung oder Trennmuster korrekt wählen. Der Wert des Attributs ist ein Sprachcode, der aus zwei Buchstaben besteht. In Anhang C auf Seite 275 finden Sie eine Übersicht der Codes, die international standardisiert ist. Für deutschsprachige Dokumente sollten Sie also `xml:lang="de"` verwenden.

`xml:lang` ist für die WML-Elemente vorgesehen, die Text enthalten können oder deren Darstellung sprachabhängig sein kann.

Auf einer Seite können Sie somit verschiedene Sprachen mischen, indem Sie jeweils das richtige Sprachkürzel angeben.

Ist bei einem Tag kein `xml:lang`-Attribut vorhanden, werden die umschließenden Elemente – bis zu `<wml>` – danach durchsucht. Ist dort nichts angegeben, versucht das Gerät aus den Übertragungsdaten – wie der HTTP-Information `Content-Language` – die Sprache zu ermitteln. Ist auch das nicht möglich, gilt die in den Einstellungen des Geräts ausgewählte Sprache.

3.2 Dokumenteninformationen

<head>

Das Tag `<head>` umschließt die Dokumenteninformationen. Es darf nur einmal pro WML-Dokument auftreten und muß das erste Element nach dem öffnenden `<wml>` sein. Bei `<head>` sind lediglich die beiden Standardattribute `id` und `class` möglich.

Innerhalb von `<head>...</head>` können zwei Elemente auftreten, die Informationen über das Dokument enthalten und eine einfache Art des Zugriffsschutzes realisieren.

Ein WML-Dokument enthält Informationen, die im Browser dargestellt werden. Neben diesem Inhalt gibt es weitere Informationen, die das Dokument selber beschreiben und beispielsweise angeben, wer der Autor ist. Diese Informationen über Informationen nennt man *Metainformationen*.

Metainformationen sind einer der Schlüssel zur effektiven Informationserschließung in Netzen ([25]). Wenn Sie zum Auffinden von Web-Informationen eine Suchmaschine benutzen, geben Sie üblicherweise Suchworte an, die innerhalb gefundener Seiten auftreten. Könnte man zuverlässig Metainformationen bei Dokumenten finden oder extrahieren, wären Suchergebnisse erheblich präziser. Sie könnten dann nämlich auf der Metainformation »Dieses Dokument handelt von drahtlosen Informationssystemen« arbeiten und sich nicht auf das Auftreten der Zeichenketten »WAP« oder »WML« in der Information selbst beschränken.

Zur Notation von Metainformationen in Dokumenten sind zwei Dinge erforderlich. Einmal muß Metainformation als solche markiert werden und zum anderen muß es ein Schema geben, das die Art der Information und deren Interpretation definiert. So wäre zu standardisieren, ob der Autor eines Dokuments in der Metainformation »Autor«, »author« oder »Creator« vermerkt ist.

Für die Standardisierung eines solchen Schemas für Metadaten hat in den letzten Jahren die Arbeit des Konsortiums *Dublin Core Metadata Initiative* (siehe `http://purl.org/DC`) große Be-

deutung gewonnen. Der »Dublin Core« definiert eine kleine Menge von Metainformationen zu Dokumenten, bestimmt einen Namen dafür und beschreibt die Bedeutung. So enthält nach Dublin Core die Metainformation `Creator` den Namen des Erstellers des Inhalts einer Seite, der eine Person oder eine Organisation sein kann. Langsam, aber doch zunehmend werden Informationen im Web nach diesem Schema markiert und der Dublin Core wird auch im WAP seine Bedeutung erlangen.

<meta>

WML stellt für die Markierung von Metainformation das `<meta>`-Tag bereit. Neben den beiden Standardattributen sind die folgenden Attribute möglich:

- `schema` benennt das verwendete Metadatenschema, mit dessen Hilfe die Datenfelder und ihre Inhalte interpretiert werden sollen.
- `name` gibt den Namen der Metainformation an.
- Das auf jeden Fall zu verwendende `content` enthält die eigentliche Metainformation. Sie soll unter dem Namen in `name` mithilfe des Schemas aus `schema` interpretiert werden.

Gäbe es dieses Buch in einer WML-Version, sollte jeder Kartenstapel darin die folgenden Zeilen enthalten:

```
<meta schema="http://purl.org/DC"
 name="Creator"
 content="Robert Tolksdorf, Michael Krutwig">
```

✘ Sie sollten das `<meta>`-Tag in Ihren Informationsinhalten verwenden. Als Metadatenschema sollten Sie sich momentan an Dublin Core halten. Sie können damit rechnen, daß Ihre WML-Seiten auf diese Weise präziser in Suchmaschinen auffindbar sind.

Neben dieser informationsorientierten Sicht auf Metadaten gibt es noch eine protokollorientierte. Bei der Übertragung von WML-Kartenstapeln im Netz unterscheidet das verwendete Protokoll zwischen dem eigentlichen Inhalt und den sogenannten *Header*, die beispielsweise angeben, wieviele Bytes im Inhalt folgen.

Mit `<meta>` kann man solche Header-Zeilen zusätzlich in das Protokoll einfügen, indem man anstelle von `name` das Attribut `http-equiv` verwendet. In diesem Fall soll der Server den Inhalt von `content` in einer solchen Protokollzeile versenden.

Es kann beispielsweise sein, daß Kartenstapel auf dem Weg vom Ursprungsserver zum Gateway in einem Cache zwischengespeichert werden. Ändert sich nun die Ursprungsinformation,

sind diese Kopien veraltet. Man kann dem Cache mitteilen, daß er zu einem bestimmten Zeitpunkt eine Kopie verwerfen soll, indem man ihm diese Metainformation mitteilt. Dies geschieht über das Transportprotokoll und eine zusätzliche Zeile, die mit `Expires:` beginnt und ein Datum enthält. Eine solche Zeile können Sie mit dem `<meta>`-Tag so erzeugen:

```
<meta http-equiv="Expires"
 content="Mon 1 Jan 2001 00:00:00 GMT">
```

Es kann sein, daß man möchte, daß diese Information nicht zum WAP-Gerät gelangt. Man kann dies verhindern, indem man das Attribut `forua` (von »for user agent« – »für Endgerät«) mit einem Wert `false` verwendet. Trägt es hingegen den Wert `true`, muß die eingefügte Protokollzeile bis zum Endgerät transportiert werden.

<access>

Mit `<access>`, dem zweiten möglichen Tag innerhalb der Markierung `<head>...</head>`, können Sie festlegen, von welchen anderen Karten zu Ihrem Kartenstapel navigiert werden darf.

Dessen Attribut `domain` enthält ein Fragment einer Internet-Adresse, auf das die Netzadresse aller Karten enden muß, die auf die aktuelle Karte verweisen wollen. `path` enthält den Beginn eines Pfades, unter dem die Karte auf dem entsprechenden Server abgelegt sein muß.

Die Tabelle in Abbildung 3.2 macht dies anhand eines Beispiels deutlicher. Die jeweils unterstrichenen Teile der Netzadresse passen zu dem Muster, das durch `domain="wap.org"` und `path="/info"` vorgegeben ist. Damit ein Zugriff erlaubt wird, müssen beide Komponenten passen.

domain="wap.org" path="/info"	Erlaubt
http://wap.org/info	ja
http://wap.org/	nein
http://wup.org/info	nein
http://www.wap.org/info/printer	ja
http://wupwap.org/info	nein
http://wap.org/information	nein

Abbildung 3.2
domain und path bei <access>

Beim Vergleich werden die Trennzeichen . und / in Netzadressen und Pfaden beachtet und die Bestandteile darin jeweils komplett verglichen, wie die letzten beiden Beispiele zeigen.

✘ Mit WMLScript stehen Ihnen Bibliotheken zur Verfügung, mit denen Sie eine URL in diese Bestandteile per Programm zerlegen können. Sie finden die Details dazu in Abschnitt 15.4 auf Seite 156.

Fehlen `domain` oder `path`, gelten als Standardwerte die Netzadresse des aktuellen Stapels und der Pfad /. `<access>` kann natürlich auch die beiden Standardattribute `id` und `class` tragen.

Das nach `<head>` mögliche Tag `<template>` finden Sie in Abschnitt 8.8 auf Seite 82 beschrieben. Mit ihm können für alle Karten eines Stapels bestimmte Reaktionen auf Eingabeereignisse definiert werden. An dieser Stelle beschäftigen wir uns aber zunächst nur mit dem Aufbau von Kartenstapeln.

3.3 Karten

`<card>`

Jede Karte innerhalb eines Stapels wird mit `<card>` umschlossen. Die Attribute des Tags enthalten Informationen über die Karte.

Mit `title` legen Sie einen Titel für die Karte fest. Er kann geeignet vom Microbrowser dargestellt werden und beispielsweise auch in Bookmark-Listen als Kartenbeschreibung verwendet werden.

✘ Wollen Sie die Darstellung eines Titelelements verhindern und so etwas mehr Platz für Ihre Karte schaffen, kann `title=""` helfen. Gleichzeitig hat Ihre Karte aber eben keinen Titel mehr und könnte in einer Bookmark-Liste dadurch schlecht auffindbar sein.

Mit `xml:lang` bestimmen Sie wieder die Sprache des Inhalts der Karte. Sein Wert überlagert die Sprachfestlegung, die vielleicht bei `<wml>` getroffen wurde.

Das Standardattribut `id` ist bei Karten besonders wichtig, weil es als Zielname für die Navigation zu einer bestimmten Karte dient. Beachten Sie, daß `id` innerhalb eines Stapels eindeutig sein muß. Das zweite Standardattribut `class` hat wieder keine Auswirkung auf die Sichtbarkeit.

Wie in Kapitel 9 auf Seite 85 beschrieben, kann das WAP-Gerät Variablen speichern, die durch Nutzereingaben oder andere Wege verändert werden können. Zusammen mit einer Nutzungsgeschichte bilden sie den sogenannten *Kontext* des Browsers. Falls Sie nun eigene Werte in Variablen schreiben, kann es sein, daß schon vorhandene Werte die Nutzung Ihrer Seiten verhindern oder stören.

Das Attribut `newcontext` kann dies verhindern. Trägt es den Wert `true`, dann soll mit der Darstellung dieser Karte der Kontext gelöscht werden. Damit gelten alle Variablen als ungesetzt. Der Wert `false` löscht den Kontext nicht, was gleichzeitig das Standardverhalten ist.

Schließlich können Sie dem Browser noch einen Hinweis geben, ob der Inhalt der Karte in seiner textuellen Reihenfolge dargestellt werden muß, oder ob es völlig dem Browser überlassen bleibt, die Felder zu ordnen. Ein Browser könnte diese Erlaubnis nutzen, um mit der Zeit häufig benutzte Eingabefelder an den Anfang der Darstellung umzusortieren. Das entsprechende Attribut heißt `ordered` und verlangt beim Wert `true` die Beibehaltung der Reihenfolge; bei `false` bleibt dem Browser die Entscheidung darüber überlassen.

Die drei weiteren möglichen Attribute von `<card>` `onenterforward`, `onenterbackward` und `ontimer` steuern Reaktionen auf Eingabeereignisse während der Darstellung der Karte und sind in Abschnitt 8.8 auf Seite 82 beschrieben.

Innerhalb einer Karte können Sie noch Unterstrukturen markieren, die ein Microbrowser vielleicht zur Optimierung der Darstellung verwendet. Logisch zusammengehörige Felder und Text lassen sich jeweils mit `<fieldset>` umfassen. Der Browser könnte sie beispielsweise mit den Ihnen vielleicht von PC-Nutzeroberflächen her bekannten Registrierkarten darstellen.

<fieldset>

Das Attribut `title` enthält einen beschreibenden Titel für eine solche Inhaltsgruppe. `xml:lang` bestimmt wieder die Sprache des Inhalts und schließlich können Sie auch die Standardattribute `id` und `class` verwenden.

✘ WAP-Geräte limitieren die Größe verarbeitbarer Stapel, da sie bei Speichermangel Daten nicht auf eine Festplatte auslagern können wie ein PC. Die Begrenzung bezieht sich dabei nicht auf den WML-Quellcode, da dieser ja noch in ein kompakteres Format übersetzt wird (siehe Abschnitt 18.4 auf Seite 200). Für dieses Kompilat gilt beispielsweise bei einem Nokia 7110 eine Maximalgröße von 1397 Bytes, während ein Ericsson MC218 mehr als 8000 Bytes akzeptiert.

Nachdem in diesem Kapitel die Struktur von WML-Seiten Thema war, lernen Sie im folgenden, wie Sie den Textinhalt von Seiten auszeichnen, also Schriftvarianten verwenden, Absätze formatieren und Verweise auf andere Karten und Stapel setzen.

4 Textauszeichnung

In diesem Kapitel machen Sie die ersten Schritte in die Auszeichnung von Text innerhalb einer Karte. Die einzelnen Abschnitte behandeln zunächst Zeichen, Leerraum und die Entitäten – spezielle Kürzel für Sonderzeichen. Sie lernen dann die grundlegenden Mittel zur Formatierung von Zeichen und zur Markierung von Absätzen kennen.

4.1 Zeichendarstellung auf Decks

Der Inhalt von WML-Seiten besteht aus normalem Text, dessen Darstellung mit Markierungen gesteuert werden kann. Zu dieser Zeichendarstellung kommen Tags zur Festlegung des Layouts oder für komplexere Strukturen wie Tabellen oder Formulare hinzu.

Der normale Fließtext wird innerhalb einer Card als ASCII notiert und vom Microbrowser dargestellt. Für die Darstellung können Sie die normalen Zeichensätze einschließlich deutschsprachiger Umlaute verwenden.

Die WML-Spezifikation erlaubt für Texte den gesamten Zeichenumfang des ISO Standards 10646 ([1]). Dieser Zeichensatz umfaßt in einer 16-Bit-Codierung sämtliche relevanten Schriftzeichen der Welt – einschließlich umfangreicher asiatischer Schriften. Dieser Zeichensatz ist identisch mit dem Industriestandard UNICODE 2.0 ([23]), der mittlerweile breite Unterstützung findet.

Tatsächlich verwenden Sie wahrscheinlich auch jetzt schon ein UNICODE-konformes System, obwohl Ihr Rechner eine 8-Bit-Darstellung für Zeichen verwendet. Mit dem Standard UTF-8 ([57]) ist nämlich eine Abbildung des großen 16-Bit-Zeichensatzes auf eine 8-Bit-Repräsentation definiert. In dieser Darstellung ist nun der Zeichensatz ISO 8859-1 ([2]) so eingepaßt, daß er unverändert übernommen ist. ISO 8859-1 ist das wichtigste Codierungsschema für den europäischen Sprachraum und die USA, da er die gebräuchlichsten Zeichen der lateinischen Sprachenfamilie enthält – also auch die deutschsprachigen Umlaute. Zugleich ist er

seit Jahren *die* dominierende Zeichencodierung in PCs und Workstations.

✘ Während die Spezifikation also eine hohe Internationalität der Zeichendarstellung verspricht, ist ihre Einhaltung gerade in diesem Punkt aber nicht zwingend für WAP-Geräte. Sie können sich leider keineswegs darauf verlassen, daß ein nach UNICODE erfaßtes Zeichen der japanischen Kanji-Schrift auch von einem Handy korrekt dargestellt wird.

Die Behandlung von Leerraum ist in den Spezifikationen von WML und XML eindeutig geklärt. Als Leerzeichen sind das eigentliche Leerzeichen mit ASCII Code 32, Zeilenumbruch (Code 13), Zeilenvorschub (Code 10) und Tabulator (Code 9) definiert. Leerraum (im englischen »whitespace«) ergibt sich aus einem oder mehreren Leerzeichen.

4.2 Entitäten

Einige Zeichen lassen sich nicht direkt darstellen oder nicht mit dem normalen Markup mixen. Ein Beispiel dafür ist das Zeichen `<`, das alle Tags einleitet. Will man nun aber beispielsweise `es gilt 10<20` schreiben, interpretiert das WAP-Gerät das `<` als Beginn eines Tags namens `20`. Zwei Fehler entstehen also: Das Element ist nicht in WML definiert und es wird nicht geschlossen.

Abhilfe schaffen die Entities, vordefinierte Kürzel, die in den Text eingestreut werden können. Sie haben immer die Form

&*Kürzelname*;

Für das Zeichen `<` ist beispielsweise das Kürzel `<` (von »less than«) definiert, so daß das obige Beispiel als `es gilt 10<20` korrekt ausgezeichnet ist.

Abbildung 4.1 auf der nächsten Seite zeigt die in WML definierten Kürzel. Dabei erzeugen die ersten fünf tatsächlich sichtbare Zeichen. `nbsp` und `shy` beeinflussen den Zeilenumbruch und sind in Abschnitt 4.4 auf Seite 30 beschrieben.

Die fünf Zeichenentitäten vergeben quasi einen Namen für Zeichen. In den fünf Fällen ist dies auch erforderlich, um Konflikte mit den für WML-Tags und -Attribute benötigten Zeichen in direkter Darstellung zu vermeiden.

Es kann aber auch sein, daß beim Autoren ein bestimmtes Zeichen gar nicht darstellbar ist. Damit man nicht über 64000 Kürzel benötigt (auf eine leicht höhere Zahl kommt man, wenn man einen

Kürzel	Zeichen
amp	& (von »ampersand«)
quot	" (von »quotation mark«)
apos	' (von »apostrophe«)
lt	< (von »less than«)
gt	> (von »greater than«)
nbsp	Festes Leerzeichen (von »non-breaking space«)
shy	Trennstelle (von »soft hyphen«)

Abbildung 4.1
Die in WML definierten Kürzel

16-Bit-Zeichensatz komplett belegt), lassen sich alle UNICODE-Zeichen numerisch darstellen. Die Notation dafür ist

&#*Zeichencode*;.

Zeichencode ist die Nummer des gewünschten Zeichens in UNICODE. Für die Abbildung in das Codierungsschema ist dann der Browser zuständig. Um das Zeichen > zu schreiben, kann man also gt verwenden, oder in numerischer Codierung &#62 notieren. Als eine Erweiterung des Mechanismus ist auch eine hexadezimale Schreibweise möglich, die mit einem x einzuleiten ist: &#x3E.

Code	Zeichen	Code	Zeichen
Ä	Ä	ä	ä
Ö	Ö	ö	ö
ü	Ü	Ü	ü
		ß	ß

Abbildung 4.2
Die Zeichencodes der deutschsprachigen Sonderzeichen

Zu den oben genannten Kürzeln kommt ein weiteres Sonderzeichen hinzu, daß nicht direkt notiert werden kann, das Dollarzeichen $. In Kapitel 9 auf Seite 85 lernen Sie die Verwendung von Variablen innerhalb von WML-Seiten kennen. Diese Variablen werden mit einem $-Zeichen eingeleitet und das folgende Wort als Variablenname interpretiert. Damit ist $ ähnlich reserviert wie < oder \&. Sie müssen es entweder durch seine numerische Codie-

rung &#24 in eine Seite einfügen oder die Zeichenkette $$ codieren.

✘ Falls Sie dennoch aus Versehen ein einzelnes $ verwenden, macht sich in der Darstellung der Fehler dadurch bemerkbar, daß das Dollarzeichen fehlt. Eine Referenz auf eine bislang unbekannte Variable ergibt die leere Zeichenkette.

Die Verwendung von uncodierten Zeichen birgt ein Problem. Ein Microbrowser ist lediglich dazu verpflichtet, die sieben WML-Kürzel zu verarbeiten und die numerische Codierung zu beherrschen. Darüber hinaus ist der unterstützte Zeichensatz implementierungsabhängig, solange er eine Untermenge von UNICODE repräsentiert.

Der Zeichensatz ISO 8859-1 ist eine solche Untermenge und enthält beispielsweise die deutschsprachigen Umlaute neben den in Westeuropa üblichen weiteren Sonderzeichen wie den akzentuierten Zeichen. Zugleich ist auch US-ASCII eine solche Untermenge und diese enthält keinerlei in Europa notwendige Zeichen.

Insofern kann die Verwendung uncodierter Umlaute problematisch sein, wenn man beispielsweise mit einem japanischen oder US-amerikanischen WAP-Gerät auf die entsprechenden Karten zugreift.

✘ Um die Darstellbarkeit Ihrer Seiten wirklich zu sichern, sollten Sie immer die numerisch codierte Form aller Zeichen außer Groß- und Kleinbuchstaben sowie den Ziffern verwenden. Falls Ihr WAP-Angebot allerdings für einen bestimmten Sprachraum geplant ist, können Sie diese sehr strenge Regel etwas vereinfachen.

4.3 Textformatierung

WML bietet Ihnen verschiedene Möglichkeiten, um die Darstellung einzelner Textstellen zu steuern. Diese sind bei weitem nicht so reichhaltig, wie Sie es von einem Web-Browser her kennen – diese Einschränkung ergibt sich natürlich aus den Gegebenheiten der heutigen Handys. So haben Sie keine Auswahlmöglichkeit bei der Schrift – ein Microbrowser verwendet lediglich eine feste Schrift in wenigen Varianten.

<b>
<u>

Ähnlich wie bei der Textverarbeitung lassen sich Varianten der Schrift direkt festlegen. In ... (vom englischen »italics«) eingeschlossenen Text stellt der Microbrowser mit kursiver Schrift dar, <b> (von »bold«) steuert Fettschrift an und <u> schließlich be-

wirkt eine Unterstreichung des markierten Textes, wie in Abbildung 4.3 dargestellt..

```
<p>Text in <i>kursiv</i>,
<b>fett</b> und
<u>unterstrichen</u>.
</p>
```

Abbildung 4.3
Die Schriftvarianten kursiv, fett und unterstrichen

Mit diesen drei Tags steuern Sie die Darstellung von Text direkt. Eventuell ist das WAP-Gerät aber gar nicht in der Lage, alle drei genannten Textstile darzustellen.

Falls Sie die Formatierung zur Hervorhebung des Inhalts verwenden, sollten Sie dazu zwei andere WML-Tags verwenden. `<em>` (von »emphasis«) markiert einen hervorgehobenen Textabschnitt. Soll die Hervorhebung noch stärker sein, können Sie `<strong>` verwenden. Bei dieser logischen Markierung ist die Art der Darstellung nicht festgelegt und es bleibt dem gerätespezifischen Microbrowser oder den Nutzereinstellungen überlassen, ob einfache Hervorhebung durch Fettschrift oder Unterstreichung dargestellt wird. Abbildung 4.4 zeigt eine mögliche Darstellung.

`<em>`

`<strong>`

Sie sollten daher zum Betonen von Textteilen immer diese beiden Tags einsetzen.

✘

```
<p>übliches Blabla
<em>wichtig</em>
<strong>sehr wichtig</strong></p>
```

Abbildung 4.4
Unterschiedliche Hervorhebungen

Neben der Schriftvariante ist die Größe der Zeichen eine wichtige Eigenschaft der Darstellung. Für WAP-Geräte sind aufgrund

der eingeschränkten grafischen Fähigkeiten keine so großen Auswahlmöglichkeiten wie auf einem modernen PC vorgesehen. WML bietet zwei Tags durch Steuerung der Schriftgröße an. `<big>` wählt eine Schrift, die im Vergleich zur Normalschrift größer ist und `<small>` eine kleinere.

<big>

<small>


```
<p><small>klein</small>
normal
<big>gross</big>
</p>
```

Abbildung 4.5
Die Schriftgrößen

Die Abbildung 4.5 zeigt die sich damit ergebenden drei Schriftgrößen in einem Microbrowser.

✘ Ob ein Gerät tatsächlich drei unterschiedliche Größen unterstützt, ist allerdings nicht garantiert.

4.4 Absatzformatierung

<p>

Absätze innerhalb einer Card werden durch `<p>...</p>` (von »paragraph«) umschlossen.

✘ Leere Absätze (mit `<p/>` oder `<p></p>`) werden im Web oft verwendet, um vertikalen Leerraum zu schaffen. In WML sollten Sie dieses Mittel nicht einsetzen, da der Standard auffordert, solche leeren Absätze bei der Darstellung zu ignorieren.

✘ Mit einem Trick erreicht man allerdings trotzdem den gewünschten Effekt, wie in Abbildung 4.6 auf der nächsten Seite zu sehen ist. Fügt man ein festes Leerzeichen mit `nbsp` in den Absatz ein, kann der Microbrowser ihn nicht mehr ignorieren.

Die Attribute von `<p>` steuern die horizontale Ausrichtung der Zeilen und den automatischen Zeilenumbruch des Microbrowsers. Das Attribut `align` kann dafür drei Werte tragen, deren Wirkung Sie in Abbildung 4.7 auf der nächsten Seite sehen:

```
<p>Absatz</p>
<p></p>
<p>Absatz</p>
<p> </p>
<p>Absatz</p>
```

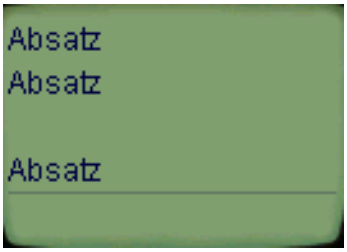


Abbildung 4.6
Ein nicht ganz leerer Absatz wird dargestellt

- `left`: Linksbündige Ausrichtung der Zeilen. Dies ist der Normalwert, der immer dann gilt, wenn das `align`-Attribut nicht verwendet wird.
- `center`: Zentrierung aller Zeilen.
- `right`: Rechtsbündige Ausrichtung der Zeilen.

```
<p>linksbündig</p>
<p align="center">
zentriert</p>
<p align="right">
rechtsbündig</p>
```

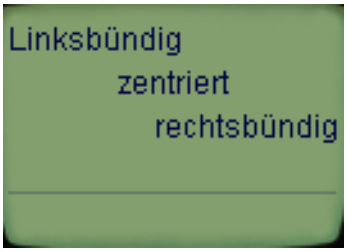


Abbildung 4.7
Die möglichen Absatzausrichtungen

Auf WAP-Geräten ist die Zeilenbreite üblicherweise sehr begrenzt. Der Browser muß bei überlangen Absätzen also entweder Zeilenumbrüche einfügen, oder den »überhängenden« Teil des Absatzes durch geeignete Mittel dem Nutzer zugänglich machen. Ein Beispiel dafür wäre eine zusätzliche horizontale Scroll-Möglichkeit. Das Attribut `mode` von `<p>` steuert dieses Verhalten durch zwei mögliche Werte:

- `wrap`: Der Browser bricht lange Zeilen um.
- `nowrap`: Der Browser führt keinen Zeilenumbruch bei langen Zeilen durch.

Fehlt das `mode`-Attribut, gilt der Umbruchmodus des vorherigen Absatzes weiter. Falls der allererste Absatz einer Karte auch keinen Modus vorgibt, dann gilt die Initialeinstellung für die Karte. Deren Wert kann sich aus einer herstellerseitigen Voreinstellung ergeben, vom Nutzer gesetzt sein oder von der vorherigen Card übernommen werden.

✘ Wie schon angemerkt, werden leere Absätze vom Browser ignoriert. Ein leerer Absatz, der ein `mode`-Attribut trägt, verändert den Umbruchmodus also nicht.

Falls der Zeilenumbruch eingeschaltet ist, kann der Browser bei jedem Leerraum zwischen Worten eine neue Zeile beginnen. Sie können dieses Verhalten durch zwei Entitäten im Text beeinflussen. Wenn Sie anstelle eines Leerzeichens die Entität `nbsp` (von »non-breaking space«) oder die gleichwertige numerische Notation `#160` verwenden, wird der Browser an dieser Stelle keinen Umbruch einfügen. Mit `shy` (von »soft hyphen«) läßt sich eine zusätzliche Trennstelle markieren: Der Browser kann hier umbrechen, muß aber einen Trennstrich an das Ende der Zeile stellen wie in Abbildung 4.8 illustriert.

nbsp

shy

```
<p>Ein mehr als
zwei&shy;zeiliger
längerer Absatz.</p>
```

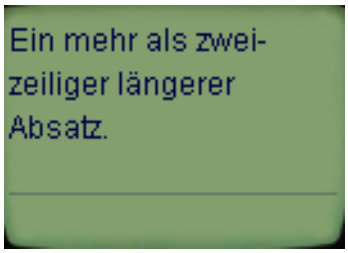


Abbildung 4.8
Eine Trennstelle

Schließlich können Sie einen Zeilenumbruch erzwingen, indem Sie das `<br/>`-Tag verwenden.

*
*

✘ Beachten Sie, daß das Tag immer leer sein muß und daher nur in der Kurzform auftreten kann.

4.5 Links

Wie das Web ist auch WAP ein Hypermedia-System. Informationseinheiten können also miteinander verzeigert werden und Nutzer können entlang dieser Verzeigerungen navigieren.

Eine solche Verbindung – ein »Link« – hat bei WML genau einen Ausgangspunkt – den Quellanker – und einen Zielpunkt – den Zielanker. Auf einen Zielanker kann von unterschiedlichsten Stellen her verwiesen werden, während ein Quellanker immer nur ein Ziel hat.

Der Quellanker besteht aus Text oder einer Abbildung, die im Microbrowser entsprechend dargestellt wird und je nach WAP-Gerät durch unterschiedliche Tasten ansteuerbar ist. Durch die Auswahl wechselt der Nutzer zu einer anderen Karte, zu einem anderen Kartenstapel oder wählt eine andere Informationseinheit aus dem Netz.

Der Quellanker wird in WML mit dem Tag `<a>` markiert. Er kann aus normalem Text mit Zeilenumbrüchen oder Bildern bestehen.

`<a>`

✘ Während ein HTML-Browser auch Auszeichnungen innerhalb eines Ankers erlaubt (wie bei `<a href="x.html"><b>wichtig</b></a>`), sind in WML neben Text ohne Markierungen lediglich `<br>` und `<img>` erlaubt. Ein Anker in fetter Schrift läßt sich also nur so erzeugen: `<b><a href="x.wml">wichtig</a></b>`.

Beim `<a>`-Element können Sie im Attribut `title` einen Kurztitel angeben, der den Link beschreibt. Ein Microbrowser kann ihn beispielsweise einblenden, wenn die Navigationsauswahl über diesem Link steht. `xml:lang`, `id` und `class` sind mit der gewohnten Bedeutung verwendbar.

Das Attribut `href` müssen Sie zwingend verwenden, um das Ziel des Links anzugeben. Die so festgelegte Netzadresse kann relativ zur aktuellen Karte, also innerhalb eines Stapels, relativ zur Netzadresse des Stapels oder eine absolute Netzadresse sein. In Abbildung 4.9 auf der nächsten Seite finden Sie Beispiele für die wichtigen Varianten von URLs.

Zur Trennung zwischen der Netzadresse eines Stapels und der Bezeichnung einer Karte darin wird das #-Zeichen verwendet. Relative URLs bezeichnen einen Stapel ab der Verzeichnis- und Server-Zugehörigkeit des Stapels, in dem der Quellanker steht.

Absolute Adressen bezeichnen ein Protokoll, einen Server und einen Stapel oder eine andere Datei. In Abbildung 4.10 auf Seite 33 finden Sie die wichtigsten Protokolle, mit denen absolute URLs gebildet werden können. Für Microbrowser ist nach dem Standard lediglich die Unterstützung des `http:`-Schemas vorgeschrieben.

URL	Bedeutung
`#zweite`	Die Karte im aktuellen Stapel mit dem `id`-Attribut `"zweite"`.
`kurse.wml`	Der Stapel `kurse.wml` in dem gleichen Verzeichnis auf dem gleichen Server in dem der Stapel mit dem Quellanker steht.
`kurse.wml#zweite`	Die Karte mit dem `id`-Attribut `"zweite"` im Stapel `kurse.wml` in dem gleichen Verzeichnis auf dem gleichen Server in dem der Stapel mit dem Quellanker steht.
`daten/kurse.wml`	Der Stapel `kurse.wml` in dem Verzeichnis `daten` in dem Verzeichnis auf dem gleichen Server in dem der Stapel mit den Quellanker steht.
`http://foo.bar/kurse.wml`	Der Stapel `kurse.wml` auf dem Server `foo.bar`.

Abbildung 4.9
Beispielhafte URLs

In Abbildung 4.11 auf Seite 34 finden Sie ein Beispiel, bei dem in einem Stapel mit zwei Karten mit `<a>` zwischen den beiden Karten und zu externen WAP-Diensten navigiert werden kann.

WML 1.2

Mit WML 1.2 wurde ein weiteres Attribut bei `<a>` eingeführt: `accesskey`. Sein Wert ist ein einzelnes Zeichen, das als Tastenkürzel für diesen Anker dient. Mit `accesskey="0"` veranlassen Sie den Microbrowser, bei Eingabe von `0` zu dem definierten Anker zu springen. Dadurch wird er einfacher auswählbar.

Normale Handys unterstützen zunächst die Tasten einer Telefontastatur: `0` bis `9` sowie `*` und `#`. Der Microbrowser kann `accesskey` ignorieren oder geräteabhängig Tastenkombinationen zur Eingabe solcher Kürzel verlangen.

✘ Bei der Darstellung von Ankern kann es zu einer unschönen Darstellung kommen, wenn Sie unabsichtlich Leerzeichen einfügen. In Abbildung 4.12 auf Seite 35 sehen Sie, daß durch Zeilenumbrüche Leerzeichen entstehen und wie diese bei der Darstellung des Links zu einer unschönen Unterstreichung führen. Durch Vermeidung von Leerraum vor und nach den Tags ist die Darstellung wieder in Ordnung.

Schema	Bedeutung
`http:`	Das Web-Protokoll; ein HTTP-Server wird angesprochen.
`https:`	Das Web-Protokoll; ein HTTP-Server wird über eine sichere, verschlüsselte Verbindung angesprochen.
`file:`	Eine Web-Seite wird geladen, dabei enthält der Pfad aber keine Web-Adresse, sondern einen Dateinamen für Ihr lokales Dateisystem.
`ftp:`	Das File-Transfer-Protokoll zur Übertragung von Dateien; ein FTP-Server wird angesprochen.
`gopher:`	Das Gopher-Protokoll zur Übertragung von Dokumenten und zur Navigation in Gopher-Systemen.
`mailto:`	Mit dem Internet-Mail-Protokoll wird eine E-Mail an den angegebenen Empfänger geschickt.
`news:`	Mit dem Internet-News-Protokoll wird ein Artikel oder eine Newsgruppe geladen und angezeigt.
`telnet:`	Eine Terminal-Sitzung über Internet wird zu einem bestimmten Rechner gestartet.

Abbildung 4.10
Die wichtigsten Protokolle in URLs

Falls beim Übergang zu einer anderen Karte Fehler auftreten, beispielsweise weil die Adresse ungültig ist oder das Netzwerk nicht funktioniert, bleibt der Microbrowser bei der Darstellung der aktuellen Karte. Er muß lediglich den Nutzer über den Fehler informieren und darf ansonsten keinerlei Änderungen an seinem Zustand, beispielsweise bei Variablen oder der Nutzungsgeschichte, vornehmen.

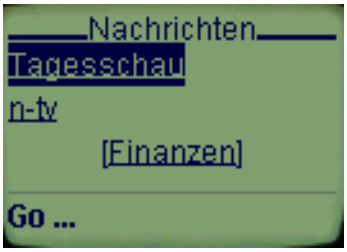
✘

`<a>` bindet eine bestimmte Aktion an die Auswahl eines Quellankers: Die Navigation zum Ziel. In Kapitel 8 auf Seite 67 werden Sie sehen, daß WML über ein komplexeres Konzept für Ereignisse und Aktionen verfügt. Neben der Navigation zu einem Ziel gibt es noch weitere Aktionen, beispielsweise die Neuanzeige der aktuellen Seite.

`<anchor>`

Mit dem Tag `<anchor>` können Sie solche weitergehenden Aktionen an einen Anker binden. Dabei umschließt `<anchor>`...

```
<card id="infolinks" title="Nachrichten">
  <p><a href="http://wap.tagesschau.de"
    >Tagesschau</a><br/>
   <a href="http://wap.n-tv.de">n-tv</a>
  </p>
  <p align="center">
    [<a href="#geldlinks">Finanzen</a>]
  </p>
</card>
<card id="geldlinks" title="Finanzen">
  <p><a href="http://wap.ftd.de"
    >Financial Times</a><br/>
    <a href="http://wap.financial.de"
    >financial.de</a>
  </p>
  <p align="center">
    [<a href="#infolinks">Nachrichten</a>]
  </p>
</card>
```



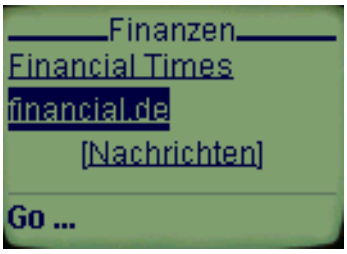


Abbildung 4.11
Ein Stapel mit verzeigerten Karten

`</anchor>` Text mit Zeilenumbrüchen oder Bildern und Tags, die Aktionen bezeichnen. Die Beschreibung der hier möglichen Elemente `<go>`, `<prev>` und `<refresh>` finden Sie in Abschnitt 8.1 auf Seite 67 – vielleicht sollten Sie nach dessen Lektüre kurz zur Beschreibung von `<anchor>` zurückkehren.

`<a href="url">Text</a>` ist eine Abkürzung für die folgende Benutzung von `<anchor>`

```
<anchor>
  <go href="url"/>
  Text
</anchor>
```

Das Tag kann neben den Standardattributen `xml:lang`, `id` und `class` das Attribut `title` verwenden. Es enthält eine Kurzbeschreibung des Ankers, die bei Auswahl angezeigt oder auf andere Weise als Hilfestellung dargestellt wird.

WML 1.2

Wie bei `<a>` können Sie bei `<anchor>` das in WML 1.2 eingeführte Attribut `accesskey` verwenden, um ein Tastenkürzel für diesen Anker zu bestimmen.

```
Ein <a href="x.wml">
Link
</a> hier,<br/>
Ein <a href="x.wml">Link</a> da
```

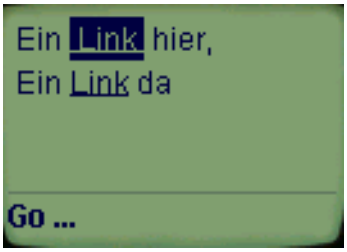


Abbildung 4.12 *Unbeabsichtigte Leerzeichen*

4.6 Uninterpretierten Text darstellen

Normalerweise werden alle Bestandteile einer WML-Seite vom WAP-Gerät auf Tags untersucht und entsprechend dargestellt. Sie können diese Interpretation aber auch für einzelne Passagen abschalten, so daß der Browser Tags, Kommentare und alles Weitere, was eigentlich zum Markup gehört, darstellt.

Dazu muß diese Passage mit der etwas ungewöhnlichen Markierung `<![CDATA[...]]>`umschlossene Inhalt wird direkt vom Microbrowser dargestellt.

`<![CDATA[ ]]>`

In Abbildung 4.13 finden Sie dafür ein Beispiel. Sie sehen, daß zwar noch Zeilenumbrüche zu Leerraum normalisiert werden, dann aber keinerlei Interpretation des Markups mehr stattfindet. Der Kommentar wird dargestellt und auch die falsch verwendeten Tags rufen keinen Fehler hervor.

```
<p>
  <![CDATA[
    <p>
      <!-- ein Kommentar -->
      Hallo
    </p><b></p>
  ]]>
</p>
```

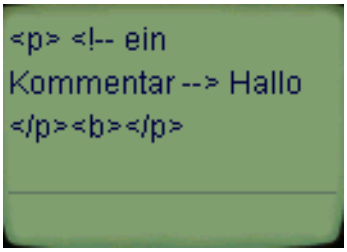


Abbildung 4.13 *Uninterpretierte WML-Daten*

Mit WML 1.2 wurde diese Möglichkeit auch einfacher angeboten: Der vom Tag `<pre>` (von »preformatted« – vorformatiert) umschlossene Text wird ohne Interpretation darin vorhandener Tags dargestellt. Dabei kann der WML-Browser Leerraum auch

WML 1.2

`<pre>` mit mehreren Leerzeichen so darstellen, einen Schreibmaschinenzeichensatz wählen und automatischen Zeilenumbruch ausschalten. Das Tag kann lediglich die Standardattribute `id` und `class` tragen.

Mit den WML-Grundlagen aus diesem Kapitel können Sie einfache Fließtexte gestalten. Sie sind in der Lage, mit Zeichen und Leerraum richtig umzugehen und einfache Schriftvarianten anzusteuern. Für die Gestaltung von Absätzen können Sie verschiedene Ausrichtungen verwenden und Zeilenumbrüche gezielt steuern.

Im folgenden Kapitel geht es um komplexere Strukturen auf WML-Seiten, die Tabellen.

5 Tabellen

Tabellen sind ein sehr gutes Mittel, um Informationen optisch strukturiert darzustellen oder Zahlen aufzubereiten. Wie HTML beherrscht auch WML ein einfaches Tabellenmodell. Dabei wird eine Tabelle mit `<table>...</table>` umschlossen und besteht aus einer Anzahl von Tabellenzeilen, die mit `<tr>...</tr>` markiert sind. In jeder Zeile befinden sich gleich viele Tabellenzellen, die jeweils das Tag `<td>` umschließt.

```
<table columns="2" title="Eurokurs">
  <tr>
    <td>Franc</td>
    <td>6.55957</td>
  </tr>
  <tr>
    <td>Peseta</td>
    <td>166.38600</td>
  </tr>
  <tr>
    <td>Mark</td>
    <td>1.95583</td>
  </tr>
</table>
```

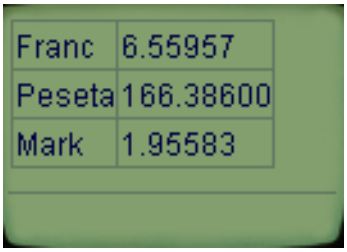


Abbildung 5.1
Eine einfache Tabelle

In Abbildung 5.1 sehen Sie eine einfache Tabelle, die Euro-Umrechnungskurse enthält. Der Browser fügt entsprechende Linien hinzu und richtet Zeilen und Spalten entsprechend aus.

5.1 Die Tabellenstruktur

`<table>`

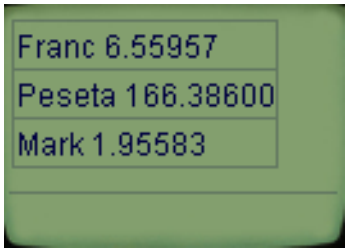
Das Tag `<table>` umschließt eine beliebige Anzahl von Tabellen-

zeilen. Es legt über seine Attribute die Anzahl der Tabellenspalten und deren Ausrichtung fest. Die Attribute sind im einzelnen:

- `columns` bestimmt die Anzahl der Spalten der Tabelle und muß verwendet werden.

✘ Anders als bei HTML ist nicht vorgesehen, daß der Browser die Spaltenanzahl selbst aus den vorgefundenen Tabellenzellen errechnet.

Befinden sich in den Tabellenzeilen weniger Zellen als Spalten definiert waren, kann der Browser einfach leere Zellen hinzuformatieren. Falls man weniger Spalten ankündigt als sich dann Zellen finden, kann der Inhalt der letzten mit dem der überzähligen Zellen zusammengefügt werden, wie Abbildung 5.2 zeigt.




```
<table columns="1" title="Eurokurs">
  <tr>
    <td>Franc</td>
    <td>6.55957</td>
  </tr>
  <tr>
    <td>Peseta</td>
    <td>166.38600</td>
  </tr>
  <tr>
    <td>Mark</td>
    <td>1.95583</td>
  </tr>
</table>
```

Abbildung 5.2
Zellen ignorieren

✘ Die Behandlung zu vieler oder zu weniger Zellen ist implementierungsabhängig, so daß Sie das `columns`-Attribut immer korrekt setzen sollten.

✘ Bedenken Sie auch, daß ein Display in einem Handy ausgesprochen klein ist. Die Verwendung von mehr als zwei Spalten kann daher problematisch werden und ist nur bei kurzen Zelleninhalten empfehlenswert.

- `title` legt einen Titel der Tabelle fest, den der Microbrowser geeignet darstellen kann aber nicht muß. Der für die Beispiele verwendete Browser ignoriert dieses Attribut.

  Sie sollten aber trotzdem einen Titel angeben – schließlich müssen Sie damit rechnen, daß Ihre Dokumente auch auf Geräten mit besseren Darstellungsmöglichkeiten angeschaut werden.

- `align` kann die horizontale Ausrichtung des Inhalts der Zellen steuern. Der Wert des Attributs ist eine Zeichenkette, deren Länge der Anzahl der vorgesehenen Spalten entspricht. Mit jeweils einem Zeichen wird die Ausrichtung der Spalte festgelegt. Mögliche Steuerzeichen sind `L` für linksbündige Ausrichtung, `C` für Zentrierung und `R` für rechtsbündige Formatierung. Abbildung 5.3 zeigt ein Beispiel.

```
<table columns="2" title="Eurokurs"
 align="LR">
  <tr>
    <td>Franc</td>
    <td>6.55957</td>
  </tr>
  <tr>
    <td>Peseta</td>
    <td>166.38600</td>
  </tr>
  <tr>
    <td>Mark</td>
    <td>1.95583</td>
  </tr>
</table>
```

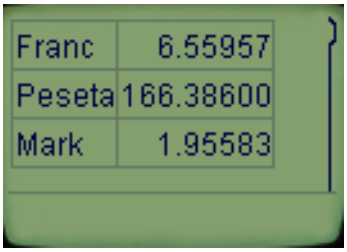


Abbildung 5.3
Rechtsbündige Ausrichtung der zweiten Spalte

WML 1.2

Mit WML 1.2 ist ein weiteres Steuerzeichen für die Ausrichtung hinzugekommen: `D` für die Einstellung der Standardformatierung. Der Textinhalt könnte ja in einer Sprache sein, die von rechts nach links formatiert wird. Eine Vorgabe `L` wäre also nicht richtig. Die Bestimmung der Standardformatierung ist Aufgabe des Microbrowsers und implementierungsabhängig.

✘ In der Darstellung stehen bei den Zahlen der zweiten Spalte die Dezimalpunkte genau übereinander. Dies rührt daher, daß die Anzahl der Nachkommastellen übereinstimmt und rechtsbündige Formatierung gewählt wurde. Eine Möglichkeit zur Ausrichtung am Dezimalpunkt ist in WML nicht vorgesehen.

- `xml:lang` gibt an, in welcher Sprache der Inhalt der Tabelle gehalten ist. Anhand des Sprachkürzels nach den Tabellen in Anhang C auf Seite 275 legt der Browser damit auch die Ausrichtung für Tabellenzellen fest, falls eine entsprechende Angabe in `align` fehlt. Bei Sprachen, die von links nach rechts geschrieben werden, ist die Standardausrichtung linksbündig, bei Sprachen, die von rechts nach links laufen, ist sie rechtsbündig.

<tr>

Jede Tabellenzeile schließt `<tr>` (von »table row«) ein. Dabei kann eine Zeile auch leer sein – der Browser darf leere Tabellenzeilen in der Darstellung nicht unterschlagen.

<td>

Die Zellen einer Tabelle werden innerhalb einer Zeile mit `<td>` (von »table data«) markiert. Hier ist wieder das Attribut `xml:lang` möglich, mit dem die Sprache für einzelne Zellen markiert wird.

`<table>`, `<tr>` und `<td>` können jeweils die in Abschnitt 3.1 auf Seite 16 beschriebenen Standardattribute `id` und `class` tragen.

✘ Anders als in HTML dürfen Tabellenzellen keineswegs komplexe Strukturen enthalten wie beispielsweise geschachtelte Tabellen. Zulässig sind im Detail die Tags `<em>`, `<strong>`, `<b>`, `<i>`, `<u>`, `<big>`, `<small>`, `<br>`, `<img>`, `<anchor>` und `<a>`.

Mit Tabellen lassen sich ein paar einfache Effekte erzielen. So können Sie mit einer Tabelle mit nur einer Zelle den Textblock darin umranden, wie in Abbildung 5.4 auf der nächsten Seite zu sehen.

Wollen Sie auch die Tabelle selber zentrieren, so müssen Sie sie in einen zentrierten Absatz verpacken. Abbildung 5.5 auf der nächsten Seite zeigt den Effekt und zudem, daß auch mehrzeilige Textblöcke auf diese Weise mit einem Rahmen versehen werden können.

In diesem Kapitel haben Sie die Möglichkeiten zur Auszeichnung von Tabellen mit WML kennengelernt. Mit ihnen lassen sich sehr einfach Datenmengen strukturiert darstellen, allerdings

```
<p>
  <table columns="1" align="C">
    <tr><td>Finanznachrichten<td/><tr>
  </table>
  1.4.2000 (xyz) Alles wird teurer.
</p>
```

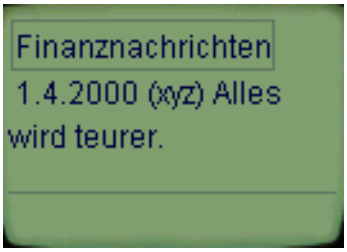


Abbildung 5.4
Eine Tabelle zur Umrahmung eines Textblocks

reicht das WML-Tabellenmodell nicht an die von der Web-Sprache HTML gewohnte Gestaltungsvielfalt heran.

Mit Tabellen lassen sich Datenmengen übersichtlich darstellen. Oft führen aber Grafiken sehr schnell zur Information und darum geht es im nächsten Kapitel. Sie lernen, wie Sie einfache Grafiken auf WML-Seiten einbinden.

```
<p align="center">
  <table columns="1" align="C">
    <tr>
     <td>Finanznachrichten von heute</td>
    </tr>
  </table>
</p>
<p>1.4.2000 (xyz) Alles wird teurer.
</p>
```

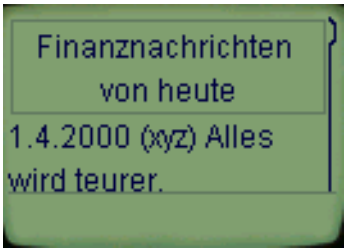


Abbildung 5.5
Eine zentrierte Tabelle zur Umrahmung

6 Grafiken

Die Verbreitung von grafikfähigen HTML-Browsern, die auch Bilder darstellen konnten, war einer der entscheidenden Auslöser für den rasanten Erfolg des Web. Auch in WML-Karten lassen sich Grafiken einbinden – dabei schränken die Gegebenheiten von WAP-Geräten allerdings die Möglichkeiten ein.

Microbrowser können lediglich ein Grafikformat darstellen, das speziell auf sie abgestimmt ist: Das Wireless Bitmap Format WBMP. Während die Spezifikation mehrere Ausbaustufen des Formats vorsieht, ist bislang erst der einfachste Typ standardisiert, WBMP Level 0. Bei diesen Grafiken handelt es sich um Schwarz-Weiß-Pixelmuster, die keinerlei Komprimierung kennen. Sie eignen sich damit beispielsweise nicht für die Darstellung von Fotos.

Die Frage der Grafikqualität ist damit eine Frage der unterstützten Grafikformate. Für die Einbindung von Grafiken in WML-Seiten ist ein Tag – `<img>` – vorgesehen und jedes grafikfähige WAP-Gerät muß zumindest das einfache Level-0-Format beherrschen.

6.1 Grafiken einbinden

`<img>`

WBMP-Grafiken binden Sie in eine WML-Seite mit dem `<img>`-Tag ein. Mit einer Reihe von Attributen steuern Sie dabei Details der Darstellung.

Notwendig ist die Verwendung des Attributs `src`. In ihm vermerken Sie einen Verweis auf die darzustellende Grafik im WBMP-Format. Dieser Verweis ist eine normale Netzadresse als URL.

Einige Microbrowser bieten einen Satz vordefinierter und oft verwendeter Icons fest gespeichert an. In diesem Fall wäre ein Nachladen über das Netz sicherlich überflüssig. Sie können mit dem Attribut `localsrc` den Namen eines solchen vordefinierten Icons angeben. Ein Netzzugriff findet nur dann statt, wenn das angegebene Icon dem Gerät nicht bekannt ist.

✘ Es gibt momentan keine Standardisierung der Benennung von Icons, so daß alle Namen herstellerabhängig sind, wenn `localsrc` überhaupt implementiert wird. Während es sich also konzeptionell um eine gute Idee zur Vermeidung von Netzzugriffen handelt, sollten Sie sicherheitshalber immer auch im `src`-Attribut auf sinnvolle Grafiken verweisen.

Falls das WAP-Gerät eine Grafik nicht darstellen kann oder sie nicht auffindbar ist, soll ein textueller Ersatz zumindest etwas Information an den Nutzer liefern. Im `alt`-Attribut – dessen Verwendung wie bei `src` vorgeschrieben ist – müssen Sie eine kurze Zeichenkette vermerken, die anstelle der Grafik dargestellt werden kann. Das Beispiel in Abbildung 6.1 zeigt das beim zweiten `<img>`-Tag, das auf die nicht existierende Grafik `fehler.wbmp` verweist.


```
<p>Ein Bild:
<img src="square.wbmp" alt="[X]"
 align="top"/>
<br/>Kein Bild:
<img src="fehler.wbmp" alt="[X]"
 align="middle"/></p>
```

Abbildung 6.1
Das `alt`*-Attribut*

Die Größe einer eingebundenen Grafik läßt sich für den Microbrowser erst ermitteln, wenn sie geladen wurde. Mit den Attributen `height` und `width` können Sie dem Microbrowser mitteilen, wieviel Raum er für die Grafik lassen soll. Dadurch kann während des Ladevorgangs schon mit Formatierung und Darstellung der restlichen Karteninhalte begonnen werden.

Als Attributwerte können Sie die tatsächlichen Ausmaße der Grafik in Pixel angeben. Bei der Angabe größerer oder kleinerer Werte kann der Microbrowser versuchen, die Grafik entsprechend zu skalieren. Anstelle eines Pixelmaßes kann der Wert des Attributs auch einen prozentualen Anteil an dem zur Verfügung stehenden Raum sein.

✘ Beide Attribute sind lediglich Hinweise an das WAP-Gerät. Sie können ganz oder teilweise beachtet oder einfach ignoriert werden. Trotzdem sollten Sie immer beide Attribute nutzen.

Zwei weitere Attribute steuern den Leerraum, den das WAP-Gerät um eine Grafik läßt. `hspace` kann einen absoluten Wert als Pixelanzahl enthalten, wie in Abbildung 6.2 gezeigt. Da `hspace` eine Längenangabe ist, können Sie als Wert auch einen Prozentwert verwenden, der angibt, welcher Anteil des verbleibenden Leerraums links und rechts der Grafik freigelassen werden soll.

```
M<img src="square.wbmp"
 alt=""/>M<img src="square.wbmp"
 alt="" hspace="10"/>M
```

Abbildung 6.2
Leerraum neben Grafiken

`vspace` steuert den Leerraum über und unter einer Grafik. Abbildung 6.3 zeigt die Darstellung ohne das `vspace`-Attribut und mit einem Wert von fünf Pixeln mit `vspace="5"`. Es handelt sich wieder um eine Längenangabe und Sie können auch dieses Attribut mit einem Prozentwert belegen.

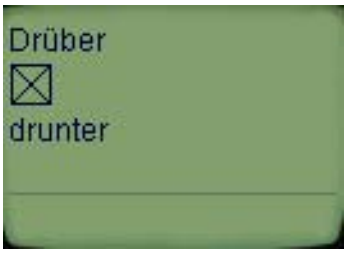


Abbildung 6.3
Leerraum über und unter Grafiken

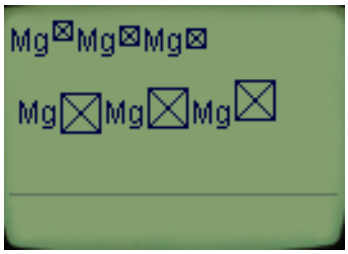
Beide Attribute sind lediglich Hinweise an das WAP-Gerät und können von diesem ignoriert werden. ✘

Das `align`-Attribut bestimmt die vertikale Ausrichtung des Bildes zum Text. Es gibt drei mögliche Werte:

- `bottom` richtet die Unterkante des Bildes mit der Grundlinie der Zeile ab. An der Grundlinie werden die Buchstaben ausgerichtet, wobei Unterlängen – wie beim »g« – unter die

Grundlinie ragen. In Abbildung 6.4 stehen somit die Unterkanten vom »M« und dem Bild auf einer Höhe.

- Der Wert `middle` richtet das Bild an der vertikalen Mitte der Textzeile aus.
- `top` schließlich bringt die Oberkante des Bildes und den oberen Rand der Zeile auf eine Linie. Wie in der Abbildung zu sehen, ist die Zeilenhöhe über der Oberkante von Großbuchstaben.




```
<p>Mg<img src="smallsquare.wbmp"
 alt="" align="top"
/>Mg<img src="smallsquare.wbmp"
 alt="" align="middle"
/>Mg<img src="smallsquare.wbmp"
 alt="" align="bottom"/>
<br/>
Mg<img src="square.wbmp" alt=""
 align="top"/>Mg<img src="square.wbmp"
 alt="" align="middle"
/>Mg<img src="square.wbmp" alt=""
 align="bottom"/>
</p>
```

Abbildung 6.4
Vertikale Ausrichtung von Grafiken

✘

An dem Beispiel sehen Sie an der zweiten dargestellten Zeile, daß die Grafik selbst natürlich die Höhe der Zeile beeinflußt und damit auch die Ausrichtung. Während in der ersten dargestellten Zeile eine Grafik `squaresmall.wbmp` mit 8 Pixeln Höhe und Breite verwendet wird, ist `square.wbmp` 16 mal 16 Pixel groß. Sie ist damit größer als die verwendete Schrift und erhöht somit die Gesamthöhe der Zeile. Dementsprechend sind die relativen vertikalen Abstände zwischen den Buchstaben und den Grafiken gerade umgekehrt.

Schließlich kann `<img>` das Ihnen schon bekannte Attribut `xml:lang` für die Sprache des Textes im `alt`-Attribut tragen und mit den Standardattributen `id` und `class` versehen sein.

Ähnlich wie bei HTML eröffnen Grafiken verschiedene Möglichkeiten, die Darstellung von Karten sehr genau zu steuern.

Einer der Schlüssel dazu sind leere Grafiken, mit denen sich Leerraum erzeugen läßt. »Leere Grafik« bedeutet hier eine Grafik, die nicht sichtbar ist. Im Web erzeugt man diesen Effekt mit vollständig transparenten Grafiken, bei denen die einzige verwendete Farbe quasi »durchsichtig« gemacht wird. Bei WBMP-Grafiken ist dies momentan nicht vorgesehen, da ja lediglich zwei Farben möglich sind.

In den Beispielen verwenden wir stattdessen Grafiken mit wenigen Pixeln, die immer die Farbe Weiß haben. Mit `1x2.wbmp` ist eine Grafik benannt, die ein weißes Pixel hoch und zwei weiße Pixel breit ist.

In Abbildung 6.5 sehen Sie, wie man damit horizontalen Leerraum pixelgenau erzeugt. Zwischen die beiden »M« wird jeweils

```
<p>
  M<img src="1x1.wbmp" hspace="0"
        alt=""/>M<br/>
  M<img src="2x1.wbmp" hspace="0"
        alt=""/>M<br/>
  M<img src="3x1.wbmp" hspace="0"
        alt=""/>M<br/>
  M<img src="1x1.wbmp" hspace="0" alt=""/>
   <img src="3x1.wbmp" hspace="0"
        alt=""/>M
</p>
```

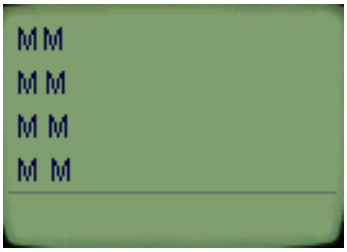


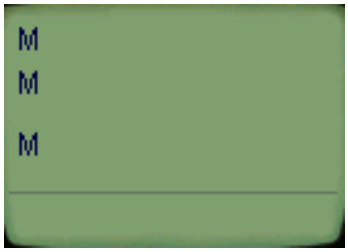
Abbildung 6.5
Horizontaler Leerraum mit Grafiken

eine unsichtbare Grafik eingefügt, die ein, zwei oder drei Pixel breit ist. In der letzten Zeile sind zwei Grafiken kombiniert, um vier Pixel Leerraum zu erzeugen.

Die Skalierung einer Grafik mit einem Pixel Größe mit `height` und `width` könnte als Alternative zur Verwendung mehrerer Grafiken in unterschiedlichen Größen erscheinen. Ein Microbrowser kann diese Attribute aber ignorieren, so daß der Weg über mehrere Grafiken sicherer ist. ✘

Um wirklich bildpunktgenau zu arbeiten, müssen Sie mit `hspace="0"` dafür sorgen, daß kein zusätzlicher Leerraum neben den Grafiken entsteht. ✘

Das gleiche Verfahren kann auch vertikale Zwischenräume schaffen, wenn man hohe unsichtbare Grafiken nutzt, wie Abbil-




```
<p>
  M<br/>
  M<br/>
  <img src="1x7.wbmp" vspace="0"
       alt=""/><br/>
  M<br/>
</p>
```

Abbildung 6.6
Vertikaler Leerraum mit Grafiken

dung 6.6 zeigt. Hier wurde zwischen die Zeilen eine Grafik eingefügt, die sieben weiße Pixel hoch ist und damit einen entsprechenden Abstand erzeugt.

✘ Auch hier müssen Sie zusätzlichen Leerraum um die Grafik vermeiden, also mit `vspace="0"` einen Rand um die Grafik unterbinden.

6.2 Grafiken erstellen

WBMP ist ein eigens für WAP definiertes Bildformat. Wenn Sie Web-Angebote unterhalten, werden Sie Tools benutzen, die GIF- und JPG-Dateien erzeugen und verarbeiten – für Ihr WAP-Angebot benötigen Sie entsprechende Tools für WBMP.

In vielen Entwicklungsumgebungen finden Sie kleine Zeichenprogramme für WBMP-Dateien eingebaut – Abbildung 6.7 auf der nächsten Seite zeigt diese Komponente aus dem *Nokia WAP Toolkit*.

Mittlerweile gibt es auch einfache Editoren als eigenständige Programme wie *WAPDraw*, das Sie unter *http://www.phnet.fi/public/jiikoo* finden.

Viele Grafikprogramme lassen sich auch um neue Formate erweitern – ein solches Plugin für *Adobe Photoshop* finden Sie bei *http://www.creationflux.com/laurent/wbmp.html*. Es ergänzt das Programm um die Fähigkeit, WBMP-Dateien zu schreiben und zu lesen. Solche Erweiterungen werden sich für alle nennenswerten Bildbearbeitungsprogramme auftreiben lassen. In der nächsten Generation dieser Tools wird WBMP sicherlich direkt unterstützt.

Bei größeren Bildbeständen bietet sich die Konvertierung aus einem gängigen Format wie GIF, JPG oder BMP an. Sie finden eine Liste solcher Konvertierprogramme beispielsweise beim *Open*

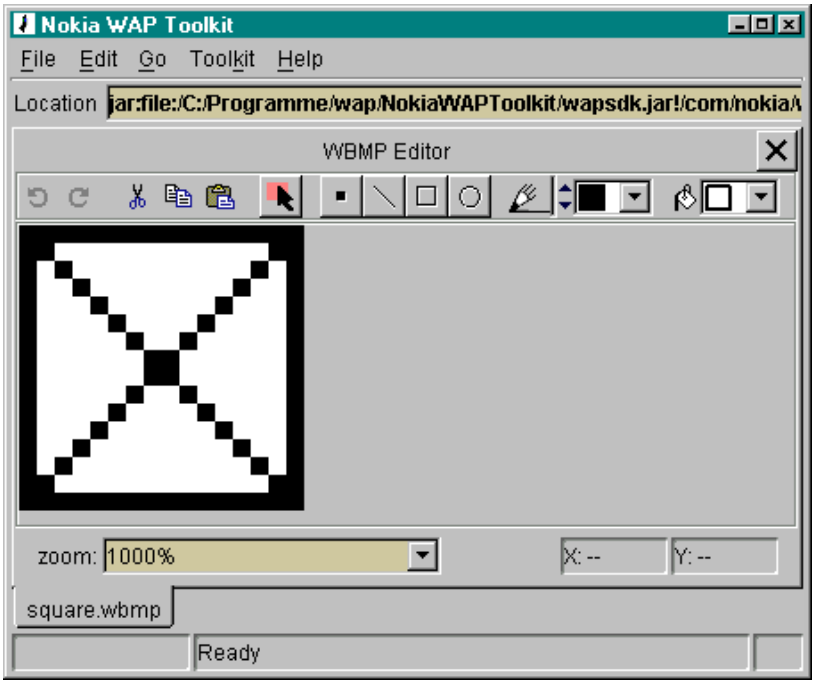


Abbildung 6.7
Der WBMP-Editor im Nokia WAP Toolkit

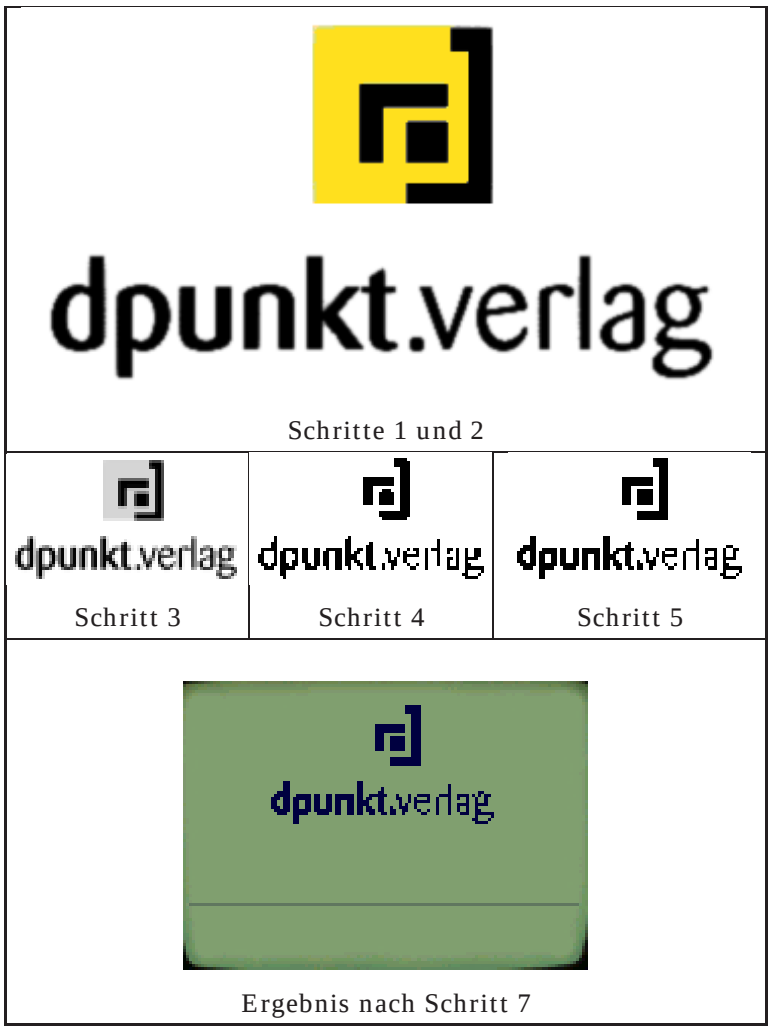
Directory Project unter *http://www.dmoz.org* durch Suchen nach dem Schlagwort WBMP.

6.3 Vom Web-Logo zum WBMP-Bild

Zur Verwendung im Web sind sehr viele Logos, Grafiken und Abbildungen speziell aufbereitet. Um sie auch in WAP-Diensten verwenden zu können, müssen sie konvertiert werden. Dabei ist der folgende Weg gangbar. Er zeigt, daß eine automatische Konvertierung wahrscheinlich nicht möglich ist, und auch viele Grafiken gar nicht für WML-Seiten einsetzbar sind. Die Schritte – deren Ergebnisse Sie in Abbildung 6.8 auf der nächsten Seite sehen –, um von einem Web-Logo zu einem WBMP-Bild zu gelangen, sind:

1. Im Ausgangsschritt liegt ein Logo oder eine einfache Grafik in einem üblichen Bildformat wie GIF oder JPG vor. In unserem Beispiel handelt es sich um das Logo des dpunkt.verlag mit einer Größe von 360 mal 193 Pixel.

Abbildung 6.8
Bearbeitungsschritte vom GIF zu WBMP



2. Die Farben der Grafik sollten im ersten Schritt zu Graustufen konvertiert werden. Sie sollten sie noch nicht nach Schwarz-Weiß-Grafiken umwandeln. Vor dem Palettenwechsel können Sie – falls erforderlich – mit Retouchefiltern wie Kantenglättung o.ä. das Motiv klarer erscheinen lassen. Oft bietet sich auch eine Kontrastverstärkung an.

3. Der nächste Schritt ist das Skalieren der Grafik auf eine den üblichen Handy-Displays angepaßte Größe. Hier beginnt ein grundsätzliches Problem: Es gibt keine »Normalgröße«, auf

die man hinarbeiten könnte. Während ein Nokia 7110 einem 95 Pixel Breite und 45 Pixel Höhe bereitstellt, sind es bei einem Ericsson R320 sehr ähnliche, aber dennoch größere 101 auf 52 Pixel. Stellt man dem noch die Auflösung von 360 mal 120 Pixel eines Ericsson R380 gegenüber, wird klar, daß man in einem echten Dilemma steckt. Wahrscheinlich wird man es auf den kleinsten gemeinsamen Nenner bringen und die Bildgröße dem kleinsten Display anpassen, welches bei den potentiellen Nutzer am weitesten verbreitet ist.

Beim Skalieren sollten Sie vom Anti-Alias-Effekt Gebrauch machen. Das Beispiel wurde auf eine Größe von 80 mal 29 Pixel verkleinert.

4. Das skalierte Bild wird im nächsten Schritt in eine Schwarz-Weiß-Grafik konvertiert. Hierbei müssen Sie vermutlich den Schwellenwert der Farbgrenze in mehreren Versuchen ausloten. Darin liegt eine der Hauptschwierigkeiten für eine erfolgreiche Konvertierung.

   An dieser Stelle kann sich herausstellen, daß ein Logo nicht mit akzeptabler Qualität zu skalieren und auf zwei Farben zu reduzieren ist. Ihnen wird dann keine andere Möglichkeit als ein Motivwechsel bleiben. Letzte Versuche mit einer um ein bis zwei Pixel veränderten Skalierung im vorherigen Schritt können noch sinnvoll sein. ✘

5. Die Konvertierung wird in der Regel nicht zu einem akzeptablen Ergebnis führen. Es ist dann notwendig, manuelle Nachbearbeitungen des Bildes durchzuführen. So mußten im Beispiel die Ecken der Balkengrafik nachgezeichnet werden.

6. Das Bild wird als GIF gespeichert. Damit haben Sie eine Referenzversion zur späteren Bearbeitung oder Korrektur von Fehlern.

7. Schließlich wandeln Sie im letzten Schritt das Bild nach WBMP um. Dazu können Sie einen Export-Filter benutzen, einen Konverter oder eine WML-Entwicklungsumgebung, die Standardbildformate importieren kann.

Wenn Sie einfache Grafiken haben, können Sie mittlerweile auch Online-Dienste benutzen, die ein GIF, JPG oder BMP nach WBMP wandeln. Ein solcher Dienst ist der Konverter von Teraflops unter `www.teraflops.com/wbmp`. Sie können auch beispielsweise bei ✘

www.yahoo.com nach WBMP suchen und erhalten eine Fülle von Verweisen auf Konverter und Konverterdienste.

Mit Texten, Tabellen und Grafiken können Sie viele Informationen darstellen. Interaktive Informationssysteme, in denen Nutzereingaben verarbeitet werden, bieten noch mehr Wert. Im folgenden Kapitel lernen Sie dazu den Aufbau und die Verwendung von Formularen zur Dateneingabe in WML-Seiten kennen.

7 Formulare in WML

Wie in HTML kann man mit WML Formulare darstellen, in die Nutzer interaktiv Daten eintragen, zum Beispiel Empfänger und Kontonummer in einem Überweisungsformular. Diese Daten können dann zur Weiterverarbeitung an einen Server geschickt werden, der üblicherweise neue Seiten auf Basis dieser Eingaben erzeugt. Im Beispiel der Überweisung könnte das eine Überweisungsbestätigung sein. In diesem Kapitel lernen Sie die WML-Möglichkeiten zur Erstellung von Formularen kennen.

Während in HTML ein Formular immer einen Button zur Bestätigung braucht, der die eingegebenen Daten sofort an den Server schickt, basieren WML-Formulare darauf, die Eingaben zunächst an Variablen zu binden. Erst durch die Navigation zu einer URL, der diese Variablen als Parameter hinzugefügt sind, gelangen sie zum Server.

Durch diese Entkoppelung der Eingabe von der Übertragung benötigt man in WML keine gesonderte Auszeichnung von Formularen (in HTML mit `<FORM>`) mehr, sondern kann Eingabefelder beliebig auf Karten einstreuen.

Die Details der Übermittlung von Formulardaten an einen Web-Server finden Sie in Abschnitt 8.2 auf Seite 70 beschrieben. Dieses Kapitel beschäftigt sich zunächst nur mit den Möglichkeiten, Formulare zu gestalten. ✘

Zwei Arten von Eingabefeldern kann man in WML unterscheiden. Mit Auswahlfeldern kann sich der Nutzer zwischen vorgegebenen Möglichkeiten entscheiden, während er bei Eingabefeldern die Gerätetastatur zur Eingabe von Zeichen und Ziffern nutzt.

7.1 Auswahlfelder

`<select>`

Mit `<select>` (von »auswählen«) definieren Sie Auswahllisten, bei denen der Benutzer einen der vorgegebenen Werte selektiert. Die Einträge in der Auswahlliste werden jeweils mit dem Tag

<option> markiert. Der Microbrowser bietet dem Nutzer bei Anwahl einer solchen Liste eine entsprechende Auswahlmöglichkeit an.

✘ <select> und <option> arbeiten sehr ähnlich den Ihnen vielleicht aus HTML bekannten <SELECT>- und <OPTION>-Tags, unterscheiden sich aber deutlich in den Attributen und der Behandlung der Auswahl.


```
<card id="Karte" title="Bitte wählen">
  <p>
    <select name="Einheit" value="EUR">
      <option value="DEM">
        Deutsche Mark
      </option>
      <option value="EUR">
        Euro
      </option>
    </select>
  </p>
</card>
```

Abbildung 7.1
Eine einfache Auswahlliste

In Abbildung 7.1 sehen Sie eine einfache Auswahlliste samt Darstellung. Sie wird umschlossen vom <select>-Tag. Das Attribut name legt fest, an welche Variable die Auswahl gebunden wird – in diesem Fall an die Variable Einheit. In value können Sie einen Vorgabewert einstellen, auf den die Variable gesetzt wird, falls sie vorher keinen Wert enthielt.

Innerhalb von <select>...</select> sind die Auswahlmöglichkeiten mit <option> markiert. Das Element umschließt jeweils den Text, der für die Option dargestellt wird. Im Beispiel ist mit dem Attribut value für jede Option der Wert definiert, den die Variable Einheit nach Anwahl trägt. Wenn wie hier die zweite Option ausgewählt wird, hat sie den Wert EUR.

<select>

<select> kann folgende Attribute tragen:

- title: Ein Hilfstext, den der Microbrowser während der Auswahl geeignet verwenden kann.
- name: Der Name der Variablen, in die der Wert der Auswahl geschrieben wird.

- `value`: Ein Vorgabewert, auf den die Variable aus `name` gesetzt wird, falls sie noch keinen Wert hat. Der entsprechende `<option>`-Eintrag wird in der Darstellung vorausgewählt. Im obigen Beipiel schaltet `value="EUR"` die zweite Option ein.

- `iname`: Der Name einer Variablen, in die die Position der Auswahl in der Liste geschrieben wird. Dieser Wert ergibt sich aus der textuellen Reihenfolge der `<option>`-Elemente, wobei von 1 ab gezählt wird (das `i` in `iname` steht für »Index«). Im obigen Beispiel hat also die Auswahl »Deutsche Mark« den Index 1 und »Euro« die 2. Bei den Mehrfachauswahlen, die Sie unten beschrieben finden, kann es sein, daß der Nutzer keine Option gewählt hat. In diesem Fall enthält die Variable `iname` hinterher den Wert 0.

  Bei der Bestimmung der Vorauswahl überprüft ein Microbrowser zunächst das Vorhandensein und die Gültigkeit von `iname` und verwendet die dort genannte Variable für die Vorauswahl. War die Variable nicht mit einem Wert belegt, übernimmt der Browser den Wert von `ivalue`.

  Falls `iname` nicht vorhanden oder ungültig ist, werden `name` und `value` entsprechend verwendet. Ist keines des genannten Attribute gesetzt, wählt der Microbrowser die erste Option – also die mit Index `1` – an.

- `ivalue` ist der Vorgabewert für die Auswahl, allerdings als Indexnummer. Der Microbrowser ignoriert alle ungültigen Werte, z.B. negative Zahlen oder zu große Indexwerte.

- `multiple`: Ist das Attribut auf den Wert `true` gesetzt, dann kann ein Nutzer mehrere Optionen gleichzeitig auswählen, bei `false` nur eine. Sie kennen dieses Verhalten vielleicht von HTML-Formularen mit ihrer Unterscheidung von Check- und Radioboxen.

  Diese Möglichkeit beeinflußt stark die Art, in der Vorgabewerte gesetzt werden, sowie die Repräsentation des Auswahlergebnisses in Variablen. Da hier mehrere Werte gleichzeitig gelten können, ist eine entsprechende Schreibweise notwendig: Mehrere Vorgabe- oder Ergebniswerte werden durch Strichpunkte (`;`) getrennt dargestellt. Der Microbrowser überprüft hier die Gültigkeit der Vorgabewerte und ignoriert beispielsweise doppelte Angaben.

In Abbildung 7.2 finden Sie ein Beispiel. Mit dem Inhalt von `value` werden die erste und letzte Option vorausgewählt. Alternativ dazu hätte man im Beispiel `ivalue="1;3"` verwenden können. Falls der Nutzer beispielsweise die zweite Option zusätzlich markiert und »Bausparen« abwählt, hat hinterher die Variable `Info` den Wert `Konto;Depot` und eine über `iname` definierte Variable den Wert `1;2`.


```
<card id="Karte"
 title="Schicken Sie Info zu">
  <p>
    <select name="Info" value="Konto;Bau">
      <option value="Konto">
        Kontoeröffnung
      </option>
      <option value="Depot">
        Depoteröffnung
      </option>
      <option value="Bau">
        Bausparen
      </option>
    </select>
  </p>
</card>
```

Abbildung 7.2
Eine Liste mit mehrfacher Auswahl

Der obige Mechanismus zur Vorauswahl von Listenfeldern gilt entsprechend für Mehrfachauswahlen. Der Unterschied ist nur, daß eben mehrere Optionen gesetzt werden können. Ist keines der Attribute `iname`, `ivalue`, `name` oder `value` geeignet zu verwenden, wählt der Browser im Gegensatz zur Einzelauswahl keine Option an – der Auswahlindex ergibt dann zunächst `0`.

- `tabindex` erlaubt es, eine bestimmte Navigationsreihenfolge von Eingabefeldern festzulegen. Es kann sein, daß das WAP-Gerät eine zusätzliche Möglichkeit anbietet, zwischen Eingabefeldern schnell hin- und herzuschalten. In HTML-Browsern ist dies oft mit der Tabulator-Taste möglich – daher der Name des Attributs.

  Normalerweise schaltet eine solche Navigation durch die Eingabefelder in der Reihenfolge, in der sie im WML-Text auftre-

ten. Mit dem Attribut `tabindex` können Sie diese Reihenfolge ändern. Es enthält als Wert eine Zahl, die als relative Position in der aktuellen Card zählt. Ein Feld mit einem höheren Wert soll vom Microbrowser erst nach Feldern mit niedrigerem Wert angesteuert werden. Felder ohne dieses Attribut werden vom Microbrowser selber in der Reihenfolge angeordnet, allerdings erst nach jenen mit dem Attribut.

✘ Mit `tabindex` können Sie Formulare einfacher und schneller benutzbar machen. Dabei sollten Sie zunächst durch vorgeschriebene Felder schalten lassen und optionale Eingaben erst danach vorsehen, indem Sie dort `tabindex` nicht verwenden. Bedenken Sie aber, daß die Verwendung dieses Attributs durch das WAP-Gerät nicht zwingend vorgeschrieben ist.

Neben diesen Attributen können Sie bei `<select>` das Attribut `xml:lang` für die Sprachauswahl sowie die Standardattribute `id` und `class` verwenden.

`<option>`

Mit `<option>` markieren Sie die einzelnen Auswahlmöglichkeiten, wobei das Element jeweils die darzustellende Beschreibung der Option umfaßt. Im Gegensatz zu HTML-Formularen kann in diesem Text kein weiteres Markup zur Formatierung des Optionstextes verwendet werden.

Das Attribut `value` trägt den Wert, der bei Auswahl an die in `name` bei `<select>` benannten Variablen zugewiesen wird.

Falls ein Ausgabegerät nicht den umschlossenen Optionstext darstellt, kann der Inhalt des Attributs `title` bei der Darstellung des Auswahlpunkts verwendet werden.

`onpick` kann die Navigation zu einer anderen Karte auslösen, wenn der Nutzer das Element an- oder abwählt. Als Wert trägt es eine URL, also einen Verweis auf ein anderes Deck, eine Karte darin oder auf eine Karte im aktuellen Stapel. Sie können dieses Ereignis auch mit `<onevent>` an eine Aktion binden – die Details dazu finden Sie in Kapitel 8 auf Seite 67.

Bei einer Liste mit Mehrfachauswahl – das Attribut `multiple` von `<select>` hat den Wert `true` – erfolgt diese Navigation sowohl bei An- als auch bei Abwahl der Option. Bei Listen mit einfacher Auswahl nur bei der Anwahl.

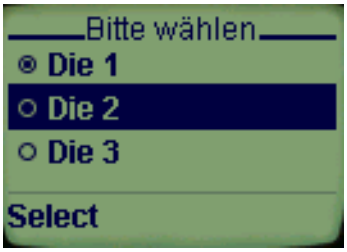
In Abbildung 7.3 auf der nächsten Seite finden Sie ein Beispiel für `onpick`. Wenn in der Auswahlliste eine der drei Möglichkeiten angewählt wird, springt der Browser sofort zur entsprechenden Karte.

```
<card id="auswahl" title="Bitte wählen">
  <p><select>
    <option onpick="#eins">Die 1</option>
    <option onpick="#zwei">Die 2</option>
    <option onpick="#drei">Die 3</option>
  </select></p>
</card>

<card id="eins" title="Ihre Wahl">
  <p align="center">
    <big>1</big>
  </p>
</card>

<card id="zwei" title="Ihre Wahl">
  <p align="center">
    <big>2</big>
  </p>
</card>

<card id="drei" title="Ihre Wahl">
  <p align="center">
    <big>3</big>
  </p>
</card>
```



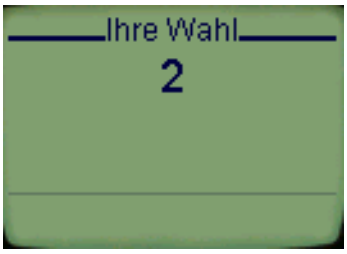


Abbildung 7.3
Eine Auswahlliste mit unmittelbarer Navigation

Wie fast immer, können Sie auch hier die Attribute `xml:lang`, `id` und `class` verwenden.

7.2 Geschachtelte Auswahllisten

Bei langen Auswahllisten ist es ausgesprochen schwierig, mit einem Blick alle Optionen zu erfassen. Und auch die Navigation in einer solchen Liste ist umständlich. Eine logische Gruppierung von Auswahlmöglichkeiten bietet sich an, um Überblick und Navigation zu schaffen.

<optgroup>

Dafür ist in WML das Element `<optgroup>` vorgesehen, das Sie innerhalb von `<select>` verwenden können. Es dient als logi-

sche Zusammenfassung mehrerer Optionen, die vom Microbrowser geeignet bei der Darstellung genutzt werden können.

Abbildung 7.4 zeigt ein Beispiel, in dem sechs Auswahlmöglichkeiten in zwei Gruppen zusammengefaßt wurden. Das Attribut `title` gibt den Namen der Optionengruppe für die Darstellung an. Im Beispiel verwendet der Browser ihn, um in der Auswahlliste die Gruppe zu benennen und bei der Darstellung der Gruppe eine Display-Überschrift anzuzeigen.

```
<card id="karte" title="Sie wünschen:">
  <p>
    <select name="Auswahl">
      <optgroup title="Infomaterial">
        <option value="Konto">
          Kontoeröffnung
        </option>
        <option value="Depot">
          Depoteröffnung
        </option>
        <option value="Bau">
          Bausparen
        </option>
      </optgroup>
      <optgroup title="Börsenindizes">
        <option value="DAX">
          DAX
        </option>
        <option value="Nikkei">
          Nikkei
        </option>
        <option value="SP500">
          S&amp;P 500
        </option>
      </optgroup>
    </select>
  </p>
</card>
```

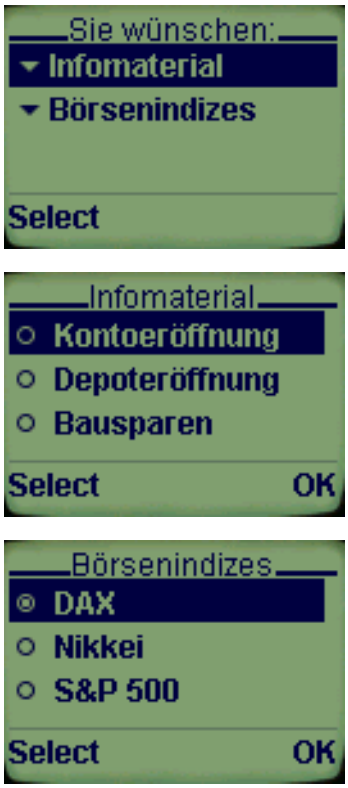


Abbildung 7.4
Eine hierarchische Auswahlliste

Neben `title` können Sie `xml:lang` und die Standardattribute `id` und `class` verwenden.

In Abbildung 7.5 sehen Sie, daß das Tag `<optgroup>` an jeder Stelle innerhalb von `<select>` verwendbar ist, an der `<option>` auftreten kann.

```
<card id="Karte" title="Währung">
  <p>
    <select name="Einheit">
      <option value="Y">Yen</option>
      <optgroup title="Euroland">
        <option value="DEM">D-Mark</option>
        <option value="L">Lira</option>
      </optgroup>
      <option value="USD">
        US Dollar
      </option>
    </select>
  </p>
</card>
```

Abbildung 7.5 *Eine gemischte Auswahlliste*

Falls es die inhaltliche Bedeutung der Auswahlmöglichkeiten erlaubt, kann man mit den Optionsgruppen auch Navigationsebenen einbauen, wie Abbildung 7.6 auf der nächsten Seite zeigt. Hier sind zwei `<optgroup>`-Ebenen verschachtelt und der Nutzer navigiert zunächst durch die Gruppen, um schließlich zu den Optionen zu gelangen.

✘ Auch bei der Strukturierung mit Gruppen bilden alle Optionen in `<select>` eine einzige Auswahlliste. Bei einer Einfachauswahl schaltet also die Anwahl einer Option in einer Gruppe die vorherige Auswahl ab, auch wenn sie in einer anderen Gruppe stand.

7.3 Eingabefelder

<input>

Zur Eingabe von Texten oder anderen Zeichenfolgen ist das Element `<input>` gedacht. Der Nutzer kann es in einem Handy anwählen und dann mit je nach Gerät unterschiedlichem Komfort Eingaben per Tastenfeld vornehmen.

Die Inhalte und eine Vorbelegung des Feldes ist an eine Variable gebunden, deren Namen Sie mit dem vorgeschriebenen `name`-

```
<card id="karte" title="Sie wünschen:">
  <p>
    <select name="Auswahl">
      <optgroup title="Infomaterial">
        <optgroup title="Privatkunden">
          <option value="Konto">
            Kontoeröffnung
          </option>
          <option value="Depot">
            Depoteröffnung
          </option>
          <option value="Bau">
            Bausparen
          </option>
        </optgroup>
        <optgroup title="Firmenkunden">
          <!-- weitere Optionen hier -->
        </optgroup>
      </optgroup>
      <optgroup title="Börsenindizes">
        <!-- weitere Optionen hier -->
      </optgroup>
    </select>
  </p>
</card>
```

Abbildung 7.6
Eine geschachtelte Auswahlliste

Attribut angeben. Ist die Variable belegt, so erscheint ihr Inhalt als Vorgabe im Eingabefeld und kann editiert werden.

Sie können mit dem Attribut `value` einen Initialwert definieren, den das WAP-Gerät der Variable `name` zuweist, falls sie nicht belegt ist. Geben Sie keinen Initialwert an und war die Variable unbelegt, ist das Eingabefeld zunächst leer.

Das Attribut `type` legt fest, ob die Eingaben in dem Feld auf dem Display sichtbar sein sollen – etwas, das Sie beispielsweise bei Zugangskennungen sicherlich vermeiden wollen. Hat das Attribut den Wert `password`, macht das WAP-Gerät alle Eingaben unleserlich. In Abbildung 7.7 auf der nächsten Seite sehen Sie ein Beispiel dafür – alle Ziffern wurden durch Sternchen ersetzt. Fehlt

das Attribut oder hat es den Wert `text`, bleiben alle Eingaben normal zu lesen.


```
<p>
  Nummer:
  <input name="nummer"  type="text"/><br/>
  PIN:
  <input name="pin" type="password"/>
</p>
```

Abbildung 7.7
Offene und geheime Eingabefelder

✘

Während dieser Schutz vor Ausspähung von Passworten auf den ersten Blick sinnvoll ist, lassen sich aber auch gute Argumente gegen die Verwendung von `type="password"` finden. So ist die Verwendung einer Handytastatur für solche Eingaben ohne Feedback im Display extrem schwierig, wenn durch mehrere Tastendrücke durch eine Liste von Buchstaben geschaltet werden muß. Es scheint nur mit sehr hoher Konzentration möglich, die richtigen Zeichen zu finden. Und daneben ist fraglich, ob die Ausspähung von Passworten auf einem Handydisplay überhaupt praktisch möglich ist und ein direkter Schutz durch physisches Abschirmen der Anzeige durch den Nutzer nicht den gleichen Zweck erfüllt. Aus praktischen Gründen könnte man also von `password`-Feldern generell abraten.

Die Eingabe eines Textfelds geschieht oft in einer gesonderten Darstellung im Handy. Mit dem Attribut `title` können Sie eine Zeichenkette angeben, die der Microbrowser dabei zur Information verwenden kann. In Abbildung 7.8 auf der nächsten Seite sehen Sie eine solche Eingabedarstellung ohne und mit Verwendung von `title`.

Mit dem Attribut `size` steuern Sie die Darstellung des Feldes im Display. Es enthält die Breite des Eingabefeldes in Zeichen. Der Microbrowser kann das Attribut allerdings ignorieren. `maxlength` regelt die maximale Länge der Eingabe, die das WAP-Gerät auch beachten muß. Der Wert des Attributs ist die Anzahl der akzeptablen Zeichen.

Die Einschränkung der Eingabelänge kennen Sie vielleicht schon aus HTML. WML bietet darüber hinaus mit dem `format`-Attribut eine Möglichkeit zur Festzulegung, welche Zeichen an

```
<p>
  Nummer:
  <input name="nummer" type="text"/><br/>
  PIN:
  <inputname="pin" type="password" tit-
le="Ihre PIN" />
</p>
```

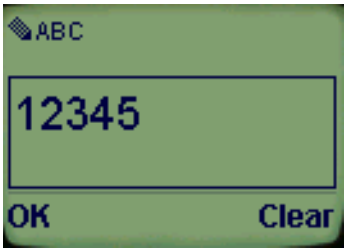


Abbildung 7.8
Eingabefelder ohne und mit `title`

welcher Stelle eingegeben werden dürfen. Dadurch kann schon der Microbrowser ungültige Eingaben vermeiden und muß erst mit korrekten Daten eine Verbindung zum Server aufbauen.

Dazu legt für jede Stelle der Eingabe ein Code-Buchstabe die Menge der erlaubten Zeichen fest. Sie finden in Abbildung 7.9 eine tabellarische Aufstellung dieser Zeichen.

Code	Erlaubte Zeichen
m	Beliebige Zeichen, Kleinbuchstaben wahrscheinlich
M	Beliebige Zeichen, Großbuchstaben wahrscheinlich
N	Alle Ziffern von 0 bis 9
x	Alle Kleinbuchstaben
X	Alle Großbuchstaben
a	Alle Kleinbuchstaben und Interpunktionszeichen, also alle Zeichen, die weder Buchstaben noch Ziffern sind
A	Alle Großbuchstaben und Interpunktionszeichen, also alle Zeichen, die weder Buchstaben noch Ziffern sind

Abbildung 7.9
Die Codes für das Eingabeformat

Die Codebuchstaben unterscheiden drei Gruppen von Zeichen: Ziffern, Buchstaben und Interpunktionszeichen. Die ersten beiden Formatcodes stehen für beliebige Zeichen, sie geben dem WAP-Gerät aber den Hinweis, daß Klein- oder Großschreibung zu erwarten ist. Diesen Hinweis kann das Gerät geeignet verwenden, beispielsweise, indem auf der Handytastatur zunächst Kleinbuchstaben angeboten werden oder eine virtuelle Feststelltaste für Großbuchstaben vorausgewählt ist.

Bei der Codierung für die anderen Zeichengruppen geben Sie mit der Schreibweise des Codebuchstabens die geforderte Groß- oder Kleinschreibung der Eingabe an. Um die Eingabe einer 8-stelligen Kontonummer zu verlangen, könnte man also die folgende Markierung verwenden:

```
<input type="text" format="NNNNNNNN">
```

Soll die Eingabe eine 5-stellige Postleitzahl mit Länderkennung sein (`D10567` oder `E13021`), wäre die richtige Markierung

```
<input type="text" format="ANNNNN">
```

Die Eingabe mehrerer Zeichen der gleichen Zeichenkategorie läßt sich abkürzen, indem dem Codebuchstaben eine Ziffer von `1` bis `9` vorangestellt wird. Die obige Eingabe einer Kontonummer könnte man also auch so auszeichnen:

```
<input type="text" format="8N">
```

WML 1.2 In WML 1.2 wurde die Bedeutung dieses Codebuchstabens leicht geändert. Während vorher bei x`N` genau x Zeichen verlangt waren, besagt die nun gültige Definition, daß bis zu x Zeichen, also auch weniger oder gar keines als Eingabe gefordert sind.

Eine Variante dieser Codierung läßt sich dann verwenden, wenn die Eingabe von beliebig vielen Zeichen einer Kategorie beendet werden kann. Falls Sie zulassen wollen, daß die obigen Postleitzahlen aus beliebig vielen Ziffern bestehen können, würden Sie schreiben:

```
<input type="text" format="AA*N">
```

Die `*`-Variante darf nur einmalig im Wert des `format`-Attributs verwendet werden und muß der letzte Code in der Formatdefinition sein.

Die Voreinstellung für `<input>`-Tags ohne `format`-Attribut ist übrigens `*M`, also beliebig viele beliebige Zeichen, wobei eher Großbuchstaben zu erwarten sind. ✘

Das Postleitzahlenbeispiel soll für eine letzte Variante der Eingabeformate dienen. In der Realität wird das Länderkürzel ja mit einem Bindestrich von der Zahl abgegrenzt. Sie können dafür in einem Eingabeformat konstante Zeichen vorgeben, die der Microbrowser darstellt, aber nicht als Eingabefeld zuläßt. Solche Zeichen im Format leitet der Rückstrich \ ein:

```
<input type="text" format="AA\-NNNNN">
```

Im Normalfall ist bei einem Format `NNNNN` auch die Eingabe von fünf Ziffern verlangt. Falls dies nicht notwendig sein soll, müssen Sie dies mit dem Attribut `emptyok` markieren.

Wie schon bei `<select>`, legt das `tabindex`-Attribut die relative Position des Eingabefeldes für eine schnelle Navigation fest

WML 1.2

Mit WML 1.2 wurde auch bei `<input>` das Attribut `accesskey` eingeführt. Wie bei `<a>` und `<anchor>` enthält es als Wert ein einzelnes Zeichen, das als Tastenkürzel dient. Wird es eingegeben, springt der Microbrowser zu diesem Eingabefeld, das damit einfacher zu erreichen ist.

Neben diesen spezifischen Attributen können Sie wieder mit `xml:lang` die Sprache für das Feld festlegen und die Standardattribute `id` und `class` verwenden.

In diesem Kapitel haben Sie gesehen, wie Sie mit WML Formulare konstruieren. Die beiden Feldarten, Auswahl- und Eingabefelder, bieten durch eine Reihe von Attributen viel Spielraum für die Gestaltung von interaktiven Cards.

Im folgenden Kapitel lernen Sie die Möglichkeiten mit WML kennen, auf Eingabereignisse zur reagieren und andere Aktionen am WAP-Gerät zu steuern. Dabei geht es auch um die Übermittlung von Formulareingaben an einen Web-Server.

8 Ereignisse und Aktionen

Wie schon bei den Formularen sichtbar, legt WML besonderen Wert darauf, möglichst viele Interaktionen ohne weitere Netzzugriffe zu behandeln. Dementsprechend definiert die Auszeichnungssprache neben dem Darstellungsmodell für Seiten zusätzlich ein Ereignismodell, mit dem der Microbrowser autonom auf die Anwahl von Steuerelementen oder andere Ereignisse reagieren kann. Sie lernen in diesem Kapitel die verschiedenen Arten von Ereignissen und Aktionen kennen.

8.1 Steuerelemente

Während bei einem PC mit einem Web-Browser die Maus das wichtigste Eingabegerät ist, stehen bei einem Handy in der Regel nur eine numerische Tastatur und einige wenige Spezialtasten zur Verfügung. Die Navigation durch Menüs ist damit ausgesprochen langwierig und umständlich. Daher können Sie mit WML bestimmte Aufgaben an Steuerelemente binden.

Ein WAP-Gerät kann solche Steuerelemente auf ganz unterschiedliche Weisen anbieten. So können bestimmte Tasten für sie reserviert sein, eine Spracheingabe könnte zu ihrer Auswahl dienen, mit einem Stift antippbare Buttons im Display wären denkbar oder auch eine Menüsteuerung.

<do>

Das Element `<do>` belegt ein Steuerelement mit einer Reaktion des WAP-Geräts. Es kann innerhalb einer Karte auftreten und definiert eine Bindung eines bestimmten Steuerelements an eine bestimmte Aktion – im Englischen *Task* genannt.

Das Beispiel in Abbildung 8.1 auf der nächsten Seite zeigt die Funktionsweise. Der Stapel enthält drei Karten, die mit `die3`, `die2` und `die1` benannt sind und jeweils eine große `3`, `2` und `1` darstellen.

Bei jeder Karte legt ein `<do>`-Element das Verhalten des WAP-Geräts bei unterschiedlichen Steuerelementen fest. Bei der Karte `die3` legt das Attribut `type` mit dem Wert `accept` fest, daß die

```
<wml>
  <card id="die3" title="Countdown">
    <do type="accept" label="Weiter">
      <go href="#die2"/>
    </do>
    <p align="center">
      <big>3</big>
    </p>
  </card>
  <card id="die2" title="Countdown">
    <do type="accept" label="Weiter">
      <go href="#die1"/>
    </do>
    <do type="prev" label="Zurück">
       <go href="#die3"/>
    </do>
    <p align="center">
      <big>2</big>
    </p>
  </card>
  <card id="die1" title="Countdown">
    <do type="prev" label="Zurück">
      <go href="#die2"/>
    </do>
    <p align="center">
      <big>1</big>
    </p>
  </card>
</wml>
```

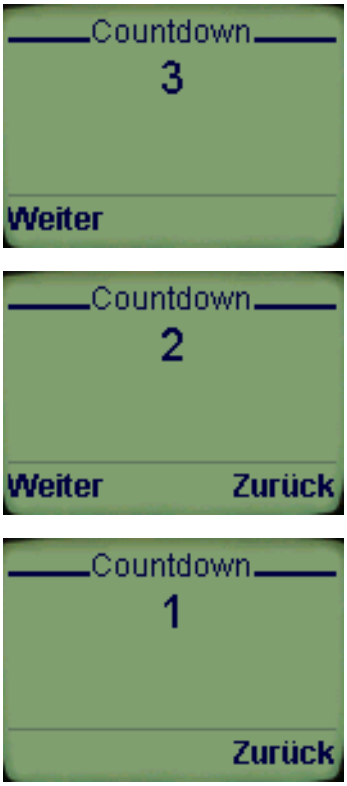


Abbildung 8.1
Durchtippen in einem Kartenstapel

Annahmetaste – oder ein entsprechendes Steuerelement – die folgende Aktion auslösen soll. Bei vielen Handys ist die Annahmetaste links unter dem Display.

Das Drücken der Taste löst die Ausführung der Aktion aus, die innerhalb von `<do>` definiert ist. Im Beispiel handelt es sich um eine Navigationsaktion, die mit dem Element `<go>` markiert ist. Das Attribut `href` enthält das Ziel der Navigation: Die Karte `die2`.

Dort gibt es zwei <do>-Elemente, davon bestimmt das eine mit dem type-Wert accept wieder die Navigation zur nächsten Karte, die3. Das zweite <do> legt mit dem Wert prev im type-Attribut die Aktion für die Zurück-Taste fest, die oft rechts unter dem Display zu finden ist: Eine erneute <go>-Aktion – allerdings zur Karte die3. Auf der Karte die1 gibt es nur eine Aktion für Zurück: Navigation zur Karte die2.

<do> kann folgende Attribute tragen:

- type bindet die <do>-Aktion an eine bestimmte Eingabeart und muß angegeben sein. Im WML-Standard sind die Namen in der folgenden Tabelle definiert:

Steuerelement	Bedeutung
accept	Bestätigung einer Auswahl
delete	Löschen einer Auswahl
prev	Navigation zurück
help	Hilfestellung
options	Hilfestellung über Auswahl
reset	Rücksetzen
unknown	Unbekannte Eingabe
vnd.*	Herstellerspezifische Eingabe

  Während dieser kleine Satz von Namen im WML-Standard vordefiniert ist, können Hersteller eigene Steuerelemente einbinden, die durch spezielle <do>-Aktionen handhabbar sind. Die Namen dieser Eingaben müssen mit den Zeichen vnd. (von »Vendor«) in beliebiger, aber relevanter Groß- und Kleinschreibung beginnen, also beispielsweise vnd.menu für eine herstellerspezifische »Menü«-Taste.

  Entstehen solche Eingaben, ohne daß für sie ein passendes <do> existiert, nennt sie das WAP-Gerät in unknown um und sucht ein entsprechendes <do>. Wird keines gefunden, passiert nichts.

- Mit label legen Sie eine Zeichenkette fest, die das WAP-Gerät geeignet für diese Eingabe anzeigen kann. In Abbildung 8.1 auf der vorherigen Seite hatten Sie gesehen, wie die Beschriftungen für die accept und prev Steuerelemente aussehen können.

- name vergibt einen Namen an die Bindung von Steuerelementen an eine Aktion. Pro Karte darf ein solcher Name nur einmalig auftreten. Er wird in Kombination mit <template> in Abschnitt 8.8 auf Seite 82 relevant.
- optional bestimmt die Verbindlichkeit der Aktionsbindung. Bei true muß das WAP-Gerät die Vorgabe beachten und dem Nutzer beispielsweise in einem Menü die Steuerelemente anbieten, für die eine Aktionsbindung besteht. Bei false kann es sie ignorieren. Eine Anwendung dafür ist beispielsweise das Ausblenden aller vnd-Typen auf einem Handy eines anderen Herstellers.

Als Standardattribute kann <do> wiederum xml:lang, id und class tragen.

8.2 Navigationsaktionen

Mit <do> werden Aktionen – die *Tasks* – an Steuerelemente gebunden. In WML sind vier solcher Tasks definiert, die jeweils als eigenes Tag innerhalb von <do>...</do> notiert werden müssen.

<go>

Mit <go> navigiert das WAP-Gerät zu einer neuen Karte. Das Ziel der Navigation muß im notwendigen href-Attribut als URL vermerkt sein. Dabei kann es sich um eine Karte aus dem aktuellen Stapel handeln oder um eine Karte, deren Stapel neu geladen werden muß.

Im HTTP-Protokoll kann bei einer neuen Seitenanfrage dem Server mitgeteilt werden, auf welcher Seite ein Verweis stand. Dieser »Referer« läßt sich serverseitig sehr gut zur Erstellung von Statistiken, für Abrechnungsschemata bei angeklickten Werbeeinblendungen und auch für das Nachvollziehen des Navigationsverhaltens von Nutzern verwenden.

Während ein HTML-Browser selber entscheidet, ob er den Referer an den Server schickt, kann man einen Microbrowser durch das Attribut sendreferer bei <go> dazu zwingen. Ist der Attributwert gleich true, muß die Referer-URL der Seitenanfrage beigelegt werden, beim Wert false nicht.

In Abbildung 8.2 auf der nächsten Seite sehen Sie zwei Beispiele für die Verwendung von <go>. Auf der Karte erste ist in einem Anker mit <anchor> die Navigation zur Kart zweite festgelegt, wenn der Link ausgewählt wird. Natürlich könnte man das gleiche Verhalten mit der Abkürzung <a> erreichen.

```
<card id="erste" title="Willkommen">
  <p>Homepage<br/>
    <anchor>Weiter<
      go href="#zweite"
    /></anchor>
  </p>
</card>
<card id="zweite" title="Infos">
  <do type="accept" label="Heim">
    <go href="#erste"/>
  </do>
  <p>Die Welt ist rund</p>
</card>
```

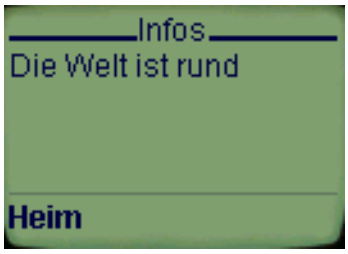


Abbildung 8.2
Navigationsaktionen

✘ Die auf den ersten Blick seltsame Formatierung des WML-Textes sorgt dafür, daß keine überzähligen Leerzeichen entstehen, durch die die Darstellung des Ankertextes sehr unschön würde.

Auf der Karte `zweite` wird eine `<go>`-Aktion an die Annahmetaste per `<do>` gebunden. Mit dem `label`-Attribut erhält sie in der Anzeige die Beschriftung »Heim«. Das Ziel der Navigationsaktion ist wieder die erste Karte.

8.3 Datenfelder an Server übermitteln

Interaktive Informationsangebote beruhen darauf, daß Daten beim Klienten – dem WAP-Gerät – erfaßt und an einen Server zur weiteren Verarbeitung übermittelt werden.

Für `<go>` lassen sich Daten definieren, die automatisch mit der Seitenanforderung an einen Server geschickt werden sollen. Sie sind innerhalb der `<go>...</go>`-Markierung mit dem Tag `<postfield>` ausgezeichnet. Neben den optionalen Standardattributen `id`, `class` müssen zwei Attribute vorhanden sein:

`<postfield>`

- `name` beinhaltet den Namen des Feldes, unter dem sein Inhalt bei Weitergabe der Daten übertragen wird.
- `value` ist der Wert des Feldes.

Bei folgendem Code-Ausschnitt wird `<postfield>` benutzt, um einen Namen und eine Kontonummer an den Server zu übermitteln:

```
<anchor>Zum Konto<
  go href="http://geld.foo/info">
  <postfield name="konto" value="1324"/>
  <postfield name="name" value="Adam Smith"/>
  </go>
</anchor>
```

In dem mit »Zum Konto« beschrifteten Anker ist eine Navigationsaktion mit `<go>` markiert, die zum Server `geld.foo` führt. Mit `<postfield>` werden zwei Datenfelder für diese Navigation belegt, und zwar `konto` mit dem Wert `1324` und `name` mit dem Wert `Adam Smith`.

Bevor der Microbrowser die `<go>`-Aktion durchführt, codiert er im Normalfall diese Werte in die Netzadresse und lädt danach die Seite über die URL `http://geld.foo/info?Konto=1324&Name=Adam%20Smith`. Server `geld.foo` wird der URL-Teil hinter dem Fragezeichen abgetrennt und die Datenfelder extrahiert.

Sie sehen, daß bei der Codierung beispielsweise Leerzeichen konvertiert werden. Die genauen Regeln zur Codierung von URLs finden Sie im Internet Standard RFC 2396 ([4]) beschrieben.

`<go>` kann auf sehr unterschiedliche Weise mit den Datenfeldern umgehen. `method` bestimmt die Art der Übermittlung von Daten aus Formulareingaben an einen Server. Die zwei möglichen Übertragungsarten sind Ihnen vielleicht schon aus dem Web bekannt und folgen den Regeln des HTTP-Protokolls.

Beim Wert `get` schickt das Gerät lediglich die Anforderung einer neuen Seite an den Server. Eventuelle Eingabedaten sind dabei an die übergebene URL angehängt, wie im obigen Beispiel.

Diese Methode ist schlecht für die Übertragung großer oder binärer Datenmengen geeignet, da die URL-Codierung sehr aufwendig ist. Sie führt auch zu extrem langen Netzadressen, die die vorgesehenen Maximalgrößen bei Servern weit übersteigen können. Mit `post` als Wert von `method` wählt man eine Alternative aus.

Hier sendet das WAP-Gerät eine Mitteilung an den Server, bei dem Formulareingaben nicht in die URL codiert werden, sondern direkt in der eigentlichen Mitteilung. Der Microbrowser fordert die URL `geld.foo/info` an, schickt aber in der Anforderungsmitteilung die Zeichenkette `Konto=1324&Name=Adam%20Smith`.

accept-charset legt fest, welche Zeichensätze der Server beherrschen muß, um Eingaben zu verarbeiten. Der Inhalt des Attributs ist eine Liste solcher Zeichensätze, die mit Kommas getrennt sind. Mindestens einer davon muß dem Server bekannt sein.

Um Eingaben mit Umlauten abzusenden, könnten Sie »ISO-8859-1,ISO-10646-Unicode-Latin1« verwenden, da beide Zeichensätze Umlaute umfassen. Der Wert US-ASCII wäre nicht geeignet, da hier keine Umlaute vorgesehen sind. Die Namen der Zeichensätze sind standardisiert. Sie finden eine aktuelle Liste unter der Adresse http://www.isi.edu/in-notes/iana/assignments/character-sets.

Falls das Attribut fehlt, sollte der Microbrowser den Zeichensatz verwenden, der zur Übertragung des Kartenstapels benutzt wurde.

WML 1.2

Im mit WML 1.2 eingeführten Attribut enctype kann zusätzlich zu method die Codierung der Feldinhalte als Medientyp vermerkt werden. Bei Angabe des Standardwerts application/x-www-form-urlencoded ist die Codierung innerhalb der URL bestimmt, während beim Wert multipart/form-data die einzelnen Felder innerhalb der Mitteilung codiert sind. Ist method mit get belegt, kann für enctype lediglich der Wert application/x-www-form-urlencoded gewählt werden.

Über diese Attribute hinaus sind wieder die id und class bei <go> möglich.

8.4 Übermittlung von Formulareingaben

Das obige Beispiel gibt den Inhalt der Datenfelder fest vor, was sehr unrealistisch ist. Tatsächlich wird man ein Formular zur Eingabe der Daten erstellen und die dortigen Werte als Daten übergeben. Das könnte wie folgt aussehen:

```
<input name="name" type="text"/><br/>
<input name="konto" type="text"/><br/>
<anchor>Zum Konto<
  go href="http://geld.foo/info">
  <postfield name="konto" value="$konto"/>
  <postfield name="name" value="$name"/>
  </go>
</anchor>
```

Die beiden Eingabefelder erfragen die Eingaben und speichern sie in den WML-Variablen $konto und $name. Um sie als Datenfelder

bereit zu halten, sind wiederum `<postfield>`-Markierungen notwendig, nur erhalten die so definierten Felder ihren Wert nun aus den jeweiligen WML-Variablen, auf die in den `value`-Attributen verwiesen wird. Danach geschieht wie gehabt die Codierung als URL und die Übermittlung an einen Server.

Da Variablen ja auch in Attributen verwendet werden können, die eine URL als Wert erwarten, könnte man durch Angabe der Netzadresse `http://geld.foo/info?konto=$konto&Name=$(name:escape)` auch eine entsprechend codierte URL selber zusammenbauen. Die Schreibweise `$(name:escape)` lernen Sie in Abschnitt 9.3 genauer kennen. Sie bewirkt beispielsweise die Codierung des Leerzeichens.

✘ Dieses Vorgehen hat zwei Nachteile. Zum einen müssen Sie sich selber um die Codierung von Datenfeldern in einer URL kümmern – überlassen Sie das besser dem Microbrowser. Zum anderen beschränken Sie damit natürlich die Art der Datenübermittlung an den Server auf die Methode der URL-Codierung. Verwenden Sie `<postfield>`-Markierungen, können Sie über die Attribute `method` und `enctype` bei `<go>` einfach und flexibel auch später die Übermittlungsart wechseln.

8.5 Zurück- und Neudarstellungsaktionen

<prev>

Die Aktion `<prev>` navigiert eine Karte zurück entlang der Nutzungsgeschichte. Sie entspricht damit der Ihnen von Ihrem Web-Browser bekannten »Zurück«- oder »Back«-Aktion. Das Tag kann außer den Standardattributen `id` und `class` keine weiteren Informationen tragen. Falls der Microbrowser schon die Karte am Anfang der Nutzungsgeschichte darstellt, hat die Aktion keine Auswirkung.

✘ Ein Microbrowser kann die Nutzungsgeschichte verkürzen, wenn er anderweitig den dafür verwendeten Speicher im WAP-Gerät nutzen muß. Er wird dabei die ältesten Adressen löschen und versuchen, zumindest zehn Einträge zu erhalten.

<refresh>

`<refresh>` – mit den Standardattributen `id` und `class` – zeigt die aktuelle Karte erneut an. Zweck dieser Aktion ist das Verändern von Variablen und damit Seiteninhalten.

In Abbildung 8.3 auf der nächsten Seite sehen Sie eine Anwendung von `<refresh>`. Wieder sollen Daten für eine Banküberweisung eingegeben werden. Dabei sind alle drei Eingabefelder sichtbar und zusätzlich sollen in der unteren Zeile die Eingaben in der

```
<wml>
  <card id="Eingabe"
   title="">
    <do type="accept">
      <refresh/>
    </do>
    <p>
      Konto:
      <input name="KontoNummer"/><br/>
      BLZ:
      <input name="BLZ"/><br/>
      Betrag:
      <input name="Betrag"/><br/>
      $(KontoNummer)-$(BLZ)-$(Betrag)
    </p>
  </card>
</wml>
```

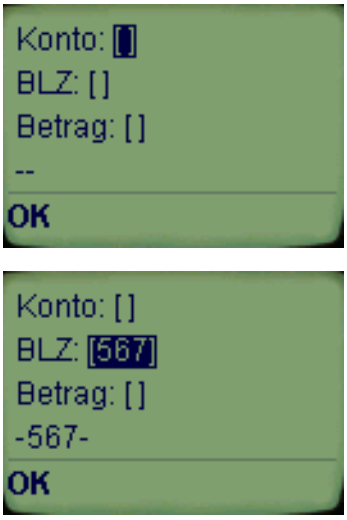


Abbildung 8.3
Nutzereingaben als Variablen

Form *Konto-BLZ-Betrag* stehen. Zum Verständnis des WML-Codes ist ein kleiner Vorgriff auf Kapitel 9 auf Seite 85 notwendig: `$(KontoNummer)` fügt den Inhalt der Variabelen `KontoNummer` in die Seite ein.

Die drei `<input>`-Felder setzen jeweils eine Variable – ihr geänderter Inhalt wird allerdings nur im Eingabefeld dargestellt. Damit auch die untere Zeile aktuell ist, muß sie neu angezeigt werden, damit der Microbrowser die geänderten Variableninhalte verwendet. Genau dafür dient die `<refresh>`-Aktion. Im Beispiel ist sie an die Eingabetaste gebunden und wird damit bei jeder Eingabe ausgeführt. Somit bleibt die unterste Zeile aktuell.

<noop>

Schließlich gibt es noch die mit `<noop>` ausgezeichnete Aktion »no operation«, die nichts tut. Sie kann lediglich die Standardattribute `id` und `class` tragen und dient dazu, Bindungen auf Stapelebene individuell auf Kartenebene wieder auszuschalten – mehr dazu in Abschnitt 8.8 auf Seite 82.

8.6 Standardereignisse

Neben den Steuerelementen können auch anderweitig eintretende Ereignisse Aktionen auslösen. Beispiele sind Navigationsaktionen, der Ablauf eines Stoppuhr oder das Eintreffen eines Anrufes an einem Handy.

In WML sind vier solcher Ereignisse definiert:

- `onenterforward` tritt dann auf, wenn eine Karte durch eine Navigationsaktion betreten, also zum ersten Mal angezeigt wird. Das Ereignis ist damit an einzelne Karten gebunden. Ausgelöst wird es beispielsweise durch die `<go>`-Aktion.
- `onenterbackward` tritt dann auf, wenn eine Navigation zurück entlang der bisherigen Nutzungsgeschichte zu einer Karte führt. Die `<prev>`-Aktion löst unter anderem dieses Ereignis aus. Das Ereignis ist an eine Karte gebunden.
- Das `onpick`-Ereignis hatten Sie schon bei der Beschreibung von `<option>` in Abschnitt 7.1 auf Seite 57 kennengelernt: Es tritt auf, wenn der Nutzer ein Element in einer Auswahllist an- oder abwählt. Dementsprechend ist es an dieses `<option>` gebunden.
- Das `ontimer`-Ereignis schließlich tritt beim Ablauf einer Alarmuhr auf, die Sie mit dem `<timer>`-Tag setzen können. Sie finden die genaue Arbeitsweise in Abschnitt 8.7 auf Seite 78 beschrieben.

Die Reaktionen auf diese Ereignisse können auf zwei Arten in einem WML-Stapel vermerkt werden:

1. Durch Verwendung von entsprechenden Attributen bei den Elementen, an die diese Ereignisse gebunden sind. In WML ist ihr Wert jeweils eine URL und das Eintreten des Ereignisses bewirkt eine Navigation zu dieser Netzadresse. Im Einzelnen sind diese Attribute:
   - `onenterforward` bei `<card>`,
   - `onenterbackward` bei `<card>`,
   - `onpick` bei `<option>` (eine Beschreibung finden Sie in Abschnitt 7.1 auf Seite 57),
   - `ontimer` bei `<card>`.

2. Durch Beschreibung der Reaktion auf das Ereignis durch Aktions-Tags in einer <onevent>-Markierung.

<onevent>

<onevent> beinhaltet eine der vier möglichen Aktionen, die Sie weiter vorne kennengelernt haben: <go>, <prev>, <refresh> und <noop>. Diese konnten nur bei zwei WML-Elementen auftreten – dementsprechend dürfen Sie <onevent> nur innerhalb von <card> oder <option> einsetzen.

<onevent> bezeichnet im notwendig zu verwendenden Attribut type die Art des Ereignisses für das es Aktionen markiert. Die vier möglichen Werte sind: onenterforward, onenterbackward, onpick und ontimer. Die eingeschlossene Aktion soll dann ausgeführt werden, wenn das betreffende Ereignis eintritt. Daneben sind die Standardattribute id und class verwendbar.

Gleichzeitig sind ja gleichnamige Attribute bei <card> und <option> definiert, die eine Navigation zu einer URL bewirken. Damit ist die Schreibweise <card onenterforward="http://x.yy"...>... identisch mit der folgenden Markierung:

```
<card ...>
  <onevent type="onenterforward">
    <go href="http://x.yy"/>
  </onevent>
...
```

Das <onevent>- und das Attributverfahren sind gleichwertig – die Attribute sind lediglich eine Abkürzung für die Verwendung von <onevent> mit einer <go>-Aktion. Der entscheidende Unterschied, der die Flexibilität von <option> ausmacht, ist die Möglichkeit, darin beliebige Aktionen vorzugeben und nicht nur auf <go> festgelegt zu sein, wie bei dem Attributverfahren.

Sie können das nutzen, um eine Karte zu schreiben, die nie angesteuert werden kann, wie in Abbildung 8.4 auf der nächsten Seite gezeigt. Hier bietet die Karte Vorraum eine Navigationsmöglichkeit zur Karte Tresor.

Folgt man diesem Verweis – oder steuert man Tresor anders an – dann tritt allerdings sofort das Ereignis onenterforward auf. Auf der Karte ist dafür mit <onevent> eine Aktion definiert, nämlich <prev/>. Damit gelangt man sofort wieder zu der Karte zurück, auf der man vorher war – Tresor wird also nie angezeigt.

✘

Erweitert man das Beispiel wie in Abbildung 8.5 auf der nächsten Seite, hat man eine regelrechte Falle für einen Microbrowser aufgestellt. Navigiert man über den Vorraum zum Tresor und wird

Abbildung 8.4
Eine nicht ansteuerbare Karte

```
<card id="Vorraum" title="">
  <p>
    <a href="#Tresor">Hier zum Tresor</a>
  </p>
</card>
<card id="Tresor" title="">
  <onevent type="onenterforward">
    <prev/>
  </onevent>
  <p>Bedienen Sie sich!</p>
</card>
```

dort abgewiesen, schickt einen das umgehend eintretende Ereignis `onenterbackward` sofort wieder zur `Tresor`-Karte.

Abbildung 8.5
Eine Falle

```
<card id="Vorraum" title="">
  <onevent type="onenterbackward">
    <go href="#tresor"/>
  </onevent>
  <p>
    <a href="#Tresor">Hier zum Tresor</a>
  </p>
</card>
<card id="Tresor" title="">
  <onevent type="onenterforward">
    <prev/>
  </onevent>
  <p>Bedienen Sie sich!</p>
</card>
```

Die so entstandene »Endlosschleife« muß durch Neustart des Microbrowsers gestoppt werden... Sie sollten Ihre Kreativität natürlich besser für nützlichere Dinge einsetzen!

8.7 Zeitereignisse

Weiter oben war schon das Ereignis `ontimer` angesprochen worden, das auftritt, wenn eine Alarmuhr abläuft. Eine solche Alarmuhr können Sie selber mit dem Element `<timer>` für eine Karte

<timer>

definieren und auf einen Wert stellen. Es darf nur innerhalb von `<card>` verwendet werden und dort nur einmal auftreten.

`<timer>` kann neben den Standardattributen `id` und `class` zwei weitere Attribute tragen. Das immer zu verwendende Attribut `value` bestimmt den Startwert der Uhr in Zehntelsekunden. Mit Anzeigen der Karte zählt der Microbrowser diese Stoppuhr nach unten. Ist die Uhr »abgelaufen«, ihr Zählerstand wechselt also 0, tritt das `ontimer`-Ereignis ein. Es gibt keine Vorgaben über die Genauigkeit der Uhr – sie muß also nicht in Zehntelsekunden rechnen.

Das obige Beispiel der Tresorfalle könnte man noch etwas verschärfen, wenn es wie in Abbildung 8.6 erweitert wird.

```
<card id="Vorraum" title=""
 ontimer="#Tresor">
  <onevent type="onenterbackward">
    <go href="#Tresor"/>
  </onevent>
  <timer value="10"/>
  <p>
    <a href="#Tresor">Hier zum Tresor</a>
  </p>
</card>
<card id="Tresor" title="">
  <onevent type="onenterforward">
    <prev/>
  </onevent>
  <p>Bedienen Sie sich!</p>
</card>
```

Abbildung 8.6
Eine automatische Falle

Zusätzlich ist mit der Attributmethode vermerkt, daß beim Eintreten des `ontimer`-Ereignisses automatisch zur `Tresor`-Karte navigiert werden soll – womit die Endlosschleife beginnt...

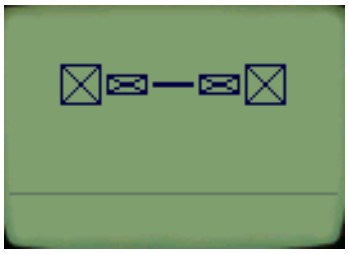
Mit `<timer>` ist ein Zeitgeber auf der Karte vermerkt, der nach 1 Sekunde das Ereignis auslöst. Der Nutzer hat damit noch eine minimale Chance, die `Vorraum`-Karte zu verlassen – bei Untätigkeit muß er aber hinterher sein Gerät zurücksetzen.

✘

Man kann aber auch sinnvollere Dinge mit dem Zeitgeber machen. Im Web sind animierte Grafiken sehr beliebt. Realisiert sind sie zumeist durch eine Variante des GIF-Grafikformats, in der mehrere Bilder mit Angaben über Darstellungslängen gespeichert sind.

In der WAP-Welt existiert momentan nur WBMP als Format in einer Version, die keine Animationen zuläßt.

Mit `<timer>` läßt sich das Manko umgehen, wie in Abbildung 8.7 zu sehen ist: Durch uhrgesteuertes automatisches Laden von Karten wird zwischen unterschiedlichen Grafiken geschaltet. Das Ergebnis ist ein Quadrat, das anscheinend im Raum rotiert.



```
<card id="one" title="Animation"
 ontimer="#two">
  <timer value="1"/>
  <p>
    <img src="square.wbmp" alt="#"/>
  </p>
</card>
<card id="two" title="Animation"
 ontimer="#three">
  <timer value="1"/>
  <p>
    <img src="flatsquare.wbmp" alt="="/>
  </p>
</card>
<card id="three" title="Animation"
 ontimer="#four">
  <timer value="1"/>
  <p>
    <img src="line.wbmp" alt="-"/>
  </p>
</card>
<card id="four" title="Animation"
 ontimer="#one">
  <timer value="1"/>
  <p>
    <img src="flatsquare.wbmp" alt="="/>
  </p>
</card>
```

Abbildung 8.7
Ein animiertes Quadrat

Mit dem zweiten Attribut von `<timer>` – `name` – wird der Zählerstand mit einer Variablen verknüpft. Der Name dieser Variablen ergibt sich aus dem Wert des Attributs. Die Details zu Variablen finden Sie in Kapitel 9 auf Seite 85 beschrieben.

Ist die Variable noch nicht vorhanden, wird sie eingerichtet und initial auf den Wert aus `value` gesetzt. Gibt es sie schon, ignoriert der Microbrowser `value` und benutzt den Wert der Variablen als Startwert für die Uhr.

Beim Verlassen der Karte, auf der sich die Stoppuhr befindet, kopiert der Microbrowser den aktuellen Stand in die benannte Variable. Das erneute Anzeigen einer Karte mit einer `<refresh>`-Aktion gilt dabei als Verlassen. Läuft die Uhr ab, wird die Variable entsprechend auf `0` gesetzt.

In Abbildung 8.8 finden Sie ein Beispiel, in dem die Zeit gemessen wird, die auf der Uhr noch übrig war, bevor das `ontimer`-Ereignis eingetreten wäre.

```
<card id="Vorraum" title=""
 ontimer="#Tresor">
  <timer value="100" name="zeit"/>
  <p>
    <a href="#Tresor">Hier zum Tresor</a><br/>
    (Sie haben $zeit Zehntelsekunden Zeit)
  </p>
</card>
<card id="Tresor" title="">
  <p>Sie hatten $zeit Zehntelsekunden
    übrig...<br/>
    Bedienen Sie sich!
  </p>
</card>
```

Abbildung 8.8
Den Stand der Uhr in einer Variablen vermerkt

Bei `<timer>` wird `zeit` als an den Zähler gebundene Variable deklariert. Bei der ersten Anzeige existiert diese Variable noch nicht und wird mit dem Wert 100 initialisiert. Die Karte enthält eine entsprechende Darstellung.

Die `Tresor`-Seite wird hier entweder durch aktive Navigation oder durch die automatische Weiterschaltung bei Ablauf der Uhr angesteuert. Bevor dies geschieht, schreibt der Microbrowser den Uhrstand nach `zeit`. Dieser wird dann auf der Folgekarte dargestellt.

Navigiert man vor Ablauf der 10 Sekunden, gibt es eine »Restlaufzeit«. Bei einer Navigation zurück zur `Vorraum`-Seite ist diese noch in `zeit` gespeichert und wird anstelle der 100 zur Initialisierung der Stoppuhr benutzt.

8.8 Aktionsschablonen

In Ihrem Informationssystem sollten Sie dafür sorgen, daß eine Einheitlichkeit in Ihrer Nutzeroberfläche besteht. So sollte beispielsweise das Drücken einer Hilfe-Taste entweder nie oder aber immer zu einer Hilfeseite führen.

Um dies zu realisieren, müßte Ihr Kartenstapel so aussehen:

```
<wml>
  <card id="erste" title="Willkommen">
    <do type="help" label="Hilfe">
      <go href="help.wml"/>
    </do>
    ...
  </card>
  <card id="zweite" title="Weiter">
    <do type="help" label="Hilfe">
      <go href="help.wml"/>
    </do>
    ...
  </card>
  <card id="dritte" title="Ok">
    <do type="help" label="Hilfe">
      <go href="help.wml"/>
    </do>
    ...
```

<template>

Sie können die Wiederholungen von <do> vermeiden, indem Sie mit <template> am Beginn des Stapels vor der ersten Karte <do>- und <onevent>-Elemente umschließen, die dann für alle Karten gelten.

Das Beispiel läßt sich damit wie folgt abkürzen:

```
<wml>
  <template>
    <do type="help" label="Hilfe"
        name="Hilfestellung">
      <go href="help.wml"/>
    </do>
  </template>
  <card id="erste" title="Willkommen">
    ...
  </card>
  <card id="zweite" title="Weiter">
```

```
  ...
</card>
<card id="dritte" title="Ok">
  ...
```

Die von `<template>` umschlossenen Aktionsbindungen werden so behandelt, als stünden sie auf jeder Karte nochmals direkt. Neben den eingeschlossenen `<onevent>`-Markierungen können auch die Attribute `onenterforward`, `onenterbackward` und `ontimer` verwendet werden, um jeweils die `<go>`-Aktion an das entsprechene Ereignis zu binden. Daneben kann `<template>` noch die Standardattribute `id` und `class` tragen.

Mit `<template>` gibt es nun einerseits Aktionsbindungen auf Stapelebene, andererseits können in den Karten jeweils individuelle Bindungen vorhanden sein. In WML ist definiert, daß die Bindungen kombiniert werden und dabei die individuelleren Einstellungen auf den Karten aus `<template>` überlagern. Dabei gelten folgende Regeln:

1. Durch Attribute oder `<onevent>` gebundene Aktionen an Ereignisse überlagern die Definition aus einem `<template>`, wenn sie im `type`-Attribut dasselbe Ereignis bezeichnen.

2. Durch `<do>` hergestellte Bindungen zwischen Steuerelement und Aktionen aus einer `<card>` überlagern jene aus einem `<template>`, wenn sie im Attribut `name` denselben Namen haben.

   So könnte man in dem Beispiel für die zweite Karte festlegen, daß die Hilfestellung nicht mehr über die Hilfetaste, sondern über die Options-Taste angesteuert werden soll. Man schreibt dazu wie folgt:

   ```
   <card id="zweite" title="Weiter">
     <do type="options" label="Hilfe"
         name="Hilfestellung">
       <go href="help.wml"/>
     </do>
     ...
   </card>
   ```

   Während in der Schablone unter dem Namen `Hilfestellung` die Hilfe-Taste an die Navigation zur Hilfeseite gebunden wird, ist auf der zweiten Karte die Options-Taste daran

gebunden. Durch die Überlagerung ist auf dieser Karte nun keine Aktion mehr an die Hilfe-Taste gebunden.

✘ Hierdurch wird auch die Rolle der `<noop>`-Aktion klarer: Wenn in einer Schablone eine Aktion an ein Ereignis oder ein Steuerelement gebunden ist, kann man diese »löschen«, indem man `<noop>` daran bindet. Im Beispiel schaltet man also die Hilfefunktion für die dritte Karte so aus:

```
<card id="dritte" title="Ok">
  <do type="help" label="Hilfe">
    <noop/>
  </do>
  ...
```

Dabei ist auch das `label` nicht mehr relevant, da Microbrowser dem Nutzer keine so ausgeschalteten Eingabemöglichkeiten anbieten dürfen.

Als letzten Bestandteil von WML lernen Sie im folgenden Kapitel die Verwendung von Variablen kennen.

9 Variablen in WML-Dokumenten

Im WWW liefern die Web-Server Seiten, deren Inhalt statisch ist. Einmal vom Browser eingelesen, kann sich der Text der Seite nicht mehr verändern, wenn man Script-Sprachen und Java außen vor läßt. Bei WML ist dies anders: Die Sprache sieht vor, daß es im Browser einen Speicherbereich – den *Kontext* – gibt, der Variablen enthält. Diese Variablen können durch Tags in einer Seite gesetzt und verändert werden und ihr Inhalt läßt sich in den Text einfügen. Die Auswertung der Variablen geschieht dabei im Browser, nachdem eine Seite übertragen wurde. Dieses Kapitel beschreibt, wie Sie Variablen benennen, benutzen und verändern.

Sie haben schon in den Kapiteln vorher Möglichkeiten zur Verwendung von Variablen in WML-Seiten kennengelernt:

- Im Kontext einer Seite sind Variablen enthalten, die beim Wert `true` im Attribut `newcontext` bei `<card>` neu initialisiert werden – siehe die Beschreibung in 3.3 auf Seite 20.
- Die Auswahl bei `<select>` wird an eine Variable gebunden – siehe Abschnitt 7.1 auf Seite 53.
- Die Eingabe bei `<input>` speichert der Microbrowser in einer Variablen – siehe Abschnitt 7.3 auf Seite 60.
- Der Zählerstand einer Stoppuhr mit `<timer>` kann an eine Variable gebunden werden – siehe Abschnitt 8.7 auf Seite 78.

In diesem Kapitel lernen Sie das Tag `<setvar>`, mit dem man Variablen mit einem Wert als Nebeneffekt einer Aktion belegt. Für die Veränderung von Variablen mit WMLScript steht Ihnen zusätzlich die Funktion `setVar` zur Verfügung, die Sie in Abschnitt 15.5 auf Seite 162 kennenlernen.

9.1 Variablennamen

Variablen erhalten einen Namen, der in WML mit einem Unterstrich (»_«) oder einem Buchstaben zwischen `a` und `z` in Groß- oder

Kleinschreibung beginnen muß, auf die beliebig viele Buchstaben, Unterstriche und Ziffern von `0` bis `9` folgen können. Die deutschsprachigen Umlaute oder andere Sonderzeichen einschließlich dem Leerzeichen sind also nicht erlaubt. Zulässige Namen sind beispielsweise:

```
Variable
_titel
Zeile100
```

Nicht erlaubt sind hingegen:

`Höhe`	Umlaut verboten
`Name Nachname`	Leerzeichen verboten
`%i`	Sonderzeichen verboten
`2000er`	Ziffer am Anfang verboten
`tag.nacht`	Punkt verboten

✘ Im Gegensatz zu WMLScript ist in WML der Unterstrich _ in Variablennamen erlaubt.

9.2 Variablen setzen

WML-Variablen enthalten immer eine Zeichenkette als Wert. Wie in der Einleitung geschrieben, können sie auf verschiedene Weise gesetzt werden, dabei soll uns die WMLScript-Benutzung an dieser Stelle noch nicht kümmern.

Eingaben und Auswahlen in Formularen können an eine Variable gebunden werden und sie damit verändern. Das Beispiel in Abbildung 9.1 auf der nächsten Seite zeigt dies für `<input>`.

Im Attribut `name` ist der Variablenname `blz` angegeben. Die Nutzereingabe wird im Kontext unter diesem Namen gespeichert. Zeigt man die Karte mit gleichem Kontext erneut an, ersetzt der Microbrowser das Auftreten von `$blz` mit diesem Inhalt. In Abschnitt 9.3 auf Seite 89 lernen Sie die Details der Variablenverwendung kennen.

Neben dieser Ihnen bekannten Methode gibt es ein spezielles Tag, das Variablen als Nebeneffekt der Ereignisbehandlung verändern kann: `<setvar>`.

<setvar>

Es kann innerhalb von `<go>`, `<prev>` und `<refresh>` auftreten und bewirkt, daß zusätzlich zu der Aktion Variablen verändert

```
<card id="eingabe" title="">
  <p>
    Bitte BLZ eingeben:<br/>
    <input name="blz" type="text"/><br/>
    Eingabe: $blz
  </p>
</card>
```

Abbildung 9.1
Variable durch eine Eingabe setzen

werden. Dabei können beliebig viele `<setvar>` innerhalb der Aktionsmarkierung stehen.

`<setvar>` kennt neben den Standardattributen `id` und `class` zwei vorgeschriebene Attribute:

- `name` enthält den Namen der zu setzenden Variablen.
- `value` ist der Wert, auf den die Variable gesetzt werden soll.

Das Beispiel in Abbildung 9.3 auf der nächsten Seite soll so den Inhalt von `blz` auf einen Standardwert setzen, wenn der Nutzer den Link »Auto« wählt. Dazu ist ein Anker eingerichtet, dessen Aktion einerseits die aktuelle Seite erneut darstellt, dabei aber nebenher den Inhalt von `blz` mit `<setvar>` verändert.

```
<card id="eingabe" title="">
  <p>
    Bitte BLZ eingeben:<br/>
    <input name="blz" type="text"/><br/>
    Eingabe: $blz<br/>
    <anchor>
      <refresh>
        <setvar name="blz"
         value="123 456 78"/>
      </refresh>Auto</anchor>
  </p>
</card>
```

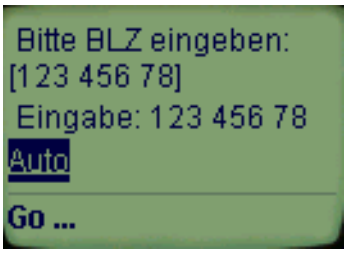
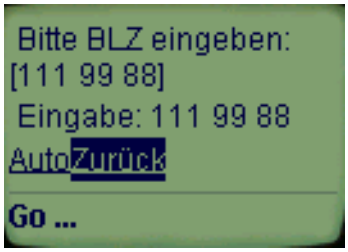


Abbildung 9.2
Setzen einer Variable

Wird die Karte dann neu dargestellt, ist der Standardwert eingetragen und kann wiederum editiert werden.

Als weiterer Komfort soll auf der kleinen Seite mit der Zurück-Taste der zuletzt eingegebene Wert für die Bankleitzahl wieder als Wert gelten.

In dem WML-Code in Abbildung 9.3 wird dazu das Setzen des Standardwerts so erweitert, daß davor der aktuelle Inhalt von `blz` in die Variable `oldblz` kopiert wird. Erst dann erfolgt das Speichern des Standardwerts.




```
<card id="eingabe" title="">
  <p>
    Bitte BLZ eingeben:<br/>
    <input name="blz" type="text"/><br/>
    Eingabe: $blz<br/>
    <anchor>
      <refresh>
      <setvar name="oldblz" value="$blz"/>
      <setvar name="blz"
       value="123 456 78"/>
    </refresh>Auto</anchor>
    <anchor>
      <refresh>
      <setvar name="blz" value="$oldblz"/>
    </refresh>Zurück</anchor>
  </p>
</card>
```

Abbildung 9.3
Speichern eines alten Variableninhalts

Mit einem zweiten Anker kann der Nutzer die Veränderung rückgängig machen. Wieder wird eine `<refresh>`-Aktion mit dem Tag `<setvar>` verbunden. Dort wird der vorher in `oldblz` vermerkte Wert wieder nach `blz` zurückgeschrieben.

✘ WAP-Geräte haben nur begrenzt Speicher. Falls in einem Kontext mehr Variablen gebraucht werden, als in den momentan freien Speicher passen, kann der Microbrowser die Einträge in der Nutzungsgeschichte kürzen und den Zwischenspeicher für Seiten löschen. Falls durch diese Maßnahmen immer noch nicht genügend Platz für alle Variablen ist, informiert der Microbrowser den Nutzer über dieses Problem und wird zu einer problemfreien Karte zurückgehen.

9.3 Variablen verwenden

Den Wert einer Variablen können Sie einfach im laufenden Text einsetzen, indem der Variablenname mit einem vorangestelltem Dollarzeichen `$` notiert wird. Nehmen wir an, daß die Variable `blz` den Wert »111 999 88« enthalte. Durch die Verwendung von `$blz` im WML-Text wird der Inhalt in die laufende Seite eingefügt, wie Sie schon in Abbildung 9.1 auf Seite 87 gesehen haben.

Variablen können an drei Stellen auf Karten verwendet werden:

1. Im fortlaufenden Text,
2. in Attributen, die eine Zeichenkette als Wert erwarten, beispielsweise `title` von `<card>`, und
3. in Attributen, die eine URL als Wert erwarten, beipielsweise `href` von `<go>`.

✘ Beachten Sie, daß das Dollarzeichen `$` reserviert ist, um eine Variablenreferenz einzuleiten. `Ein Euro ist weniger als 1$ wert` ist also zumindest syntaktisch falsch, weil ein Variablenname nicht mit einem Leerzeichen beginnen darf. Korrekt muß es `Ein Euro ist weniger als 1$$ wert` heißen.

Wenn das Ende des Variablennamens nicht einfach erkennbar ist – weil auf ihn kein Zeichen folgt, das nicht zum Namen gehören kann –, müssen Sie ihn mit Klammern einfassen. Während bei `$DM Mark` klar ist, daß die Variable `DM` bezeichnet wird, wäre in `$DM00 Pfennige` nach den WML-Regeln die Variable `DM00` gemeint. `DM` wird erst nach Klammerung angesprochen: `$(DM)00 Pfennige`.

Der Verweis auf eine bislang unbekannte Variable – wie beim obigen Fehler `DM00` – ergibt die leere Zeichenkette als Wert.

Für die Verwendung von Variablen in URLs ist es notwendig, darin enthaltene Sonderzeichen zu codieren, damit die Syntax von URLs eingehalten wird. Neben Sonderzeichen müssen beispielsweise auch Leerzeichen entsprechend übersetzt werden. Die Regeln für eine solche Codierung ist im Internet Standard RFC 2396 ([4]) festgelegt.

Für die Konvertierung stehen Ihnen besondere Darstellungsformen von Variablen neben der normalen bereit. Sie werden jeweils durch ein Schlüsselwort bezeichnet und durch einen Doppelpunkt getrennt an den Variablennamen angehängt, beispielsweise

als `$(blz:escape)`. Die drei in WML definierten Schlüsselworte sind:

escape

1. `escape` bewirkt die Codierung nach URL-Form. So werden beispielsweise die Leerzeichen codiert, so daß eine zusammenhängende Zeichenkette entsteht: Aus `111 999 88` wird `111%20999%2088`. Auch Umlaute und Sonderzeichen stellt die Konvertierung um: Aus `Zuzügl. P&P` wird `Zuz%FCgl.%20P%26P`.

unesc

2. `unesc` decodiert einen in URL-Form abgelegten Variableninhalt nach »normaler« Schreibweise, macht also die Codierungen bei `escape` rückgängig.

noesc

3. `noesc` läßt den Variableninhalt unverändert und ist damit der Standardfall, der nicht besonders ausgezeichnet werden muß. `$(blz:noesc)` ist damit gleichbedeutend mit `$(blz)` und `$blz`.

Damit haben Sie alle Bestandteile von WML kennengelernt. Sie wissen um die Struktur von WML-Stapeln, die Möglichkeiten zur Textauszeichnung, Hyperlinks und Grafiken, komplexere und interaktive Elemente wie Tabellen und Formulare und den Umgang mit Ereignissen, Aktionen und Variablen. Im folgenden Kapitel, das den WML-Teil abschließt, erhalten Sie einen Überblick zu Entwicklungsumgebungen für Ihre WML-Informationssysteme.

10 WML erstellen

WML können Sie mit einem einfachen Editor für unformatierte Texte erstellen. Unter Windows wäre das beispielsweise *Notepad*, aber nicht *Word*. Sie werden dann allerdings viel Mühe mit strukturellen oder Tippfehlern in Ihren Seiten haben. Besser ist es, mit einer Entwicklungsumgebung für WML zu arbeiten und sich auf verschiedene Emulatoren zum Test zu stützen. In diesem Kapitel lernen Sie einige solcher Tools kennen.

10.1 Entwicklungsumgebung

Eine gute Entwicklungsumgebung für WML sollte verschiedene Aufgaben übernehmen:

- Ein Editor soll Sie bei der Erstellung von syntaktisch korrektem WML unterstützen. In der Abbildung 10.1 auf der nächsten Seite sehen Sie das *Nokia WAP Toolkit*, dessen Editor schon bei der Markup-Eingabe die Verwendung korrekter Tags und Attribute überprüft. Sind sie korrekt getippt, werden sie farblich hervorgehoben.
- Notwendige Tools wie ein WBMP-Editor sollten Ihnen integriert zur Verfügung stehen. In Abbildung 6.7 auf Seite 49 hatten Sie schon den WBMP-Editor aus dem Nokia WAP Toolkit gesehen.
- Sie werden selten nur einen Kartenstapel erstellen, sondern in der Regel mehrere Stapel und Grafiken zu einem kleinen Informationssystem zusammenfügen. Eine Entwicklungsumgebung sollte Ihnen hier helfen, diese Dateien als Projekte zusammenzufassen. Die *WapIDE* von Ericsson leistet das, wie Sie in Abbildung 10.2 auf Seite 93 sehen.
- Die Umgebung sollte eine integrierte Emulation eines WML-Browsers zum schnellen Testen der Seiten haben. In den

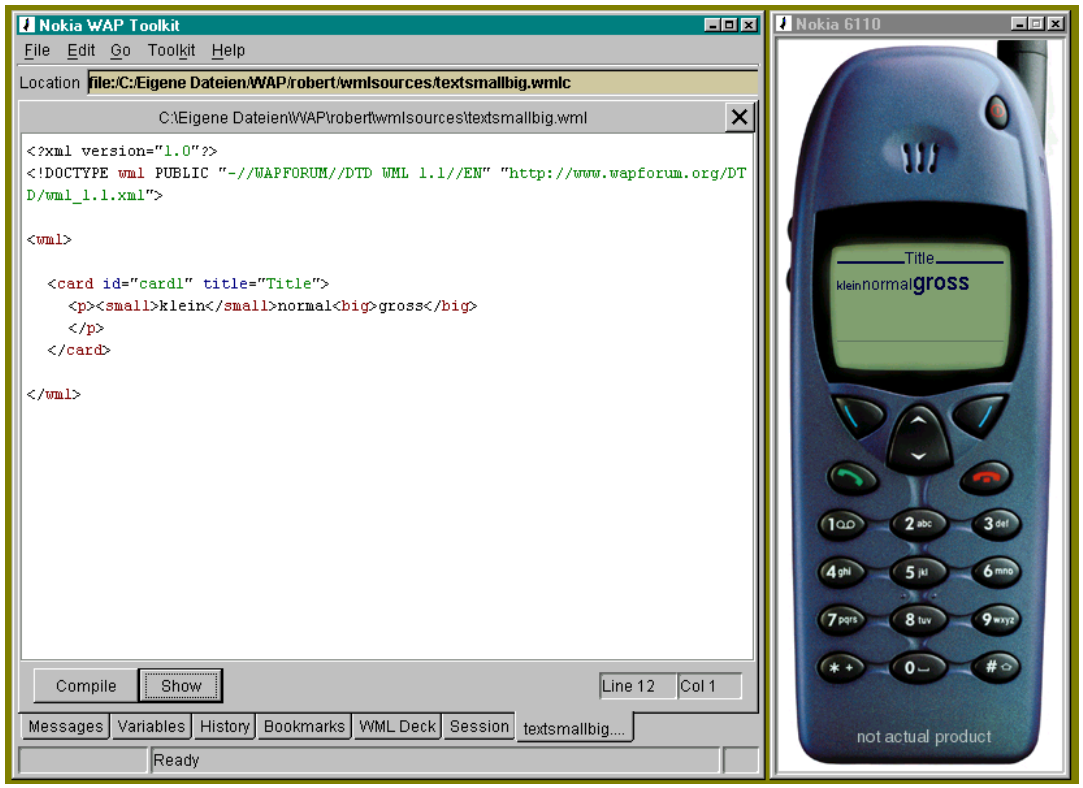


Abbildung 10.1
Das Entwicklungswerkzeug von Nokia

Tools von Nokia und Ericsson sind dies Emulationen konkreter WAP-Handys dieser Hersteller.

- Die Umgebung sollte Sie bei der Entwicklung von WMLScript gleichermaßen unterstützen. Dazu gehört wiederum ein syntaxgesteuerter Editor und ein Emulator für WMLScript. Üblicherweise haben Sie hier für Testzwecke auch einfachen Zugriff auf die jeweils aktuelle Variablenbelegung.

Die aktuell erhältlichen Tools sind schon ausgesprochen mächtig. Die Tools von Nokia und Ericsson sind kostenfrei erhältlich und erfüllen die genannten Anforderungen.

Es wird aber in den kommenden Jahren eine ähnliche Entwicklung wie bei HTML geben. Hier folgten auf erste sehr einfache Tools komfortablere WYSIWYG-Editoren. Die nächste Entwicklungsstufe brachte Site-Editoren hervor, mit denen komplette Web-Sites mit korrekter Verzeigerung editiert werden konn-

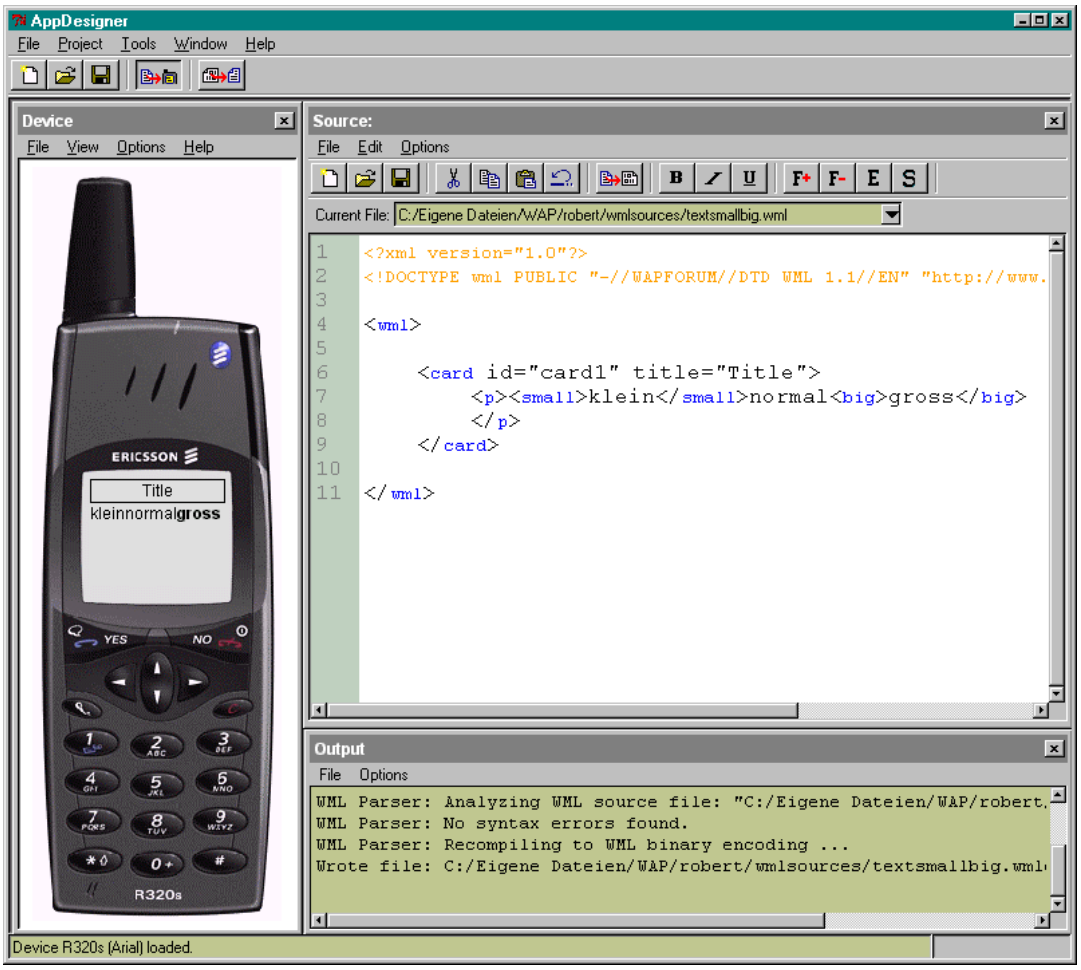


Abbildung 10.2
Das Entwicklungswerkzeug von Ericsson

ten. Mittlerweile sind spezialisierte Tools erhältlich, die auch Web-Feinheiten wie CSS und pixelgenaue Positionierung von Seitenelementen beherrschen.

Ähnlich wird auch bei WML und WMLScript die Entwicklung verlaufen – eventuell in Richtung integrierter Umgebungen, mit denen Web- und WAP-Angebote parallel erstellt werden können.

10.2 Browser zum Testen

Während bei HTML zwar auch umfangreiche Tests mit verschiedenen Browsern notwendig sind, handelt es sich dabei aber nur um eine begrenzte Anzahl wichtiger Programme, die in ähnlichen Systemumgebungen arbeiten.

Bei WML ist die Situation komplexer, da es erheblich mehr verschiedenartige Browser gibt, die in teilweise sehr unterschiedlichen Umgebungen mit oft sehr weit auseinanderliegenden Eigenschaften arbeiten. Typische Beispiele dafür sind unterschiedliche Display-Größe, andere Seitenverhältnisse von Pixeln, verschiedene Eingabemöglichkeiten und -tasten etc.

Abbildung 10.3 *Der phone.com Emulator*

✘ Sie sollten für Ihre WAP-Angebote immer umfangreiche Tests auf sehr unterschiedlichen Geräten durchführen.

In der WML-Spezifikation werden einige der definierten Tags als optional markiert – selbst wenn ein Microbrowser sie nicht implementiert, gilt er immer noch als konform zu WML 1.2. Diese 11 optionalen Tags sind:

✘

```
<b>, <i>, <u>, <em>, <strong>, <small>, <big>, <pre>,
<fieldset>, <optgroup>, <meta>
```

Glücklicherweise gibt es mittlerweile sehr viele Emulatoren für Geräte, so daß sich Tests bequem an einem herkömmlichen PC durchführen lassen. Die Emulatoren unterstützen dabei fast alle die Windows-Plattform, für Unix- und Macintosh-Umgebungen ist die Auswahl sehr dünn. Abbildung 10.3 auf der vorherigen Seite zeigt Ihnen den Emulator der Firma phone.com, dessen technische Basis *UP.Browser* in sehr vielen Handys verwendet wird.

Neben Handys werden zunehmend auch PDAs als WAP-Geräte verwendet. Sie lassen sich oft mit einem Handy verbinden, über das dann lediglich die Datenkommunikation abgewickelt wird. In Abbildung 10.4 sehen Sie den *KBrowser* für Palm-Geräte.

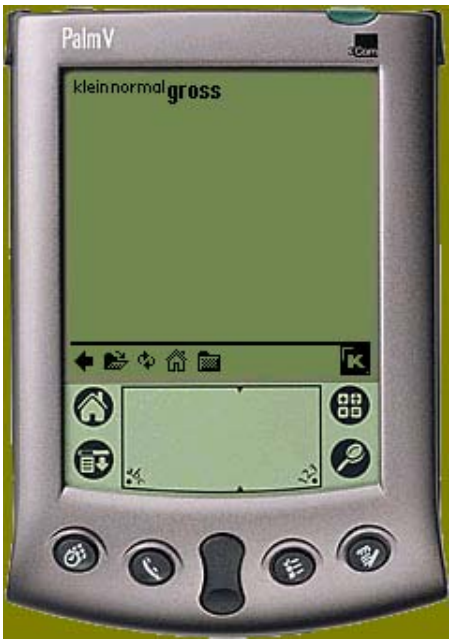


Abbildung 10.4
Der KBrowser für Palm-Geräte

Sie benötigen dazu nicht zwangsläufig einen PDA, da auch diese mittlerweile als genaue Emulation zur Verfügung stehen. Für

Palm-Geräte finden Sie den offiziellen Emulator unter `www.palm.com` im Entwicklerbereich.

Völlig ohne Bezug zu einem konkreten WAP-Gerät finden sich auch erste WML-Browser als normale Programme, wie der *winwap* in Abbildung 10.5.

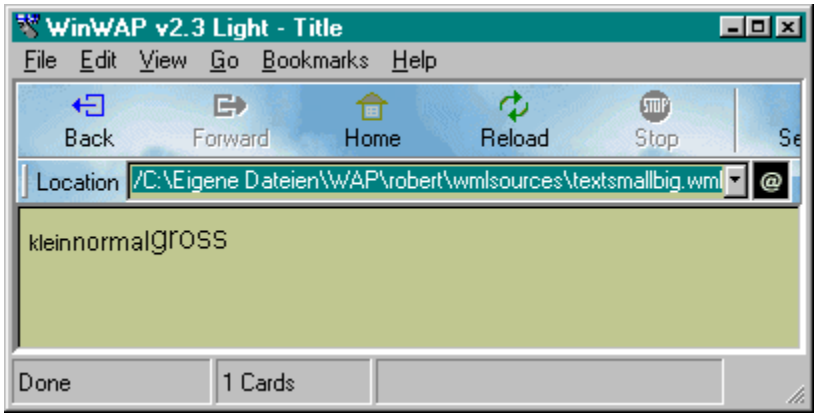


Abbildung 10.5 *Der Windows-basierte winwap-Browser*

Der Web-Browser *Opera* integriert ab der Version 4.01 sogar Web- und WAP-Browser, indem er neben HTML auch WML unterstützt, wie in Abbildung 10.6 zu sehen.

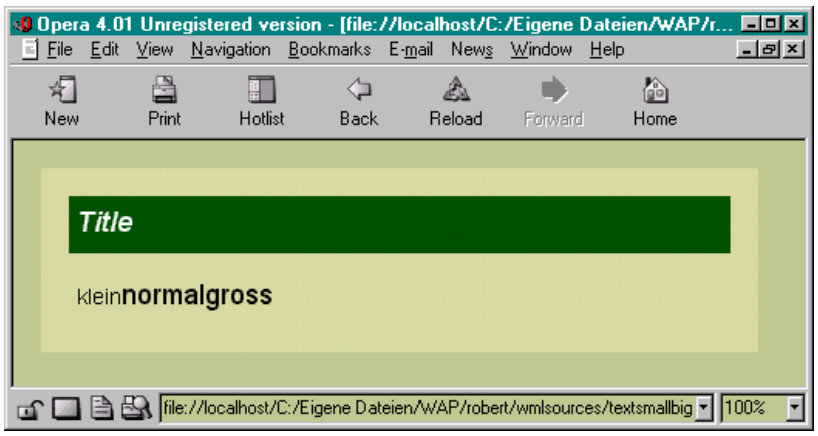


Abbildung 10.6 *Im Opera Web-Browser ist WML-Unterstützung integriert*

Schließlich gibt es auch erste Web-Dienste, die das Browsen von WML-Seiten durch Konvertierung nach HTML ermöglichen. In Abbildung 10.7 auf der nächsten Seite sehen Sie den *wapalizer*-Dienst, der WML-Seiten in einem HTML-Fenster darstellt.

Abbildung 10.7
wapalizer: WML als HTML-Seite

Sehr viele und leistungsfähige Tools sind frei erhältlich, so daß Ihnen praktisch keine Kosten für Entwicklungstools entstehen. In der Tabelle in Abbildung 10.8 auf der nächsten Seite finden Sie einige Netzadressen.

Mit diesem Blick auf Entwicklungsumgebungen sind Sie in der Lage, WML-Informationssysteme zu schreiben. Im zweiten, folgenden Teil dieses Buchs lernen Sie die Script-Sprache für WAP kennen: WMLScript.

Browser	URL
Nokia WAP Toolkit 2.0	www.forum.nokia.com/main.html
Ericsson R380 Emulator	www.ericsson.se/developerszone
Phone.com UP.SDK Emulator	developer.phone.com
Opera 4.0 Browser / Windows	www.opera.no
WinWAP Browser / Windows	www.slobtrot.com/winwap/index.htm
WAPman Browser / Windows	www.wap.com.sg/downloads/downloads.htm
EzWAP Browser / Windows	www.ezos.com/Soft/EzWAP/EzWap.asp
AU Systems Browser / Palm	www.wapguide.com/wapguide/browser.html
WAPman Browser / Palm	www.wap.com.sg/downloads/downloads.htm
WAPJAG Konverterdienst	www.wapjag.de
Wapalizer Konverterdienst	www.gelon.net
Wappy's Wapview	wappy.to

Abbildung 10.8
WAP Browser und Emulatoren

Teil II

WMLScript

11 Daten, Datentypen und Ausdrücke

Eine Programmiersprache wie WMLScript besteht aus Definitionen der Syntax, Regeln über verwendbare Daten, ihre Typen und Operatoren darauf sowie den Konstrukturen zur Ablaufsteuerung zusammen. In diesem Kapitel lernen Sie die in WMLScript erlaubten Datentypen und die Operatoren darauf kennen.

11.1 Datentypen

In WMLScript sind insgesamt fünf Datentypen bekannt. Sie sind jeweils definiert durch Konstante, einen Wertebereich und Operatoren, mit denen Ausdrücke gebildet werden können.

Während in WMLScript zwar Daten getypt sind, sind Variablen ungetypt – eine Variable in WMLScript kann also Werte aus allen Datentypen aufnehmen. Einer Zuweisung `a=10` kann also durchaus später eine Zuweisung `a="WML"` folgen. Dementsprechend gibt es auch nicht wie in anderen Programmiersprachen Schlüsselworte wie `int` oder `INTEGER`, die einen Typen bezeichnen.

Die Datentypen in WMLScript sind Wahrheitswerte, Ganzzahlen, Fließkommazahlen, Zeichenketten und ein besonderer Typ – der ungültige Wert.

11.1.1 Wahrheitswerte

Wahrheitswerte in WMLScript umfassen die Werte falsch und wahr. Es gibt zwei Konstanten, `false` für falsch und `true` für wahr. Dabei gilt `true` als Maximal- und `false` als Minimalwert, wodurch beispielsweise auch `false<true` gilt.

false

true

11.1.2 Ganzzahlen

Ganzzahlen sind positive und negative ganze Zahlen in WMLScript, die man auch öfter mit dem englischen Begriff »Integer«

bezeichnet. Der Wertebereich ist genau definiert, da er sich aus der definierten Speichergröße für eine Ganzzahl errechnet. Es werden genau 32 Bit einschließlich einem Vorzeichen in üblicher Zweierkomplement-Darstellung verwendet. Daraus ergibt sich ein Wertebereich von $-2\,147\,483\,648$ bis $2\,147\,483\,647$. Die Bibliotheksfunktionen `Lang.minInt()` und `Lang.maxInt()` liefern diese Minimal- und Maximalwerte auch in einem Programm zurück.

Integer-Konstanten werden als eine Folge von Ziffern zwischen `0` und `9` notiert, wobei allerdings keine führende `0` erlaubt ist. Negative Konstanten gibt es nach den Regeln der Syntax eigentlich nicht, sondern entstehen durch Anwendung des Operators `-` auf eine positive Ganzzahlkonstante. Praktisch macht diese Feinheit für Sie keinen Unterschied.

Während die normalen Ganzzahlen zur üblichen Dezimalbasis notiert werden, kennt WMLScript zusätzlich noch die hexadezimale und oktale Schreibweise:

- *Oktale* Zahlen werden zur Basis 8 interpretiert – verwendbare Ziffern reichen also von `0` bis `7`. In WMLScript stellen Sie einer solchen Konstante das Zeichen `0` (die Null) voran. Damit hat also die Oktalkonstante `0273` den dezimalen Wert $2*8^2+7*8^1+3*8^0=2*64+7*8+3*1=187$.

  0

  ✘ Wegen der Markierung mit `0` dürfen Dezimalzahlen keine führenden Nullen haben. Gleichzeitig sind beispielsweise `08` und `09` in WMLScript nicht erlaubt, da sie mit Oktalzahlen beginnen und dann aber ungültige Ziffern verwenden.

  ✘ Oktale Schreibweise ist mittlerweile wirklich ein Relikt aus der frühen Computerzeit. Sie werden sie kaum in Ihren Programmen verwenden.

- *Hexadezimalzahlen* sind zur Basis 16 gerechnet. Damit läßt sich der Wert eines Bytes immer mit zwei Ziffern ausdrücken. Verwendbare Ziffern sind `0` bis `9`, die dann von den Buchstaben `A` bis `F` und deren Kleinschreibungen gefolgt werden. In WMLScript werden hexadezimale Konstanten mit den Zeichen `0x` oder `0X` begonnen.

  0x

  0X

  Die Zahlen `0xF`, `0xf` `0Xf` und `0XF` sind jeweils hexadezimale Konstanten mit dem dezimalen Wert 15. `0xD00F` entspricht $13*16^3+0*16^2+0*16^1+16*16^0=13*4096+0*256+0*16+15*1=53263$.

11.1.3 Fließkommazahlen

Fließkommazahlen sind in WMLScript als 32-Bit Werte nach dem Standard IEEE754 definiert. Damit ist der größte mögliche Wert $3,40282347 * 10^{38}$ und der kleinste darstellbare Wert $1,17549435 * 10^{-38}$. Die beiden Bibliotheksfunktionen `Float.maxFloat()` und `Float.minFloat()` liefern diese Werte auch in einem Programm zurück.

Die Definition des Zahlenformats sieht auch ungültige Zahlen – als $0/0$ definiert – sowie positive und negative Unendlichkeit ($1/0$ und $-1/0$) vor. Falls in einer Berechnung ein solcher Wert auftritt, wird der Ausdruck mit `invalid` (siehe Abschnitt 11.1.5 auf der nächsten Seite) bewertet.

Fließkommakonstante setzen sich aus einem Vorkomma-, einem Nachkomma- und einem Exponententeil zusammen. Vor- und Nachkommateil sind durch den Dezimalpunkt `.` getrennt. Es kann sich mit einem `e` oder `E` getrennt ein Exponentialteil anschließen.

`.`
`e`
`E`

Beispiele für gültige Fließkommazahlen sind:

```
1.
1.0
.01
-1.0E21
-10.0e-10
```

Nicht erlaubt sind beispielsweise `1e8` – der Nachkommateil fehlt – oder `1.2e40` – der Exponent ist zu groß.

11.1.4 Zeichenketten

Zeichenketten in WMLScript sind Listen aus den unterstützten Zeichen. Da WMLScript eine XML-basierte Sprache ist, können alle Zeichen nach der UNICODE-Codierung verwendet werden, die Sie schon in Abschnitt 4.1 auf Seite 23 kennengelernt haben.

Konstante Zeichenketten werden in WMLScript entweder von einem Paar doppelter (") oder einfacher Anführungszeichen (') umschlossen. `"Robert"` und `'Robert'` sind also gleich.

`"`
`'`

Um nun aber innerhalb einer Zeichenkette auch die Anführungszeichen verwenden zu können, sind eine Reihe von Kürzeln – die »Escape Sequences« definiert. Sie finden sie in der Tabelle in Abbildung 11.1 auf der nächsten Seite aufgelistet. Um also »`\"` und `\'`« zu notieren, müßten Sie mit den Escape-Sequenzen arbeiten und `"\\\"` und `\\'"` oder `'\\"` und `\\\''` schreiben.

Abbildung 11.1 *Die Escape-Sequenzen*

Operator	Bedeutung
`\'`	Apostroph: '
`\"`	Anführungszeichen: "
`\\`	Rückwärtsstrich: \
`\/`	Vorwärtsstrich: /
`\b`	Steuerzeichen Backspace
`\f`	Steuerzeichen Seitenvorschub (»form feed«)
`\n`	Steuerzeichen neue Zeile (»newline«)
`\r`	Steuerzeichen Wagenrücklauf (»carriage return«)
`\t`	Steuerzeichen Tabulator
`\x`*hh*	ISO 8859-1 Zeichen hexadezimal
`\o`*ooo*	ISO 8859-1 Zeichen oktal
`\u`*hhhh*	UNICODE Zeichen hexadezimal

Neben den vier ausdrückbaren druckbaren Zeichen finden Sie fünf erzeugbare Steuerzeichen in den Escape-Sequenzen, die textorientierte Terminals interpretieren.

Für die Notation von Zeichen durch ihren Zeichencode können Sie hexadezimale und oktale Schreibweise für Zeichen aus dem Code ISO 8859-1 verwenden, oder ein UNICODE-Zeichen direkt per hexadezimalem Code auswählen.

11.1.5 Ungültige Werte

invalid

WMLScript kennt einen besonderen Typen mit nur einem konstanten Wert: `invalid`. Er bezeichnet den ungültigen Wert und Typen: Ein Wert, der das Ergebnis einer ungültigen Berechnung ist. WMLScript nutzt diesen Mechanismus, um beispielsweise eine Division durch 0 durchzuführen – das Ergebnis ist `invalid`.

11.2 Typisierung

Typisierung bedeutet Entscheidungen darüber, von welchem Typ ein Wert ist und ob er zu einem anderen Typen verträglich ist.

In WMLScript haben Variablen selber keinen festen Typen – er ergibt sich aus dem jeweiligen Inhalt.

Zur Laufzeit ist in WMLScript der Typ eines Ausdrucks ermittelbar. Die Konstanten liefern die Ausgangsbasis und sind unterscheidbar. Die damit gebildeten Ausdrücke verwenden Operatoren, die Sie in Abschnitt 11.3 auf der nächsten Seite kennenlernen werden. Bei ihnen allen ist die Typisierung bekannt, so ergibt die Addition zweier Fließkommazahlen einen neuen Fließkommawert. Auf diese Weise ist zu jedem Zeitpunkt feststellbar, von welchem Typen ein bestimmter Ausdruck ist.

typeof

Sie können zur Laufzeit diese Typisierung mit Hilfe des Operators `typeof` ermitteln. Er wird auf einen Ausdruck angewandt und liefert eine Codezahl, die für einen der fünf Typen steht. Sie finden diese Liste in Abbildung 11.2.

Codezahl	Typ
0	Ganzzahl
1	Fließkommazahl
2	Zeichenkette
3	Wahrheitswert
4	ungültig

Abbildung 11.2
Die arithmetischen Grundoperatoren

isvalid

Zusätzlich können Sie mit dem Operator `isvalid` überprüfen, ob ein Ausdruck als `invalid` bewertet wird. Das Ergebnis des Tests ist ein Wahrheitswert.

Operatoren zur Bildung von Ausdrücken sind nicht für alle Kombinationen von Datentypen definiert. So kann man beispielsweise einen Wahrheitswert nicht mit einer Fließkommazahl vergleichen. WMLScript versucht immer, durch Typanpassung dafür zu sorgen, daß eine korrekte Typisierung vorgefunden wird. Dabei gelten folgende Regeln:

- *Wahrheitswerte* ergeben sich zu `false` aus der Ganzzahl 0, der Fließkommazahl 0.0 und der leeren Zeichenkette. Alle anderen Zahlen und Zeichenketten werden zu `true` angepaßt. `invalid` läßt sich nicht zu einem Wahrheitswert konvertieren. Dazu einige Beispiele, zu denen Sie nach der Beschreibung der Operatoren für Ausdrücke in Abschnitt 11.3 auf der nächsten Seite nochmals zurückkehren sollten:

  - `true && 3.4` ergibt `true`
  - `"A"||"B"` ergibt `true`
  - `!42` ergibt `false`

- *Ganzzahlen* ergeben sich aus dem Wahrheitswert `false` zu 0 und aus `true` zu 1. Zeichenketten können dann zu ganzen Zahlen gewandelt werden, wenn sie den syntaktischen Regeln für Ganzzahlkonstante folgen. Weder aus Fließkommazahlen noch aus `invalid` lassen sich ganze Zahlen erzeugen. Beispiele sind:

  - `"7"<<3` ergibt `56`
  - `true<<3` ergibt `8`
  - `+true` ergibt `1`

- *Fließkommazahlen* ergeben sich aus den Wahrheitswerten `false` und `true` zu 0.0 bzw. 1.0 und aus allen Ganzzahlen zur entsprechenden Fließkommazahl mit leerem Nachkommateil. Zeichenketten müssen den syntaktischen Regeln für Fließkommazahlen folgen um konvertierbar zu sein. `invalid` ist nicht zu einem Fließkommawert wandelbar. Dazu wieder Beispiele:

  - `7/2.5` ergibt `2.8`
  - `"2.5"-"7"` ergibt `-4.5`
  - `9.9+true` ergibt `10.9`

- *Zeichenketten* sind aus Wahrheitswerten als »`false`« und »`true`« erzeugbar und aus Ganz- und Fließkommazahlen als entsprechende Zahlenrepräsentation. `invalid` kann nicht in eine Zeichenkette umgewandelt werden.

- *`invalid`* kann nie Ergebnis einer Konvertierung sein.

11.3 Ausdrücke

Durch Ausdrücke entstehen Werte, die in Anweisungen verwendet werden können. Ausdrücke in WMLScript sind:

1. die Konstanten, deren Wert sich aus ihnen ergibt,

2. Variablen, deren Wert der Inhalt der Variablen ist,
3. Funktionsaufrufe, deren Wert das Ergebnis der Funktion ist,
4. in (und) geklammerte Ausdrücke, deren Wert sich aus ihrer Berechnung ergibt, ( )
5. durch Operatoren verknüpfte Ausdrücke, deren Art und Bedeutung Sie in den folgenden Abschnitten kennenlernen.

WMLScript orientiert sich damit an den üblichen Konventionen zur Ausdruckbildung in imperativen Programmiersprachen.

Eine besondere Art von Ausdrücken sind die Konditionalausdrücke. Sie haben die folgende Form:

Bedingung?*true-Wert*:*false-Wert*. ?

Bei der Auswertung wird zunächst die *Bedingung* getestet. Ergibt sie `true`, dann ist der Wert des gesamten Ausdrucks der Wert von *true-Wert*. Lieferte sie `false` oder `invalid`, dann berechnet sich der gesamte Ausdruck zu *false-Wert*.

Um beispielsweise das Doppelte einer Variablen `a` zu liefern, falls sie positiv ist, und 0 falls nicht, könnten Sie folgenden Ausdruck schreiben:

```
(a>=0) ? 2*a : 0
```

11.3.1 Arithmetische Operatoren

Die arithmetischen Operatoren in WMLScript umfassen zunächst die normalen Grundrechenarten, die Sie in der Tabelle in Abbildung 11.3 auf der nächsten Seite dargestellt sehen. Beachten Sie, daß die Division von Ganzzahlen anders notiert wird als die Fließkommadivision.

Der Summenoperand + hat eine andere Bedeutung, wenn er auf Zeichenketten angewandt wird. Dort bedeutet er die Verbindung zweier Zeichenketten zu einer neuen. `'Heller ' + 'Pfennig'` ergibt also die Zeichenkette `'Heller Pfennig'`. Die anderen arithmetischen Operatoren sind nicht für Zeichenketten definiert. +

Die komplexeren arithmetischen Operatoren aus der Abbildung 11.4 auf der nächsten Seite umfassen die Vorzeichenbildung und die Prä- und Postdekrement und -inkrement Operationen.

Bei einem Präinkrement wird zunächst der Wert einer Variablen ermittelt und um eins erhöht. Der Wert des gesamten

Abbildung 11.3
Die arithmetischen Grundoperatoren

Operator	Bedeutung
a+b	Summe, also $a + b$
a-b	Differenz, also $a - b$
a*b	Produkt, also $a * b$
a/b	Division, also a/b
a%b	ganzzahliger Divisionsrest $a \bmod b$
i `div` j	Division von Ganzzahlen, also i/j

Abbildung 11.4
Die komplexen arithmetischen Operatoren

Operator	Bedeutung
+b	Plus b
-b	Minus b
++b	Präinkrement von b
b++	Postinkrement von b
--b	Prädekrement von b
b--	Postdekrement von b

Ausdrucks ist dann der erhöhte Wert. Bei einem Postinkrement wird zunächst der Wert des Ausdrucks ermittelt und dann die Erhöhung durchgeführt. Wenn die Variable `i` also den Wert 10 enthält, dann ergibt `++i` als Ausdruck die 11, während `i++` eine 10 liefert. In beiden Fällen ist `i` hinterher 11. Die Prä- und Postdekrement-Operatoren arbeiten genauso, substrahieren aber jeweils 1.

In der Abbildung 11.5 auf der nächsten Seite finden Sie die WMLScript-Operatoren, die auf den Bitmustern von Werten arbeiten.

Zur Bildung von Ausdrücken mit Wahrheitswerten stehen Ihnen die üblichen logischen Operatoren aus Abbildung 11.6 auf der nächsten Seite bereit. Genauso liefern die Vergleiche aus Abbildung 11.7 auf Seite 110 Wahrheitswerte.

✘ WMLScript benutzt die *Shortcut Evaluation* bei der Auswertung logischer Ausdrücke. Mit ihr wird die Auswertung so früh wie möglich beendet und damit nicht der gesamte Ausdruck unter-

Operator	Bedeutung
a<<n	Verschiebung des Bitmusters von a nach links um n Bit-Stellen
a>>n	Verschiebung des Bitmusters von a nach rechts um n Bit-Stellen unter Beibehaltung des Vorzeichenbits
a>>>n	Verschiebung des Bitmusters von a nach rechts um n Bit-Stellen unter Auffüllung der neuen Stellen links im Bitmuster mit 0
a&b	UND-Verknüpfung der Bitmuster von a und b
a|b	ODER-Verknüpfung der Bitmuster von a und b
a^b	Exklusiv-ODER-Verknüpfung der Bitmuster von a und b
~b	Negation des Bitmusters von b

Abbildung 11.5
Bit-Operatoren

Operator	Bedeutung
f&&g	Logisches UND ($f \wedge g$)
f||g	Logisches ODER ($f \vee g$)
!g	Negation ($\neg g$)

Abbildung 11.6
Die logischen Operatoren

sucht. Befinden sich Funktionsaufrufe in diesen nicht bewerteten Ausdrücken, ergibt sich ein anderer Programmablauf. In drei Situationen tritt eine solche abgekürzte Auswertung auf:

- Bei `f&&g` kann der gesamte Ausdruck schon dann als `false` bewertet werden, wenn `f` `false` ergibt. WMLScript wertet in diesem Fall `g` nicht mehr aus.
- Bei `f||g` kann der gesamte Ausdruck schon dann als `true` bewertet werden, wenn `f` `true` ergibt. WMLScript wertet in diesem Fall `g` nicht mehr aus.

- Bei `f&&g` und `f||g` kann der gesamte Ausdruck schon dann als `invalid` bewertet werden, wenn `f` `invalid` ergibt. Der WMLScript-Interpreter wertet in diesen Fällen `g` nicht mehr aus.

Abbildung 11.7 *Die Vergleichsoperatoren*

Operator	Bedeutung
$a<b$	Kleiner
$a<=b$	Kleiner gleich
$a==b$	Gleichheit
$a!=b$	Ungleichheit
$a>=b$	Größer gleich
$i>j$	Größer

11.4 Zuweisungen

WMLScript kennt für Zuweisungen an Variablen auch abgekürzte Schreibweisen, wenn die Zuweisung der Art *v=v operator Ausdruck*, also beispielsweise `a=a+b*3`. In diesem Fall wird dem Zuweisungszeichen der Operator vorangestellt und somit abgekürzt *v operator = Ausdruck* geschrieben. Für das Beispiel ergibt sich `a+=b*3`. Die Tabelle in Abbildung 11.8 auf der nächsten Seite zeigt die definierten Zuweisungsoperatoren.

11.5 Bindungsregeln

( )

In Ausdrücken können Sie mit der Klammerung mit (und) die Auswertungsreihenfolge für einen Ausdruck festlegen. Ansonsten gelten die für die Operatoren definierten und in Abbildung 11.9 auf der nächsten Seite dargestellten Bindungsregeln nach der Stufe S geordnet.

Mit den Grundlagen aus diesem Kapitel kennen Sie die Arten von Daten in WMLScript und die darauf definierten Operatoren. Im folgenden Kapitel kommen die Sprachbestandteile hinzu, die den Ablauf eines Programms steuern.

Kurzform	Entspricht
v+=*Ausdruck*	*v*=*v*+*Ausdruck*
v-=*Ausdruck*	*v*=*v*-*Ausdruck*
*v**=*Ausdruck*	*v*=*v***Ausdruck*
v/=*Ausdruck*	*v*=*v*/*Ausdruck*
v div=*Ausdruck*	*v*=*v* div*Ausdruck*
v%=*Ausdruck*	*v*=*v*%*Ausdruck*
v<<*Ausdruck*	*v*=*v*<<*Ausdruck*
v>>=*Ausdruck*	*v*=*v*>>*Ausdruck*
v>>>=*Ausdruck*	*v*=*v*>>>*Ausdruck*
v&=*Ausdruck*	*v*=*v*&*Ausdruck*
v^=*Ausdruck*	*v*=*v*^*Ausdruck*
v|=*Ausdruck*	*v*=*v*|*Ausdruck*

Abbildung 11.8
Die Zuweisungsoperatoren

S	Operatoren
1	++, --, + (unär), - (unär), ~ (unär), ! (unär), typeof, isvalid
2	*, /, div, %
3	-, +
4	<<, >>, >>>
5	<, <=, >, >=
6	==, !=
7	&
8	^
9	|
10	&&
11	||
12	? :
13	=, *=, -=, /=, %=, div=, +=, <<=, >>=, >>>=, &=, ^=, |=
14	,

Abbildung 11.9
Die Bindungskraft der Operatoren

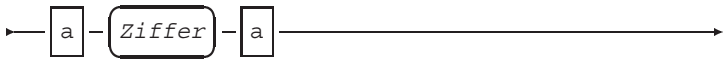
12 Kontrollstrukturen

Imperative – also anweisungsorientierte – Programmiersprachen wie Pascal, Modula-2, C, JavaScript und auch WMLScript beruhen auf einem einfachen Modell der Ausführung von Programmen: Ein Prozessor führt Anweisung nach Anweisung aus. Die Anweisungen manipulieren den Speicher durch Änderungen an Variablen. Diese Zuweisungen sind die einzigen Anweisungen, die das Ergebnis des Programmablaufs bewirken. Ein- und Ausgaben sind dabei lediglich Nebeneffekte.

Hinzu kommen Anweisungen, die den Kontrollfluß steuern, also Einfluß auf die normale sequentielle Ausführung nehmen. So lenken Entscheidungen den Programmfluß in Teile des Programms und Schleifen wiederholen die Ausführung von einzelnen Programmteilen.

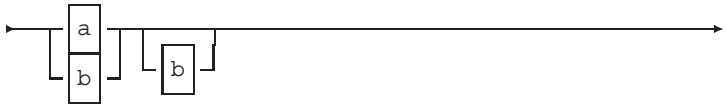
In diesem Kapitel lernen Sie die Teile von WMLScript kennen, die aus Variablen und Ausdrücken Zuweisungen machen und den Kontrollfluß des Programms steuern.

Wir verwenden zur Illustration der Syntax Diagramme. Sie sind wie Ablaufpläne zu lesen und alle möglichen Wege entlang der Pfeile sind gültige WMLScript-Ausdrücke:

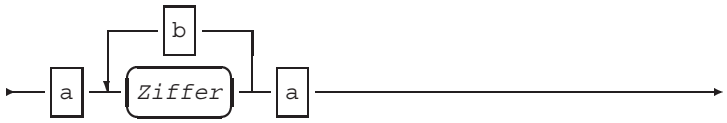


Hiermit ist eine Syntax beschrieben, bei der zunächst ein `a` stehen muß, auf das eine Ziffer folgt und schließlich ein weiteres `a`. Sie sehen, daß Schlüsselworte der Sprache in normaler Schreibmaschinenschrift in einem Kästchen notiert sind und weitere syntaktische Einheiten in kursiver Schreibmaschinenschrift in einem Kasten mit abgerundeten Ecken. Solche Strukturen wie *`Ziffer`* finden Sie im Text oder durch ein weiteres Diagramm erläutert.

In den Diagrammen finden Sie zwei komplexere Strukturen: Alternativen und Wiederholungen. Hier ein Beispiel für ein Diagramm mit Alternativen:



Es verlangt zunächst ein `a` oder `b`, das von einem weiteren `b` oder keinem Zeichen gefolgt wird. Konform zu dieser Syntax sind also `a`, `b`, `ab` und `bb`. Die Wiederholung sieht so aus:



Hiermit muß am Beginn und Ende eines Ausdrucks jeweils ein `a` stehen. Dazwischen können beliebig viele Ziffern auftreten, die jeweils von `b` getrennt sein müssen. Beispiele wären `a1a`, `a1b2a` usw. `aa` ist nicht konform zu der Syntax, da mindestens eine Ziffer auftreten muß.

In WMLSkript gibt es die folgenden Arten von Anweisungen:

- Einfache Anweisungen umfassen die leere Anweisung und Anweisungslisten, die Zuweisung und Blöcke und die Variablendeklaration. Hinzu kommen Kommentare im Programmtext.
- Schleifenanweisungen sind `while`, `for`, `break` und `continue`.

12.1 Anweisungen und Anweisungslisten

Die einfachste Anweisung in WMLScript ist die leere Anweisung, die von `;` abgeschlossen sein muß. Anweisungslisten fassen mehrere Anweisungen zusammen. Die in Programmiersprachen gar nicht so unübliche Definition einer leeren Anweisung hat zwei wichtige Auswirkungen:

1. Da Anweisungslisten aus mehreren Anweisungen bestehen, können Strichpunkte darin beliebig häufig an beliebigen Stellen auftreten. So finden sich in den folgenden beiden Beispielzeilen insgesamt vier leere Anweisungen vermischt mit Zuweisungen, die von `;` abgeschlossen sind und die hier mit ☆ gekennzeichnet sind:

   ```
   ☆; a = 10 ;
   ☆;☆; b = 20 ;☆; c = 30 ;
   ```

2. Eine Anweisungsliste kann auch leer sein – sie enthält dann eine leere Anweisung, die von keiner anderen durch `;` getrennt ist. Damit können beispielsweise Schleifenkörper leer bleiben, wenn es sich um eine reine Zählschleife handelt.

12.2 Zuweisungen

Eine leere Anweisung hat natürlich keinen nutzbaren Effekt. Die erste wirklich sinnvolle Art von Anweisung in WMLScript kennen Sie schon – es ist die Zuweisung, die ebenfalls von `;` abgeschlossen sein muß. Sie hat syntaktisch die folgende Form:

Damit der Compiler eine solche Zuweisung akzeptiert, muß *Variable* an der Stelle der Anweisung gültig, also deklariert sein. Wie Variablen deklariert werden, finden Sie in Abschnitt 12.5 auf Seite 117 beschrieben.

Bei einer Zuweisung finden gleichzeitig automatische Anpassungen statt. Sie passen Ausdrücke so an, daß ihre Typen zum verwendeten Operator passen. Die Details von Typanpassungen haben Sie schon in Abschnitt 11.2 auf Seite 104 kennengelernt.

Während für den Einstieg diese Form der Zuweisung ausreicht, ergeben sich für Fortgeschrittene eine ganze Reihe weiterer Feinheiten, wenn man sich die Definition von WMLScript genau anschaut. Als Neueinsteiger können Sie diese Details überspringen und in Abschnitt 12.3 auf Seite 117 fortfahren.

So ist genaugenommen in WMLScript eine Zuweisung keine Anweisung sondern ein Ausdruck. Die Veränderung des Inhalts einer Variablen ist die Anwendung eines Zuweisungsoperators in einem Ausdruck. Zugleich besteht eine Zuweisungsanweisung aus genau einem Ausdruck. Mit dem Operator `,` (das Komma) kann man mehrere solcher Zuweisungsausdrücke zusammenfassen:

```
a = 10 , b = 20
```

Da der `,`-Operator tatsächlich nur Ausdrücke kombiniert, ist auch `a = 10, 20` ein gültiger Zuweisungsausdruck. Die Syntax erlaubt sogar `10, 20;` als Anweisung. Sie sollten solche Möglichkeiten besser nicht nutzen, da sie einerseits ausgesprochen schlecht les- und wartbar sind und andererseits mit den »normalen« WMLScript-Konstrukten gleichwertig zu notieren sind.

Im Gegensatz zu `a = 10 ; b = 20` handelt es sich beim obigen Beispiel syntaktisch um eine einzige Anweisung und nicht um eine Anweisungsliste. Kontrollstrukturen wie Schleifen erfordern teilweise genau eine Anweisung an bestimmten Stellen und in diesem Fall kommt diese Ausdruckskombination zum Einsatz.

Ein Zuweisungsausdruck selber hat auch einen Wert, so berechnet sich `b = 10` insgesamt zu 10 und hat gleichzeitig den Nebeneffekt, die Variable `b` auf diesen Wert zu setzen. Man kann einen solchen Ausdruck auch auf der rechten Seite einer Zuweisung verwenden:

```
a = b = 10 ;
```

Aufgrund der Vorrangregeln für Operatoren ist diese Anweisung gleich `a = (b = 10) ;`. Da `b = 10` den Wert 10 ergibt, haben hinterher sowohl `a` als auch `b` den Inhalt 10.

Mit einem Ausdruck lassen sich durch Operatoren weitere Ausdrücke bilden, also ist auch die folgende Zuweisung möglich. wobei die Klammerung nur der besseren Lesbarkeit dient:

```
a = (b = 10) + 20;
```

`b` erhält als Nebeneffekt den Inhalt 10, während `a` auf $10 + 20$, also 30 gesetzt wird. Schließlich kann auch die Summe aus zwei Zuweisungsausdrücken gebildet werden:

```
a = (b = 10) + (b = 20);
```

Die Auswertung weist zunächst den Wert 10 der Variablen `b` zu und vermerkt 10 als ersten Summanden. Danach geschieht die Auswertung des zweiten Zuweisungsausdrucks, der als Nebeneffekt den Wert 20 an `b` zuweist und gleichzeitig 20 als zweiten Summanden vermerkt. Durch die Addition entsteht schließlich der Wert 30 als Inhalt von `a` nach der Zuweisung.

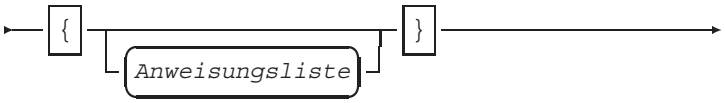
✘ Auch wenn die Definition von WMLScript solche komplexen Mechanismen vorsieht, sollten Sie sie nicht verwenden. Etwas genauere Überlegung zeigt überraschende Folgen dieser Mechanismen auf. Während `a = 10 + 20;` den gleichen Effekt wie `a = 20 + 10;` hat, ist die Anweisung `a = (b = 10) + (b = 20);` nicht gleichbedeutend mit `a = (b = 20) + (b = 10);`, weil `b` hinterher einen anderen Wert hat.

12.3 Blöcke

Während Anweisungslisten tatsächlich als mehrere Anweisungen zählen, umfassen Blöcke eine Anweisungsliste und gelten in ihrer Gesamtheit aber als eine einzige Anweisung. Sie werden von den geschweiften Klammern { und } umschlossen und können auch leer sein:

{

}



Blöcke sind an den Stellen notwendig, an denen die WMLSyntax eine Anweisung fordert, die keinen Wert ergibt. Dieses ist bei den Entscheidungen und Schleifen aus den folgenden Abschnitten der Fall.

12.4 Kommentare

WMLScript kennt zwei Arten von Kommentaren, deren Syntax Ihnen vielleicht von C++ oder Java her bekannt ist:

1. *Blockkommentare* werden von `/*` und `*/` umschlossen und können sich so über mehrere Zeilen erstrecken. Es ist nicht möglich, Blockkommentare zu schachteln – bei

   /*

   */

   ```
   /* Kommentar /* noch einer */ hier */
   ```

   erkennt der Compiler das Ende des Kommentars vor `hier */` und wird einen Syntaxfehler monieren.

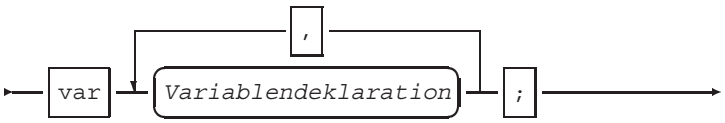
2. *Zeilenkommentare* beginnen mit `//` und enden am Ende der Zeile im Quelltext.

   //

12.5 Variablendeklarationen

Variablen müssen vor ihrer Verwendung im Programmtext deklariert sein. Um dies zu erreichen, müssen Sie die Deklarationsanweisung verwenden. Die wird vom Schlüsselwort `var` eingeleitet und enthält eine oder mehrere von `,` getrennte Deklarationen:

var

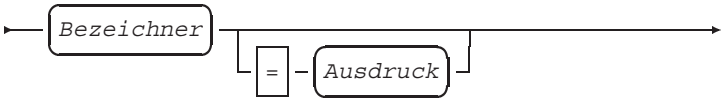


Eine solche Deklaration kann an jeder Stelle im Programmtext stehen, an der eine Anweisung stehen darf. Die deklarierten Variablen sind ab dieser Stelle im Programmtext verwendbar.

✘ Anders als in vielen gebräuchlichen Programmiersprachen sind Variablen in WMLScript immer für die gesamte umschließende Funktion deklariert und nicht etwa nur innerhalb eines umschließenden Blocks.

✘ Auch wenn Deklarationen fast an beliebiger Stelle im Programm stehen können, sollten Sie sie immer am Anfang zusammenfassen. Sie behalten so einen erheblich besseren Überblick über die gültigen Deklarationen.

Jede Variablendeklaration benennt einen Namen für die Variable und gibt eventuell einen Initialwert als Ausdruck an:



Für den Bezeichner der Variablen gelten drei Regeln:

1. Er muß mit einem Groß- oder Kleinbuchstaben aus ‘`A`’ bis ‘`Z`’ und ‘`a`’ bis ‘`z`’ beginnen und kann dann aus beliebig vielen Groß- oder Kleinbuchstaben sowie den Ziffern von ‘`0`’ bis ‘`9`’ bestehen. Damit ist `Karte100` gültig, während `10Zeilen` nicht erlaubt ist.

✘ Im Gegensatz zu anderen Programmiersprachen sind Sonderzeichen wie _ nicht in Variablennamen erlaubt. Ebenfalls können Sie keine deutschsprachigen Umlaute verwenden.

2. Der Bezeichner darf nicht aus der Menge der reservierten Schlüsselworte stammen. Sie finden eine vollständige Liste dieser Namen in Abbildung 12.1 auf der nächsten Seite.

3. Der Bezeichner darf nicht für eine vorher schon deklarierte Variable verwendet sein.

Eine Variablendeklaration besteht entweder lediglich aus der Angabe eines Namens oder aus einem Namen und einem Initialisierungsausdrucks. Er wird bei der Deklaration der Variablen zugewiesen und muß ein gültiger WMLScript-Ausdruck sein, der einen Wert ergibt. Ist kein Initialisierungsausdruck vorhanden, enthält die Variable die leere Zeichenkette als Wert. Beispiele für Deklarationsanweisungen sind:

Gültige Schlüsselworte von WMLScript
access agent break continue div div= domain else equiv extern for function header http if isvalid meta name path return typeof use user var while url
Ungenutzte Schlüsselworte von WMLScript
delete lib null void in new this with
Für Erweiterungen reservierte Schlüsselworte
case catch class const debugger default do enum export extends finally import private public sizeof struct super switch throw try

Abbildung 12.1 *Reservierte Worte in WMLScript*

```
var a;
var b=10;
var c=a+b;
var d=invalid;
```

Da ja nach `var` mehrere Deklarationen folgen dürfen, könnten Sie diese vier Anweisungen auch zu einer zusammenfassen:

```
var a, b=10, c=a+b, d=invalid;
```

Keineswegs ist eine Art »Mehrfachinitialisierung« möglich – `a` enthält in dem Beispiel initial die leere Zeichenkette. ✘

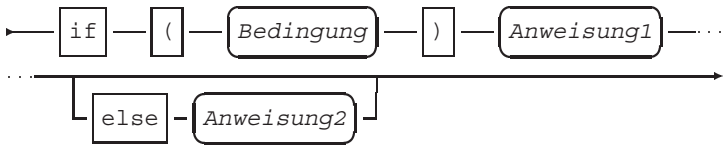
Während die Mehrfachdeklaration auf den ersten Blick praktisch ist, weil sie Tipparbeit spart, ist es sehr empfehlenswert, nur zwingend zusammen vorhandene Variablen auf einmal zu deklarieren. Damit erleichtern Sie sich spätere Änderungen am Programmtext. ✘

12.6 Entscheidungen

if

else

Die einfachste Art, den Kontrollfluß in einem Programm zu steuern ist die Auswahl eines Anweisungszweigs auf der Basis einer Bedingung. In WMLScript hat die entsprechende `if-else`-Anweisung die folgende Form:



Bedingung ist ein Ausdruck, der zu einem Wahrheitswert oder `invalid` verrechnet werden kann. Ergibt er `true`, wird *Anweisung1* ausgeführt. Bei `false` oder `invalid` kommt *Anweisung2* zur Ausführung oder der Programmcode nach der `if`-Anweisung.

Das folgende Beispiel überprüft, ob bei der Division von `a` durch 2 ein Rest bleibt, der Variableninhalt also eine gerade Zahl ist. Wir verwenden dabei eine vordefinierte Funktion – `Dialogs.alert` – die eine Mitteilung auf dem Display anzeigt und auf eine Bestätigung durch den Benutzer wartet.

```
if ((a%2)==0) {
  Dialogs.alert("a ist gerade");
} else {
  Dialogs.alert("a ist ungerade");
}
```

Anweisung1 und *Anweisung2* müssen jeweils genau eine Anweisung sein. Eine Folge von mehreren Anweisungen müssen Sie hier also als Block zusammenfassen.

✘ Es empfiehlt sich mit Blick auf spätere Erweiterungen, auch dann einen eigenen Block zu notieren, wenn nur eine einzelne Anweisung vorhanden ist.

✘ Während der `else`-Zweig mit *Anweisung2* entsprechend dem Syntaxdiagramm optional ist, muß nach dem `if` immer eine Anweisung folgen. Während als `if (a==1) else b=10;` nicht akzeptiert wird, könnten Sie sich mit `if (a==1) ; else b=10;` oder `if (a==1)  else b=10;` behelfen oder die Bedingung negieren: `if (!a==1) b=10;`

✘ `else` bezieht sich immer auf das letzte `if` davor. Das folgende Beispiel – ohne Verwendung von Blöcken – verdeutlicht das mit Einrückungen.

```
if (a==10)
  if (b==10)
    Dialogs.alert("erster Zweig");
  else
    Dialogs.alert("zweiter Zweig");
```

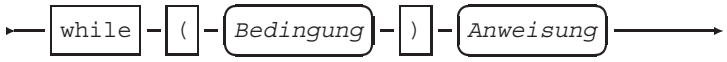
Falls `a` ungleich 10 ist, wird das Programm also ohne weitere Ausgabe fortgeführt. Nur wenn `a` gleich 10 ist und `b` ungleich 10 kommt es zur Anzeige der Mitteilung »zweiter Zweig«.

Die Ihnen vielleicht von anderen Sprachen wie C, Java oder JavaScript her bekannte mehrfache Fallunterscheidung mit `switch` und `case` gibt es in WMLScript nicht. ✘

12.7 Schleifen

Es gibt zwei Arten von Schleifen in WMLScript, die jeweils eine Anweisung sind. `while` leitet eine abweisende Schleife ein, deren Wiederholungsbedingung vor Beginn des Schleifenkörpers getestet wird und `for` ist die Zählschleife, die zusätzlich einen Ausdruck nach Ausführung des Schleifenkörpers durchführt. Eine nichtabweisende Schleife, wie Sie sie vielleicht aus anderen Programmiersprachen als `repeat...until` oder `do...while` kennen, gibt es in WMLScript nicht.

Die `while`-Anweisung hat die folgende Syntax: *while*



Bedingung ist ein Ausdruck, der einen Wahrheitswert oder `invalid` liefert. *Anweisung* wird ausgeführt, wenn die *Bedingung* den Wert `true` lieferte. Danach wird sie erneut geprüft. Liefert sie `false` oder `invalid`, wird das Programm nach der `while`-Anweisung fortgeführt.

Nach `while` ist genau eine Anweisung als Schleifenkörper möglich. Sie müssen also eine Folge von Anweisungen innerhalb der Schleife als Block zusammenfassen.

Auch wenn nur eine einzelne Anweisung notwendig ist, empfiehlt sich der Aufwand, auch hierfür einen eigenen Block zu notieren. Falls Sie später eine weitere Anweisung dem Schleifenkörper hinzufügen, können Sie so die Blockbildung nicht mehr vergessen und ersparen sich eine mühsame Fehlersuche. ✘

Als einfaches Beispiel soll ein Programm die Zahlen von 1 bis 9 aufaddieren:

```
var i = 1 ;
var sum = 0;
while (i<10) {
  sum+=i;
```

```
    i++;
  }
  Dialogs.alert("Summe: " + sum);
```

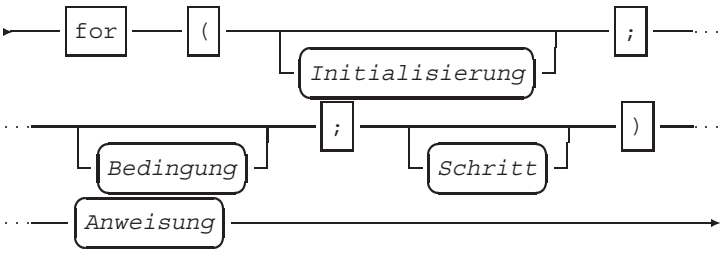
Die Variablen `i` und `sum` enthalten den aktuellen Summanden und die Gesamtsumme. Die `while`-Schleife testet zunächst, ob `i` kleiner 10 ist und führt entsprechend den Schleifenkörper aus. Die Bedingung ist genau dann zutreffend, wenn `i` zwischen 1 und 9 liegt. Im Schleifenkörper wird die Summe gebildet und der Zähler um 1 erhöht.

✘ *Bedingung* muß ein Ausdruck sein, der einen Wahrheitswert *liefert* – er muß nicht in Gesamtheit ein Ausdruck sein. Mit dem `,`-Operator können ja mehrere Berechnungen zu einem Ausdruck zusammengefaßt werden. Ein solcher Ausdruck liefert dann einen Wahrheitswert, wenn sein *letzter* Teil ein Ausdruck ist. Man könnte also beispielsweise folgendes schreiben:

```
while (i++,i<10) {
  sum+=i;
}
```

Damit enthält *Bedingung* zwei Berechnungen: Die Veränderung von `i` und die Ermittlung eines Wahrheitswerts. Ersetzt man damit die Schleife im obigen Beispiel ist die gelieferte Summe nur noch 44, da die 1 als ursprünglicher Inhalt von `i` am Beginn der Auswertung von `i++,i<10` mit 2 überschrieben wird.

for Die Zählschleife mit `for` hat die folgende Form:



Bevor die Bearbeitung der Schleife beginnt, kommen die Berechnungen in *Initialisierung* zur Ausführung. Sollten hier mehrere Anweisungen stehen, müssen sie mit dem `,`-Operator zusammengefaßt sein. Die Initialisierung kann auch Variablendeklarationen enthalten, die ab der Schleife gültig sind.

Bedingung ist ein Wahrheitswert. Ergibt seine Auswertung `true`, wird der Schleifenkörper ausgeführt, bei `false` oder `invalid` geht der Programmablauf nach der Schleife weiter.

Schritt ist eine Berechnung, die nach der Ausführung des Schleifenkörpers, also von *Anweisung* ausgeführt wird. Seine Aufgabe ist es, den Variablenzustand zu ändern, der die Fortführung der Schleife durch den Test in *Bedingung* entscheidet. Bei einer herkömmlichen Zählschleife bedeutet das das Herauf- oder Herabzählen eines Zählers.

Tatsächlich läßt sich die `for`-Schleife auf `while` abbilden:

```
Initialisierung;
while (Bedingung) {
  Anweisung
  Schritt
}
```

Der Vorteil von `for` liegt darin, daß alle Berechnungen zur Initialisierung und Fortführung der Schleife syntaktisch an einem Ort konzentriert sind.

Auch bei `for` empfiehlt sich der Aufwand für einen eigenen Block auch bei einer einzelnen Anweisung im Schleifenkörper im Hinblick auf spätere Erweiterungen. ✘

Das obige Beispiel zum Aufsummieren der Zahlen von 1 bis 9 läßt sich mit `for` so ausdrücken:

```
for(i=1, sum=0; i<10; i++) {
  sum+=i;
}
```

Wäre `i` noch nicht deklariert, könnte die Initialisierung auch `var i=1, sum=0` lauten. Um nur die ungeraden Zahlen zwischen 1 und 9 aufzusummieren, müßten Sie die »Schrittweite« des Zählers entsprechend vergrößern:

```
for(i=1, sum=0; i<10; i+=2) {
  sum+=i;
}
```

Die Veränderung des Zählerzustands kann natürlich auch innerhalb des Schleifenkörpers stattfinden. Die Schrittweite im Beispiel könnten Sie auch dort durch eine weitere Zuweisung vergrößern: ✘

```
for(i=1, sum=0; i<10; i++) {
  sum+=i;
  i++;
}
```

Es kann Zählerzustände geben, die sich nur im Schleifenkörper verändern lassen, beispielsweise, wenn sich ihre Art erst durch ein `if...else` festlegen läßt. Falls es sich aber um einen so einfachen Zähler wie im Beispiel handelt, sollten Sie seine Veränderung im Schleifenkopf konzentrieren, um spätere Änderungen ohne schwer zu findende Fehler durchführen zu können.

Wie Sie am Syntaxdiagramm erkennen können, sind alle drei Komponenten der `for`-Schleife optional. Für das Zählerbeispiel ergeben sich folgende Varianten:

- `for(; i<10; i++)`: `i` und `sum` sind vorher schon deklariert und korrekt gesetzt worden. Vielleicht übernehmen sie das Ergebnis einer vorherigen Berechnung.
- `for(i=0, sum=0; ; i++)`: Die Entscheidung über die Beendigung der Schleife wird im Schleifenkörper getroffen. Dazu lernen Sie gleich die Anweisung `break` kennen. Tatsächlich ist in WMLScript definiert, daß eine leere Bedingung bei `for` als `true` gewertet wird und sich praktisch eine Endlosschleife ergibt.
- `for(i=0, sum=0; i<10 ; )`: Der Zählerzustand wird ausschließlich im Schleifenkörper – beispielsweise als `sum+=i; i++;` – verändert.

Die Anweisungen im Schleifenkörper werden normalerweise immer komplett pro Iteration ausgeführt. Es gibt Algorithmen, bei denen es aus verschiedenen Gründen notwendig wird, die Ausführung des Schleifenkörpers abzubrechen. WMLScript kennt zwei Anweisungen, die das leisten:

break

- `break` unterbricht die Ausführung des Schleifenkörpers und führt das Programm nach der Schleife fort. Insofern bricht `break` die gesamte umschließende Scheife ab. Im folgenden Beispiel ist so ein einfacher Zähler implementiert:

```
var counter = 0;
while (true) {
  if (!Dialogs.confirm("Zähler = "+counter,
                       "Weiter", "Schluß"))
    break;
  counter++;
}
```

In einer Endlosschleife wird die Variable `counter` nach oben gezählt. Der Benutzer kann jeweils in einem mit `Dialogs.alert` erzeugten Auswahldialog zwischen Weiterzählen und Abbrechen wählen. Die Parameter der Funktion sind ein Mitteilungstext, der den aktuellen Zähler ausgibt und die Beschriftung der Bestätigung und der Abbruchauswahl. Liefert die Funktion `false`, hatte der Benutzer »Schluß« gewählt und die `break`-Anweisung beendet die ganze Schleife.

continue

- `continue` unterbricht die Ausführung des Schleifenkörpers und kehrt zur Überprüfung der Bedingung für die Fortführung von `while` oder `for` zurück. Insofern bricht `continue` die aktuelle Iteration, aber nicht die gesamte umschließende Scheife ab.

  Das folgende Beispiel verändert so die Summation der Zahlen von 1 bis 9:

```
while (i<10) {
  sum+=i;
  if (Dialogs.confirm("Summand = "+i,
                      "Nochmal", "Weiter"))
    continue;
  i++;
}
Dialogs.alert("Summe: " + sum);
```

  Nach jeder Aufsummierung wird der Benutzer gefragt, ob der aktuelle Inhalt von `i` nochmals verrechnet werden soll. Falls ja, bricht `continue` den Schleifenkörper ab und bewertet `i<10` erneut. Dieser Test wird in jedem Fall `true` ergeben, so daß nochmals addiert wird. Nur bei einer Auswahl von »Weiter« geht die Ausführung des Schleifenkörpers weiter und `i` wird um 1 erhöht.

✘

Während in anderen Programmiersprachen `break` und `continue` auch an anderen Stellen verwendet werden können, sind sie in WMLScript ausschließlich innerhalb von `while`- und `for`-Schleifen erlaubt.

Mit Datentypen und Kontrollstrukturen könnte man schon jede erdenkliche Berechnung mit WMLScript durchführen. Es ist aber sinnvoll, wiederverwendbare Programmteile als Funktionen zu definieren. Die Details dazu sind Thema des nächsten Kapitels.

13 Funktionen

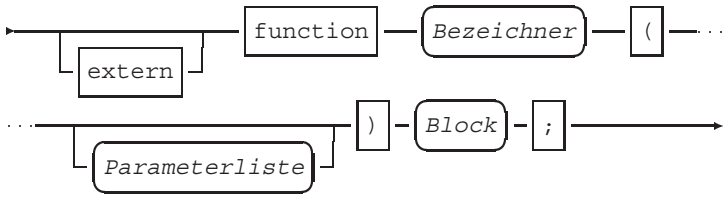
Mit Funktionen fassen Sie die Funktionalität von Programmabschnitten zusammen. Eine Funktion kann von anderen Programmteilen unter Angabe von Parametern aufgerufen werden und ein Ergebnis zurückliefern.

Ein WMLScript-Programm besteht eigentlich aus einer Liste von Funktionen, von denen zumindest eine über eine WML-Seite aufrufbar sein muß. Die Sammlung der in einer WMLScript-Datei zusammengefaßten Funktionen nennt man *Übersetzungseinheit*.

In diesem Kapitel lernen Sie die Deklaration von Funktionen und die verschiedenen Aufrufformen kennen.

13.1 Deklaration

Jede Funktion muß deklariert werden. Im Programmtext kann die Deklaration auch nach der Verwendung auftreten, da der WMLScript-Compiler solche Referenzen am Ende der Übersetzung vervollständigen kann. In einer Funktionsdeklaration legen Sie Namen und Parameter sowie die Anweisungen der Funktion fest. Die vollständige Syntax lautet:



Mit der Verwendung von `extern` erlauben Sie den Aufruf der deklarierten Funktion von außerhalb des Programms. Die genauen Mechanismen dieser externen Aufrufe finden Sie in Abschnitt 13.3 auf Seite 129 beschrieben. *extern*

Auf das Schlüsselwort `function` folgt der Name der Funktion. Es muß sich um einen Bezeichner handeln, für den die gleichen Regeln wie für Variablennamen gelten. Diese waren in Abschnitt 12.5 auf Seite 118 beschrieben. *function*

Von (und) umschlossen schließt sich die Liste der Parameter der Funktion an. Diese Liste kann auch leer sein – die Klammern müssen Sie dennoch verwenden. Die Parameterliste besteht aus durch Kommas getrennte Variablennamen und hat die folgende Syntax:

Die Parameter werden dadurch als Variablen mit Gültigkeit innerhalb der Funktion deklariert. Ihr Wert ergibt sich aus den beim Funktionsaufruf übergebenen Daten.

✘ Da WMLScript anders als Programmiersprachen wie etwa Pascal oder Java ungetypt ist, müssen Sie keinerlei Angaben über die Typisierung der Parameter machen.

An die Parameterliste schließt sich ein Block an, der die Anweisungen enthält, die in der Funktion ausgeführt werden sollen. Darin vorhandene Variablendeklarationen sind nur innerhalb der Funktion gültig und werden bei ihrer Beendigung gelöscht.

Damit läßt sich als Beispiel eine Funktion deklarieren, die zwei Parameter erhält und diese als Mitteilung ausgibt:

```
function echo2(par1, par2) {
  Dialogs.alert("Parameter: " + par1 +
                " und " + par2);
}
```

Sie läßt sich beispielsweise mit `echo2("DEM", "EUR");` aufrufen – mehr dazu in Abschnitt 13.3 auf der nächsten Seite. Beim Start der `echo2`-Funktion wird damit an die Variable `par1` die Zeichenkette »`DEM`« gebunden und »`EUR`« an die Variable `par2`. Das Programm zeigt damit die Meldung »`Parameter: DEM und EUR`« an.

Wollte man nun auch eine Funktion haben, die drei Parameter ausgibt, müßte sie einen anderen Namen erhalten und eine entsprechende Parameterliste deklarieren:

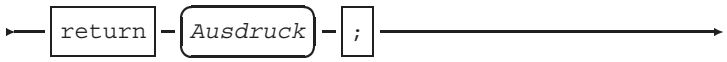
```
function echo3(par1, par2, par3) {
  Dialogs.alert("Parameter: " + par1 +
                " und " + par2 +
                " und " + par3);
}
```

13.2 Funktionsergebnisse

Funktionen zeichnen sich – zumindest mathematisch – dadurch aus, daß sie einen Ergebniswert haben, der aus den Parametern berechnet wird. Funktionen in WMLScript folgen dieser Sicht und geben *immer* einen Wert zurück. Falls nichts anderes vermerkt ist, handelt es sich bei diesem Wert um die leere Zeichenkette.

return

Mit der Anweisung `return` können Sie einen programmierten Ergebniswert an den Aufrufer zurückliefern. Die Anweisung hat die folgende Form:



Ausdruck ist dabei ein gültiger Ausdruck beliebigen Typs. Um beispielsweise Beträge von DM in Euro umzurechnen könnte man folgende Funktion deklarieren:

```
function DEMtoEUR(DEM) {
  return DEM/1.95583;
}
```

Das Ergebnis eines Aufrufs von `DEMtoEUR(1.0)` ist der Wert 0.51129186 was einer Deutschen Mark in Euro entspricht. Der Typ des Funktionsergebnisses ergibt sich in der ungetypten Sprache WMLScript erst zur Laufzeit aus der Typisierung des Ausdrucks.

✘

Da die Parameter ebenfalls ungetypt sind, ist nicht sichergestellt, daß verwendete Ausdrücke überhaupt auswertbar sind. So verhindert nichts einen Aufruf `DEMtoEUR("WMLScript")` mit der Folge, daß eine Zeichenkette durch eine Zahl dividiert werden soll. Das Kapitel 14 auf Seite 133 beschreibt das Verhalten von WMLScript in einem solchen Fall. Sie können mit entsprechenden Überprüfungen der Typisierung und Konvertierbarkeit der Parameterwerte, beispielsweise aus der `Lang`-Bibliothek, die Sicherheit Ihrer Funktionen deutlich erhöhen.

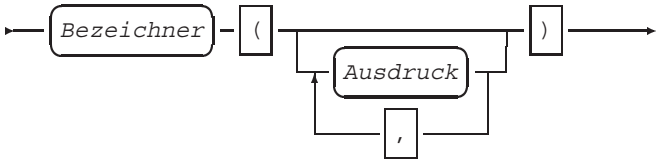
13.3 Funktionsaufruf

Der Aufruf einer Funktion bildet eine Anweisung in WMLScript. Sie haben weiter oben schon die Aufrufanweisung `echo2("DEM", "EUR");` gesehen. Der Aufruf besteht aus einem Funktionsnamen und einer Parameterliste aus durch Kommas getrennten Werten. Im Beispiel handelt es sich um zwei konstante Ausdrücke vom Typ Zeichenkette.

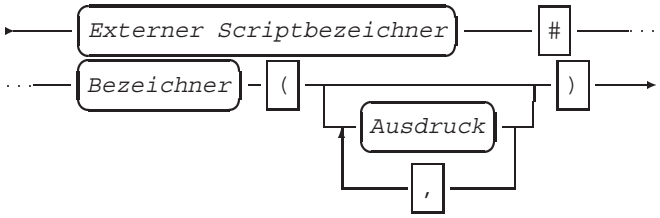
Beim Aufruf findet die Bindung der Parameterwerte an die deklarierten Parametervariablen statt. Stammt der Parameterwert aus einer Variablen, kann man fragen, ob denn der Inhalt der Variablen übergeben wird oder eine Referenz auf die Variable. Bei einer Referenz würde eine Änderung an der Parametervariable auch die referenzierte Variable des Aufrufers ändern – ein Verhalten, daß als *call by reference* bezeichnet wird. In WMLScript wird jedoch immer der Wert des Ausdrucks verwendet – der Fachbegriff lautet dafür *call by value* – und das zurückgegebene Ergebnis einer Funktion muß vom Aufrufer explizit in eine Variable gespeichert werden.

WMLScript unterscheidet drei Arten von Funktionsaufrufen. Sie unterscheiden sich darin, wo die aufgerufene Funktion definiert ist.

1. *Lokale* Funktionsaufrufe beziehen sich auf Funktionen, die in derselben Übersetzungseinheit deklariert sind, in der der Aufrufer steht. Ihre Syntax ist:



2. *Externe* Funktionsaufrufe beziehen sich auf Funktionen aus anderen Übersetzungseinheiten, die im Web abgelegt sind. Hier ist die Syntax:

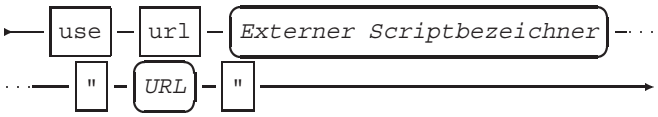


`Externer Scriptbezeichner` ist ein symbolischer Name für ein WMLScript-Programm im Web, für den die üblichen Regeln aus Abschnitt 12.5 auf Seite 118 für Bezeichner gelten.

Damit er verwendbar ist, muß ihm, ähnlich wie bei einer Variablen, eine URL zugewiesen werden, für die er steht. Da es sich aber eben nicht um eine Variable handelt, die sich verändern kann, sondern um eine Abkürzung, die schon

während der Übersetzungszeit aufgelöst wird, muß die Bindung einer URL an den Bezeichner anders geschehen. Vor der Verwendung fügen Sie dafür die Anweisung `use url` in Ihr Programm sein. Da die Anweisung sich an den Compiler wendet, spricht man hier auch von einer *Pragma*-Anweisung. Sie hat die folgende Form:

use url



Externer Scriptbezeichner ist der Name, unter dem die Funktionssammlung aus dem Netz im Programm verwendet werden soll. *URL* ist eine normale Netzadresse, allerdings muß sie sich auf ein komplettes Dokument beziehen, darf also nicht mit einem #-Fragment auf Teildokumente verweisen.

Um beispielsweise eine Bibliothek mit verschiedenen Umrechnungsfunktionen an der Adresse `http://wap.cs.tu-berlin.de/~tolk/finanzlib.wmlsc` zu nutzen, wird ihr im folgenden Beispiel zunächst der Bezeichner `devisen` zugewiesen.

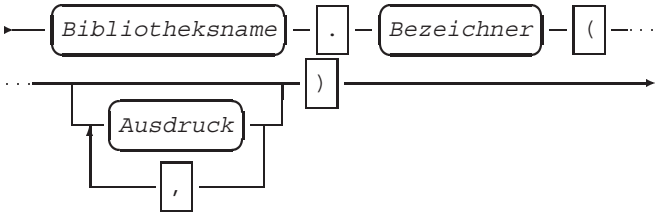
```
use url devisen
 "http://wap.cs.tu-berlin.de/~tolk/finanzlib.wmlsc";

function DEMtoLIT (DEM) {
  return devisen#EURtoLIT(devisen#DEMtoEUR(DEM));
}
```

`DEMtoLIT` kombiniert zwei Funktionen daraus und ruft sie jeweils mit vorangestellten `devisen#` auf.

Damit eine Funktion auf diese Weise von außen verwendbar ist, muß sie mit `extern` deklariert sein. ✘

3. *Bibliotheksfunktionen* beziehen sich auf den festen Satz vorgegebener WMLScript-Bibliotheken und die darin definierten Funktionen. In Kapitel 15 auf Seite 139 finden Sie die momentan definierten WMLScript-Bibliotheken beschrieben ([55]). Der Aufruf einer Funktion daraus hat folgende Syntax:



Sie haben solche Aufrufe schon an einigen Stellen in den Beispielen vorgefunden, so ist `Dialogs.alert("Hallo");` der Aufruf der Funktion `alert` aus der Standardbibliothek `Dialogs`.

Damit ist die Programmiersprache WMLScript prinzipiell komplett dargestellt. Im folgenden Kapitel finden Sie mögliche Fehler bei der Programmausführung beschrieben.

14 Fehler

In einem WMLScript-Programm können Fehler auftreten, die beispielsweise durch falsche Eingaben, Fehlfunktionen des Netzes oder falsche Programmierung entstehen.

Im Gegensatz zu anderen Sprachen kennt WMLScript keinen Mechanismus wie die Exceptions, mit denen sich strukturiert die Reaktion eines Programms auf Fehlersituationen notieren läßt. Zwei Arten von Fehlern und Reaktionen darauf werden unterschieden:

- *Behebbare Fehler*, beispielsweise eine Division durch 0, bewirken, daß WMLScript bei der Auswertung einen speziellen Ergebniswert errechnet und so die Fehlersituation dem Programm signalisiert.
- *Fatale Fehler*, beispielsweise bei Belegung des kompletten vorhandenen Speichers, sind nicht behebbar und führen zum Abbruch des Programms. Das WAP-Gerät muß darauf geeignet mit einer Mitteilung an den Benutzer reagieren.

Die folgenden Abschnitte geben einen Überblick auf Fehlersituationen und ihre Art.

14.1 Automatisch behobene Fehler

Automatisch behebbare Fehler werden von einer WMLScript-Implementierung selber korrigiert. Sie machen sich nur dadurch bemerkbar, daß das Ergebnis von Berechnungen anders als vielleicht erwartet ist. Dennoch läßt sich mit ihnen weiterarbeiten. Der Wert `invalid` dient in fast allen Fällen zum Vermerken des Fehlers in einer Berechnung.

14.1.1 Fehler bei arithmetischen Berechnungen

Bei der Auswertung von arithmetischen Ausdrücken können sich Fehler dadurch ergeben, daß nicht definierte Berechnungen durch-

geführt werden sollen oder Ergebnisse außerhalb der Wertebereiche der Zahlentypen von WMLScript liegen. Mögliche Fehler sind:

- Divide by zero – »Division durch 0«

  Das Ergebnis einer Division durch 0 wie in `a=0; b=10/a;` ist nicht definiert. Der Fehler kann bei der Division von Ganzzahlen mit `div` oder Fließkommazahlen mit `/` sowie bei der Bildung eines Divisionsrests mit `%` auftreten. Das Ergebnis eine solchen Ausdrucks ist `invalid`.

- Integer overflow – »Überlauf bei Ganzzahl«

  Eine Berechnung würde einen Ganzzahlwert außerhalb des Wertebereichs ergeben, wie beispielsweise `Lang.maxInt()+1` oder `-2-Lang.maxInt()`. Der Ausdruck hat als Ergebnis `invalid`.

- Floating-point overflow – »Überlauf bei Fließkommazahl«

  Bei einer Berechnung – beispielsweise `Float.maxFloat()*2` würde der Wertebereich für Fließkommazahlen überschritten. Der Ausdruck hat den Wert `invalid`.

- Floating-point underflow – »Zu kleine Fließkommazahl«

  Der Wert eines Ausdrucks ist zu klein, um noch darstellbar zu werden. Ein Beispiel ist `Float.minFloat()/Float.maxFloat()`. Der Wert des Ausdrucks ist dann `0.0`.

14.1.2 Fehler bei Fließkommakonstanten

Die Fähigkeit zur Berechnung von Fließkommazahlen ist implementierungsabhängig. Das in WMLScript verwendete Zahlenformat umfaßt auch ungültige Werte und Unendlich. Dementsprechend können sich bei der Verwendung von Konstanten zur Laufzeit Fehler ergeben.

- Illegal floating-point reference – »Unzulässiger Zugriff auf Fließkommazahl«

  WMLScript-Implementierungen müssen Fließkommazahlen nicht unterstützen. Die Funktion `Lang.float()`, die Sie in Abschnitt 15.1 auf Seite 142 beschrieben finden, gibt darüber Auskunft. Wird eine Fließkommakonstante angesprochen, obwohl die benutzte WMLScript-Implementierung keine Fließkommazahlen beherrscht, wird der Fehler ausgelöst. Der Wert des Ausdrucks ist dann `invalid`.

- Not a number floating-point constant – »Ungültige Fließkommakonstante«

  Der Bytecode-Compiler hat eine Konstante erzeugt, die als Non-A-Number nach dem in WMLScript verwendeten Fließkommastandard markiert ist ($0.0/0.0$). Der fehlerhafte Ausdruck hat den Wert `invalid`.

- Infinite floating-point constant – »Unendliche Fließkommakonstante«

  Der Bytecode-Compiler hat eine Konstante erzeugt, die als positiv ($1.0/0.0$) oder negativ ($-1.0/0.0$) unendlich nach dem in WMLScript verwendeten Fließkommastandard markiert ist. Der Ausdruckswert ist dann `invalid`.

14.1.3 Fehler bei Konvertierungen

Bei der Konvertierung von Werten nach den Regeln, die Sie aus Abschnitt 11.2 auf Seite 104 kennen, kann es bei der Umwandlung von Zeichenketten vorkommen, daß die Konvertierung scheitert, weil der mögliche Wertebereich von Zahlen über- oder unterschritten wird.

- Integer too large – »Ganzzahl zu groß«

  Die Erzeugung einer Ganzzahl scheitert, weil die Zahl den Wertebereich nach oben oder nach unten überschreitet. Ein Beispiel wäre `a="2500000000";` – der Wert ist größer als die größte Ganzzahl. Die erzeugte »Zahl« ist stattdessen `invalid`.

- Floating-point too large – »Fließkommazahl zu groß«

  Die Erzeugung einer Fließkommazahl scheitert, weil die Zahl zu groß ist. Beispiele wären `a="1e100";` und `a="-1e100";` – die Werte sind größer bzw. kleiner als die größte und kleinste Fließkommazahl. Die Konvertierung ergibt `invalid`.

- Floating-point too small – »Fließkommazahl zu klein«

  Die Erzeugung einer Fließkommazahl scheitert, weil die Zahl zu klein ist, um sie noch darstellen zu können. Beispiele sind `a="1e-100";` und `a="-1e-100";` – die Werte liegen unter der Genauigkeit von Fließkommazahlen in WMLScript. Die stattdessen erzeugte Zahl ist `0.0`. Ein eventuell vorhandenes Vorzeichen spielt keine Rolle, da in WMLScript $+0.0$ und -0.0 nicht unterschieden werden.

14.2 Nicht behebbare Fehler

Im Gegensatz zu den Berechnungsfehlern lassen sich viele Fehler nicht automatisch beheben, beispielsweise weil sie den Programmablauf unmöglich machen. In diesem Fall wird die Ausführung des gesamten Programms abgebrochen. Die Umgebung, also das WAP-Gerät, wird über den Abbruch informiert und kann beispielsweise eine entsprechende Meldung erzeugen.

14.2.1 Fehler im Programmcode

WMLScript wird vor der Ausführung übersetzt. Dabei entsteht ein sogenannter Bytecode, der dem ausführbaren Code für reale Prozessoren ähnelt. Vielleicht kennen Sie dieses Konzept von der Programmiersprache Java.

Dieser WMLScript-Bytecode kann nun auf einem Prozessor beliebiger Bauart ausgeführt werden, solange dort ein entsprechenden Bytecode-Interpreter vorhanden ist. Bei Java spielt die *Java Virtual Machine* JVM diesen Part.

Mögliche Fehler, die die Ausführung von Bytecode unmöglich machen sind:

- Unable to load compilation unit – »Bibliothek nicht ladbar«

  Eine Bibliothek kann nicht geladen werden, beispielsweise aufgrund eines Netzwerkfehlers oder einer falschen Konfiguration.

- Verification failed – »Überprüfung fehlgeschlagen«

  Vor der Ausführung von Bytecode fremder Funktionen wird dieser auf Korrektheit hin überprüft. Aufgrund von Übertragungsfehlern, Fehlkonfigurationen etc. kann festgestellt werden, daß es sich nicht um gültigen Bytecode handelt.

- Access violation – »Zugriffsschutz verletzt«

  Beim Laden einer Bibliothek wurde festgestellt, daß darin eine Zugriffsbeschränkung besteht, die der Aufrufer nicht erfüllt.

- External function not found – »Bibliotheksfunktion nicht gefunden«

  Ein Aufruf einer Bibliotheksfunktion scheitert, weil sie nicht in der Bibliothek zu finden ist.

- Invalid function arguments – »Falsche Parameteranzahl«

  Beim Aufruf einer Bibliotheksfunktion wurde die falsche Anzahl Parameter benutzt. Der Fehler kann nicht durch den Compiler überprüft werden, da die Bibliothek zur Übersetzungszeit nicht vorliegen muß.

- Fatal library function error – »Fehlfunktion in Bibliothek«

  Eine in einer Bibliothek aufgerufene Funktion ist mit einem nicht behebbaren Fehler abgebrochen worden. Auf diese Weise führen Fehler zum Abbruch des Gesamtprogramms, da sie sich nach »außen« fortpflanzen.

- Stack underflow – »Stapel leer«

  Wahrscheinlich aufgrund eines fehlerhaften Compilers enthält der Programmcode eine fehlerhafte Benutzung der Speicherstrukturen des Bytecode-Interpreters.

14.2.2 Programmabbrüche

Während der Ausführung von Code kann das Programm selber, der Nutzer oder das WAP-Gerät einen Programmabbruch bewirken.

- Programmed abort – »Programmierter Abbruch«

  Das ausgeführte Programm hat die Funktion `Lang.abort` aufgerufen.

- Initiated by the user – »Abbruch durch Nutzer«

  Der Benutzer hat die Ausführung unterbrochen. Die Möglichkeiten dazu hängen vom verwendeten WAP-Gerät ab.

- Initiated by the system – »Abbruch durch System«

  Das System hat entschieden, die Ausführung abzubrechen. Ein Beispiel für eine solche Situation wäre ein niedriger Füllstand des Akkus des WAP-Geräts.

14.2.3 Speicherfehler

Durch Programmierfehler und Geräteeinschränkungen kann es zu Problemen mit der Benutzung des Speichers des WAP-Geräts durch ein WMLScript-Programm kommen.

- Stack overflow – »Überlauf des Stapels«

  Ein WMLScript-Interpreter enthält eine dynamische Datenstruktur, in der die lokalen Variablen von Funktionen verwaltet werden. Kommt es aufgrund eines Programmierfehlers oder durch ungeschickte Algorithmen zu extensiv vielen geschachtelten Aufrufen von Funktionen, belegt dieser sogenannte »Stapel« den gesamten verfügbaren Speicher und läßt sich nicht mehr erweitern.

- Out of memory – »Kein Speicher mehr frei«

  Im WMLScript-Interpreter ist kein Speicher mehr verfügbar, beispielsweise um eine Zeichenkette länger werden zu lassen.

Sie kennen nun den Sprachkern von WMLScript und die wichtigsten Fehlermöglichkeiten. Zu WMLScript sind Standardbibliotheken definiert, die Ihnen nützliche Funktionen anbieten. Sie sind das Thema des nächsten Kapitels.

15 Kernbibliotheken in WMLScript

Dieses Kapitel gibt eine Gesamtübersicht über alle Kernbibliotheken von WMLScript. Beschrieben werden die Funktionen der Standardbibliotheken `Lang`, `Float`, `String`, `URL`, `WMLBrowser` und `Dialogs` sowie die derzeit einzige Funktion der seit WAP 1.2 spezifizierten `Crypto`-Bibliothek. Ergänzend wird auch eine proprietäre Bibliothek, die `Debug Library`, kurz vorgestellt.

15.1 Die `Lang`-Bibliothek

Diese Bibliothek liefert einige häufiger benötigte Funktionen, die vermutlich von den meisten WMLScript-Programmen benötigt werden. Die Funktionen können keinem speziellen Themenbereich zugeordnet werden.

`abs(`*`Wert`*`)`

Liefert den absoluten, d.h. positiven Wert der übermittelten Zahl als Ergebnis zurück .
Parameter:

- *Wert*: ein Zahlenwert als Ganz- oder Fließkommazahl

Ergebnis: Absolutwert einer Zahl
Beispiel:

```
var a=Lang.abs(-3.42); // liefert 3.42
var b=Lang.abs(5); // liefert 5
```

`min(`*`Wert1,Wert2`*`)`

Vergleicht die beiden Parameterwerte und liefert den kleineren der beiden als Ergebnis.
Parameter:

- *Wert1*: erster zu vergleichender Wert

- *Wert2*: zweiter zu vergleichender Wert

Ergebnis: Minimum der Parameterwerte
Beispiel:

```
var a=Lang.min(-5,2); // liefert -5
```

max(*Wert1*,*Wert2*)

Vergleicht die beiden Parameterwerte und liefert den größeren als Ergebnis.
Parameter:

- *Wert1*: erster zu vergleichender Wert
- *Wert2*: zweiter zu vergleichender Wert

Ergebnis: Maximum der Parameterwerte
Beispiel:

```
var a=Lang.max(10,20); // liefert 20
```

parseInt(*Wert*)

Wandelt die übergebende Zeichenkette (String) in einen numerischen Wert (Integer) um. Der übergebene Wert wird über ein optionales Vorzeichen (+/-) und alle aufeinanderfolgenden Ziffern bis zum ersten Auftritt einer nicht-Ziffer interpretiert.
Parameter:

- *Wert*: eine Zeichenkette (String)

Ergebnis: Numerischer Wert einer Zeichenkette
Beispiel:

```
var a=Lang.parseInt("-3.43"); // liefert a = -3
var b=Lang.parseInt("500 EUR"); // liefert b = 500
var c=Lang.parseInt("321"); // liefert c = 321
var d=Lang.parseInt("EUR 500 "); // liefert invalid
```

parseFloat(*Wert*)

Wandelt die übergebende Zeichenkette (String) in eine Fließkommazahl (Float) um. Der Parameter wird bis zum ersten Zeichen interpretiert, das nicht Bestandteil einer Fließkommazahl ist.
Parameter:

- *Wert*: eine Zeichenkette (String)

Ergebnis: Numerischer Fließkommawert einer Zeichenkette
Beispiel:

```
var a=Lang.parseFloat("-3.43"); // ergibt a = -3,43
var b=Lang.parseFloat("-.4"); // b = -0,4
var c=Lang.parseFloat("3.14e-2"); // c = 3,14e-2
var d=Lang.parseFloat("500.20 EUR"); // d = 500,20
var e=Lang.parseFloat("500,20 EUR"); // e = 500.0
var f=Lang.parseFloat("321"); // f = 321.0
var g=Lang.parseFloat("EUR 500.00"); // invalid
```

isInt(*Wert*)

Prüft den übergebenen Wert (üblicherweise ein String), ob dieser durch `parseInt` in eine Ganzzahl umgewandelt werden kann (Ergebnis: *true*) oder ob eine Konvertierung nur ein ungültiges Ergebnis liefert (Ergebnis: *false*).
Parameter:

- *Wert*: ein beliebiger Übergabewert

Ergebnis: Wahrheitswert (*true* oder *false*)
Beispiel:

```
Lang.isInt("141 EUR "); // liefert true
Lang.isInt("-3.42"); // liefert true
Lang.isInt("EUR 20"); // liefert false
```

isFloat(*Wert*)

Prüft den übergebenen Wert (üblicherweise ein String), ob dieser durch `parseFloat` in eine Fließkommazahl umgewandelt werden kann (Ergebnis: *true*) oder ob eine Konvertierung nur ein ungültiges Ergebnis liefert (Ergebnis: *false*).
Parameter:

- *Wert*: ein beliebiger Übergabewert

Ergebnis: Wahrheitswert (*true* oder *false*)
Beispiel:

```
Lang.isFloat("142 EUR"); // liefert true
Lang.isFloat("-3.42"); // liefert true
Lang.isFloat("EUR 20"); // liefert false
```

maxInt()

Diese Funktion liefert eine konstante Zahl, die den in WMLScript größten möglichen Ganzzahlwert beziffert.
Ergebnis: größte mögliche Integerzahl
Beispiel:

```
var Lang.maxInt(); // liefert 2147483647 (2^31 - 1)
```

minInt()

Diese Funktion liefert eine konstante Zahl. Diese beziffert den in WMLScript kleinsten möglichen Ganzzahlwert. Man erhält hier den negativen Wert von `maxInt()` minus Eins.
Ergebnis: kleinste mögliche Integerzahl
Beispiel:

```
var Lang.minInt(); // liefert -2147483648 (-2^31)
```

float()

Mit dieser Funktion kann abgefragt werden, ob das System die Verarbeitung von Fließkommazahlen unterstützt. Falls ja, so wird *true*, ansonsten *false* zurückgeliefert.
Ergebnis: Wahrheitswert (true oder false)
Beispiel:

```
var floatSupport=Lang.float();
// liefert je nach System true oder false
```

exit(*Wert*)

Mit dem Aufruf dieser Funktion kann die Ausführung des Bytecodes eines WMLScript-Programmes unterbrochen werden. Ein gewollter, vorzeitiger Abbruch kann hiermit z.B. aus einer Schleife heraus realisiert werden. Der Parameter wird als Rückgabewert gehandelt (siehe auch `abort`).
Parameter:

- *Wert*: ein beliebiger Übergabewert

Ergebnis: Rückgabewert oder Invalid
Beispiel:

```
// teile durch d falls d ungleich Null
if (n==0) {Lang.exit("Nenner=0");}
else {res=z/n;}
```

abort(*Wert*)

Mit dem Aufruf dieser Funktion wird die Ausführung des Bytecodes eines WMLScript-Programmes unterbrochen, wenn ein ungewolltes Ereignis, z.B. ein Fehler, aufgetreten ist. Der Parameter liefert die Fehlerbeschreibung. Im Unterschied zur Funktion `exit`, welche einen geplanten Abbruch verursacht, wird diese Funktion für unvorhergesehene Abbrüche eingesetzt.
Parameter:

- *Wert*: eine Fehlerbeschreibung (String)

Ergebnis: Rückgabewert oder Invalid
Beispiel:

```
Lang.abort("Float not supported");
```

random(*Wert*)

Ermittelt eine ganzzahlige Zufallszahl zwischen 0 und der übergebenen Zahl (jeweils einschließlich). Eine als Parameter übergebene Fließkommazahl wird vorher in einen Integerwert durch `Float.int()` umgewandelt. Der (Pseudo-)Zufallsgenerator wird vorher mit `Lang.seed()` initialisiert (siehe nächste Funktion).
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: ganzzahlige Zufallszahl von 0... Parameter
Beispiel:

```
var a=Lang.random(100); // liefert a=0...100
var b=Lang.random(5)+1; // liefert b=1...6
var c=Lang.random(3.3)*10; // liefert b=0...30
var d=Lang.random("drei"); // liefert invalid
```

seed(*Wert*)

Mit dieser Funktion wird ein (Pseudo-)Zufallsgenerator initialisiert, der zur Berechnung von `Lang.random()` eingesetzt wird. Ist der Parameter eine Fließkommazahl, so wird diese vorher durch `Float.int()` in einen Integerwert umgewandelt. Wird kein Wert übergeben, so startet der Generator nach einer im System festgelegten Regel.
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: (Pseudo-)Zufallsgenerator wird gestartet
Beispiel:

```
Lang.seed(55); // Initialisert den Zufallsgenerator
var a=Lang.random(5); // liefert a=0...5
```

characterSet()

Mit dieser Funktion kann der Zeichensatz ermittelt werden, der vom WML-Interpreter unterstützt wird. Sie gibt einen Wert zurück, der dem MIBenum-Wert der entsprechenden Zeichencodierung entspricht. Der *MIBenum*-Wert (MIB = Management Information Base) ist ein eindeutiger Zahlenwert zur Kennzeichnung standardisierter Zeichensätze. Eine komplette Liste dieser Werte finden Sie im Netz unter `ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets`.
Ergebnis: Code (Ganzzahl) des Zeichensatzes
Beispiel:

```
var zs=Lang.characterSet();
// liefert den MIBenum-Wert der
// Zeichencodierung, z.B.
// zs = 4 für latin1
// zs = 9 für iso-8859-6
// zs = 106 für utf-8
```

15.2 Die Float-Bibliothek

Diese Bibliothek enthält die gängigsten arithmetischen Funktionen für Fließkommazahlen. Das Vorhandensein einer Implementierung dieser Bibliothek ist durch die WAP-Spezifikation jedoch *nicht* zwingend vorgeschrieben, so daß der Entwickler sich nicht in jedem Fall auf die Verfügbarkeit dieser Bibliotheksfunktionen verlassen kann. Bei den Betriebssystemen heutiger WAP-Endgeräte können Sie jedoch fast immer von einer vorhandenen Fließkommaverarbeitung ausgehen.

int(*Wert*)

Diese Funktion konvertiert eine Fließkommazahl (Float) in eine Ganzzahl (Int), indem sie sämtliche Nachkommastellen abschneidet. Ist der übergebene Wert bereits ganzzahlig, so wird dieser beibehalten.
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: ganzzahliger Teil eines numerischen Wertes
Beispiel:

```
var a=Float.int(23); // liefert a=23
var b=Float.int(9.8694); // liefert b=9
var c=Float.int(-9.8694); // liefert c=-9
```

floor(*Wert*)

Mit dieser Funktion wird eine Fließkommazahl (Float) in eine Ganzzahl (Integer) konvertiert, indem stets auf die nächstkleinere Ganzzahl abgerundet wird. Ist der übergebene Wert bereits ganzzahlig, so wird dieser beibehalten.
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: auf Ganzzahl abgerundeter, numerischer Wert
Beispiel:

```
var a=Float.floor(23); // liefert a=23
var b=Float.floor(9.8694); // liefert b=9
var c=Float.floor(-9.8694); // liefert c=-10
var d=Float.ceil(2.5); // liefert d=2
var e=Float.ceil(-2.5); // liefert e=-3
```

ceil(*Wert*)

Mit dieser Funktion wird eine Fließkommazahl (Float) in eine Ganzzahl (Integer) konvertiert, indem stets auf die nächsthöhere Ganzzahl aufgerundet wird. Ist der übergebene Wert bereits ganzzahlig, so wird dieser beibehalten.
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: auf Ganzzahl aufgerundeter, numerischer Wert
Beispiel:

```
var a=Float.ceil(23); // liefert a=23
var b=Float.ceil(9.8694); // liefert b=10
var c=Float.ceil(-9.8694); // liefert c=-9
var d=Float.ceil(2.5); // liefert d=3
var e=Float.ceil(-2.5); // liefert e=-2
```

pow(*Wert1,Wert2*)

Diese Funktion berechnet nach einem durch die Implementierung bestimmten Verfahren einen Näherungswert für $Wert1^{Wert2}$.
Parameter:

- *Wert1*: ein numerischer Wert (Integer oder Float)
- *Wert2*: ein numerischer Wert (Integer oder Float). Ist Wert1 negativ, so muß dieser Wert ganzzahlig sein. Ist Wert1 gleich Null, so darf dieser Wert nicht negativ sein.

Ergebnis: Potenzfunktion zweier Werte oder *invalid*
Beispiel:

```
var a=Float.pow(-1.5,2); // liefert circa 2.25
```

round(*Wert*)

Diese Funktion konvertiert eine Fließkommazahl (Float) in eine Ganzzahl (Int) durch Runden zum nächstgelegenen Integer-Wert (kaufmännisches Runden). Ist der übergebene Wert bereits ganzzahlig, so wird dieser beibehalten.
Parameter:

- *Wert*: ein numerischer Wert (Integer oder Float)

Ergebnis: gerundeter, ganzzahliger Wert
Beispiel:

```
var a=Float.round(23); // liefert a=23
var b=Float.round(9.8694); // liefert b=10
var c=Float.round(-9.8694); // liefert c=-10
var d=Float.round(8.5); // liefert d=9
var e=Float.round(-8.5); // liefert e=8
```

sqrt(*Wert*)

Diese Funktion berechnet einen Näherungswert für die Quadratwurzel einer Zahl.
Parameter:

- *Wert*: eine Fließkommazahl (Float)

Ergebnis: Quadratwurzel eines Wertes oder *invalid*
Beispiel:

```
var a=Float.sqrt(22); // liefert für a etwa 4.6904
var b=Float.sqrt(-1); // liefert invalid
```

maxFloat()

Diese Funktion liefert die nach IEEE754 größte in WMLScript verfügbare Fließkommazahl (Float) einfacher Genauigkeit. Der Standard nach IEEE754 beschreibt Fließkommazahlen mit einer Länge von 32 Bit.
Ergebnis: größte mögliche Fließkommazahl
Beispiel:

```
var Float.maxFloat(); // liefert 3.40282347e+38
```

minFloat()

Diese Funktion liefert die nach IEEE754 kleinste (positive) in WMLScript verfügbare Fließkommazahl (Float) einfacher Genauigkeit. Noch kleinere Zahlen werden als Null dargestellt.
Ergebnis: kleinste mögliche Fließkommazahl
Beispiel:

```
var Float.minFloat(); // liefert 1.17549435e-38
```

15.3 Die `String`-Bibliothek

Mit den Funktionen dieser Bibliothek können Zeichenketten, auch *Strings* genannt, manipuliert werden. Eine Zeichenkette wird als ein indizierter Vektor von Einzelzeichen repräsentiert; das erste Zeichen beginnt mit dem Indexwert 0.

`length(String)`

Diese Funktion gibt die Länge einer Zeichenkette, also die Anzahl von Zeichen, zurück. Numerische Werte werden hierbei als Zeichenketten interpretiert.
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: Länge einer Zeichenkette oder *invalid*
Beispiel:

```
var a=String.length("wap"); // liefert a = 3
var b=String.length(69); // liefert b = 2
var c=String.length(""); // liefert c = 0
```

`isEmpty(String)`

Hiermit kann eine Zeichenkette geprüft werden, ob sie mindestestens ein Zeichen enthält (`false`) oder leer ist (`true`).
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: Wahrheitswert oder *invalid*
Beispiel:

```
var a=String.isEmpty("wap"); // liefert a = false
var b=String.isEmpty(""); // liefert b = true
```

`charAt(String,Wert)`

Diese Funktion extrahiert ein einzelnes Zeichen aus einer Zeichenkette, das sich an der angegebenen Position befindet. Liegt die mit *Wert* angegebene Position hinter dem Ende der Zeichenkette, wird ein leerer String zurückgegeben.
Parameter:

- *String*: eine Zeichenkette (String)

- *Wert*: ein numerischer, ganzzahliger Wert (Integer)

Ergebnis: Zeichen einer bestimmten Stelle oder *invalid*
Beispiel:

```
var a = "Willkommen im WAP";
var b = String.charAt(a,0); // liefert b = "W"
var c = String.charAt(a,4); // liefert c = "k"
var d = String.charAt(a,1000); // liefert d = ""
var e = String.charAt(a,"vorne"); // ergibt invalid
var f = String.charAt(1234,1); // liefert f = "2"
```

subString(*String,Wert1,Wert2*)

Diese Funktion liefert aus einer Zeichenkette einen Teilstring. Die Teil-Zeichenkette beginnt an der Position *Wert1* der Zeichenkette und ist *Wert2* Zeichen lang. Übersteigt die angegebene Länge die Anzahl vorhandener Zeichen, so wird die Länge auf den größtmöglichen Wert reduziert.
Parameter:

- *String*: eine Zeichenkette (String)
- *Wert1*: ein numerischer Wert
- *Wert2*: ein numerischer Wert

Ergebnis: Eine Teil-Zeichenkette ab Position *Wert1* mit der Länge *Wert2* oder *invalid*
Beispiel:

```
var a = "Willkommen im WAP";
var b = String.subString(a,0,10);
// liefert b = "Willkommen"
var c = String.subString(a,3,3); // c = "lko"
var d = String.subString(a,14,99); // d = "WAP"
var e = String.subString(a,6); // liefert invalid
```

find(*String1,String2*)

Ist *String2* ein Teilstring von *String1*, so gibt diese Funktion die Position in *String1* zurück, an welcher *String2* beginnt. Ist die zweite nicht in der ersten Zeichenkette enthalten, so liefert die Funktion `-1`.
Parameter:

- *String1*: eine Zeichenkette (String)

- *String2*: eine Teil-Zeichenkette von String1

Ergebnis: Position einer Teil-Zeichenkette oder *invalid*
Beispiel:

```
var a = "Willkommen im WAP";
var b = String.find(a,"komm"); // liefert b = 4
var c = String.find(a,"come"); // liefert c = -1
var d = String.find(a,1000); // liefert d = ""
var e = String.find(a,""); // liefert invalid
```

replace(*String1*,*String2*,*String3*)

Alle Vorkommen von *String2* in *String1* werden durch *String3* ersetzt. Hierbei ist zu beachten, daß die Funktion keine rekursive Ersetzung durchführt, d.h. ein einmal plazierter *String3* wird keiner weiteren Ersetzung unterzogen.
Parameter:

- *String1*: eine Zeichenkette (String)
- *String2*: eine Teil-Zeichenkette von String1 (String)
- *String3*: eine Zeichenkette (String)

Ergebnis: eine Zeichenkette mit der angegebenen Textersetzung oder *invalid*
Beispiel:

```
var a = "Willkommen im WWW";
var b = String.replace(a,"WWW","WAP");
// liefert b = "Willkommen im WAP";
var c = String.replace("AAAA","AA","A");
// liefert c = "AA"(keine Rekursion !)
var d = String.replace(a,"","im"); // invalid
```

elements(*String1*,*String2*)

Interpretiert *String1* als eine Liste von Elementen, die durch das erste Zeichen von *String2* getrennt sind und gibt deren Anzahl zurück. Da auch der leere String ("") als gültiges Element betrachtet wird, muß diese Funktion immer einen Wert von größer oder gleich 1 liefern. Von *String2* ist nur das erste Zeichen als Separator relevant.
Parameter:

- *String1*: eine Zeichenkette (String)

- *String2*: eine Zeichenkette (String)

Ergebnis: Anzahl durch *String2* getrennte Elemente in einer Zeichenkette oder *invalid*
Beispiel:

```
var a = "Hallo. Willkommen im WAP";
var b = String.elements(a," "); // liefert b = 4
var c = String.elements(a,"."); // liefert c = 2
var d = String.elements(a,"legal"); // d = 5
var e = String.elements(a,1000); // liefert e = ""
var f = String.elements(a,""); // liefert invalid
```

elementAt(*String1,Wert,String2*)

Interpretiert *String1* als eine Liste von Elementen, die durch das erste Zeichen von *String2* getrennt sind und gibt das Element zurück, welches an der Position *Wert* steht. Siehe auch Funktion `String.element`.
Parameter:

- *String1*: eine Zeichenkette (String)
- *Wert*: ein numerischer Wert
- *String2*: eine Zeichenkette (String)

Ergebnis: Element an der Stelle *Wert* in einer Elemente-Zeichenkette oder *invalid*
Beispiel:

```
var a = "Hallo. Willkommen im WAP";
var b = String.elementAt(a,2," ");
// liefert b = "Willkommen"
var c = String.elementAt(a,2,".");
// liefert c = " Willkommen im WAP"
var d = String.elementAt(a,99,""); // d = "WAP"
```

removeAt(*String1,Wert,String2*)

Interpretiert *String1* als eine Liste von Elementen, die durch das erste Zeichen von *String2* getrennt sind, entfernt das Element dieser Liste, welches an der Position *Wert* steht und gibt den verbleibenden String zurück. Wenn *Wert* kleiner Null oder größer der Anzahl Elemente ist, so wird das erste bzw. letzte Element entfernt.
Parameter:

- *String1*: eine Zeichenkette (String)

- *Wert*: ein numerischer Wert
- *String2*: eine Zeichenkette (String)

Ergebnis: Eine um das Element an der Stelle *Wert* gekürzte Elemente-Zeichenkette oder *invalid*
Beispiel:

```
var a = "Hallo. Willkommen im WAP";
var b = String.removeAt(a,1,"i");
// liefert b = "Hallo. Wm WAP"
var c = String.removeAt(a,0," ");
// liefert c = "Willkommen im WAP"
var d = String.removeAt(a,99," ");
// liefert d = "Hallo. Willkommen im "
var d = String.removeAt(a,0,""); // liefert invalid
```

`replaceAt(String1,Element,Wert,String2)`

Diese Funktion arbeitet analog zu `removeAt`, anstatt das betreffende Element jedoch zu löschen, wird es durch *Element* ersetzt.
Parameter:

- *String1*: eine Zeichenkette (String)
- *Element*: eine Zeichenkette (String)
- *Wert*: ein numerischer Wert
- *String2*: eine Zeichenkette (String)

Ergebnis: Eine Elemente-Zeichenkette mit einem an der Position *Wert* ersetzten Element oder *invalid*
Beispiel:

```
var a = "Hallo. Willkommen im WAP";
var b = String.replaceAt(a,"issen u",1,"i");
// liefert b = "Hallo. Wissen um WAP"
var c = String.replaceAt(a,"Guten Tag und",0," ");
// liefert c = "Guten Tag und Willkommen im WAP"
```

`insertAt(String1,Element,Wert,String2)`

Interpretiert *String1* als eine Liste von Elementen, die durch das erste Zeichen von *String2* getrennt sind, fügt die durch *Element* gegebene Zeichenkette an der Position *Wert* als neues Element ein und gibt den verbleibenden String zurück. Wenn *Wert* kleiner Null

oder größer der Anzahl Elemente ist, so wird an den Anfang bzw. an das Ende angefügt.
Parameter:

- *String1*: eine Zeichenkette (String)
- *Element*: eine Zeichenkette (String)
- *Wert*: ein numerischer Wert
- *String2*: eine Zeichenkette (String)

Ergebnis: Eine Elemente-Zeichenkette mit einem an der Position *Wert* eingefügten, neuen Element oder *invalid*
Beispiel:

```
var a = "Hallo. Willkommen im WAP";
var b = String.insertAt(a,"neuen",3," ");
// liefert b = "Hallo. Willkommen im neuen WAP"
var c = String.insertAt(a,"Herzlich",1,".");
// liefert c = "Hallo.Herzlich. Willkommen im WAP"
var d = String.insertAt(a,"Herzlich",1,"");
// liefert invalid
```

squeeze(*String*)

Mittels dieser Funktion wird jede in einer Zeichenkette auftretende Folge gleichartiger White Space-Zeichen auf ein einzelnes White Space reduziert. Als *White Space* gilt das Leerzeichen (SP), horizontaler und vertikaler Tabulator (TAB,VT), Zeilen- und Seitenvorschub (LF,FF) sowie Wagenrücklauf (CR). Zeichenketten ohne Folgen von White Space-Zeichen werden beibehalten.
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: String oder *invalid*
Beispiel:

```
var a = " Hallo,  WAP-User  !   ";
var b = String.squeeze(a);
// b = " Hallo, WAP-User ! "
```

trim(*String*)

Jede Folge gleichartiger White Space-Zeichen am Anfang und Ende einer Zeichenkette wird entfernt, falls vorhanden.
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: String oder *invalid*
Beispiel:

```
var a = " Hallo,  WAP-User  !   ";
var b = String.trim(a);
// liefert b = "Hallo,  WAP-User  !"
```

compare(*String1*,*String2*)

Der Rückgabewert dieser Funktion legt die ordinale Beziehung (basierend auf der Rangfolge des Zeichensatzes) der beiden gelieferten Zeichenketten zueinander dar. Ist *String1* kleiner als *String2*, so wird `-1` zurückgegeben, im umgekehrten Fall ist der Rückgabewert `1`. Sind beide Strings identisch, so erhält man als Rückgabewert `0`. Diese Funktion wird insbesondere bei der Implementierung von Sortieralgorithmen eingesetzt.
Parameter:

- *String1*: eine Zeichenkette (String)
- *String2*: eine Zeichenkette (String)

Ergebnis: Beziehungswert (Integer: -1,0,1) oder *invalid*
Beispiel:

```
var a = "Willkommen im WAP";
var b = String.find(a,"komm"); // liefert b = 4
var c = String.find(a,"come"); // liefert c = -1
var d = String.find(a,1000); // liefert d = ""
var e = String.find(a,""); // liefert invalid
```

toString(*Wert*)

Normalerweise werden in WMLScript übergebene Parameter automatisch als String interpretiert, wenn die aufgerufene Funktion an dieser Stelle eine Zeichenkette erwartet. Eine Ausnahme stellt jedoch der Wert *invalid* dar: hier findet keine automatische Typisierung statt. Um auch diesen Wert als String behandeln zu können, wurde diese Funktion geschaffen.
Parameter:

- *Wert*: ein beliebiger (untypisierter) Wert

Ergebnis: (ausnahmslos) String
Beispiel:

```
var a = String.toString(71);
// liefert a = "71"
var b = String.toString(true);
// liefert b = "true"
var c = String.toString(invalid);
// liefert c = "invalid"
```

format(*Regeln*,*Wert*)

Diese Funktion konvertiert den gegebenen *Wert* in eine Zeichenkette (String), wobei die Ausgabe nach den ebenfalls gelieferten *Regeln* formatiert wird. Die Art der Formatierung ist stark an die `printf`-Formatierung der Sprache C angelehnt. Es können Variablen als Ganzzahlen, Fließkommazahlen und String formatiert werden, repräsentiert durch die Typkennzeichner `d`, `f` und `s`. Auch ohne den Einsatz dieser Argumente präzise zu beschreiben, sollten die zahlreichen, angeführten Beispiele die Verwendung dieser Funktion verdeutlichen.
Parameter:

- *Regeln*: eine Zeichenkette (String)
- *Wert*: ein beliebiger (untypisierter) Wert

Ergebnis: Zeichenkette (String) oder *invalid*
Beispiel:

```
var a =45;
var b = -45;
var c = "now";
var d = 1.2345678;
var e = String.format("e: %6d", a); // e="e:   45"
var f = String.format("%6d", b); // f="   -45"
var g= String.format("%6.4d", a); // g="  0045"
var h= String.format("%6.4d", b); // g=" -0045"
var i= String.format("Buy %s", c); // i="Buy now"
var j= String.format("%3f", d); // j="1.234567"
var k= String.format("%6.2f%%", d); // k="   1.23%"
var l= String.format("%3f %2f.", d);
// l="1.234567 ."
```

```
var m= String.format("%.0d", 0); // m=""
var n= String.format("%7d", "Int"); // n=invalid
var o= String.format("%s", true); // o="true"
```

15.4 Die URL-Bibliothek

Mit den Funktionen dieser Bibliothek können absolute und relative URLs manipuliert werden. Eine *URL* (Uniform Resource Locator) bezeichnet den »Standort« einer im WWW anzufordernden Ressource. Ein einfaches Beispiel für eine (absolute) URL ist:

```
http://www.dpunkt.de/grafik.gif
```

Die allgemeine Syntax einer URL lautet:

```
<schema>://<host>:<port>/<pfad>;<parameter>?
<query-string>#<fragment>
```

isValid(*Url*)

Mit dieser Funktion kann die Syntax einer URL-Zeichenkette geprüft werden, d.h. ob es sich tatsächlich um eine gültige relative oder absolute URL handelt.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: `true` (URL ist gültig), `false` (ungültig) oder *invalid*
Beispiel:

```
var a=URL.isValid(
"http://www.dpunkt.de/bsp/wap#bsp2");
// liefert a = true
var b=URL.isValid("bsp/wap#bsp2");
// liefert b = true
var c=URL.isValid("www.dpunkt.de/bsp:wap");
// liefert c = false
```

getScheme(*Url*)

Diese Funktion liefert das verwendete Protokollschema der (absoluten oder relativen) URL, sofern angegeben. In den meisten

Fällen wird es sich hierbei um *http*-Aufrufe handeln. Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Parameter-String (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getScheme(
"http://www.dpunkt.de/bsp/wap");
// liefert a = "http"
var b=URL.getScheme("www.dpunkt.de/bsp/wap");
// liefert b = ""
```

getHost(*Url*)

Diese Funktion extrahiert den Host-Name aus der absoluten oder relativen URL. Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Host-Name (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getHost("http://www.dpunkt.de/wap");
// liefert a = "www.dpunkt.de"
var b=URL.getHost("wap#frag");
// liefert b = ""
```

getPort(*Url*)

Wird in der URL eine Portnummer explizit angegeben, so kann mittels dieser Funktion die Nummer isoliert werden. Fehlt die Portnummer, ist das Ergebnis die leere Zeichenkette. Ist *Url* ungültig, so ist das Ergebnis *invalid*. Der Standard-Port für HTTP-Anfragen ist beispielsweise 80.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Portnummer (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getPort(
```

```
"http://www.dpunkt.de:8002/wap/bsp");
// liefert a = "8002"
```

getPath(*Url*)

Diese Funktion extrahiert den Dateipfad aus der absoluten oder relativen URL. Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Dateipfad der URL (String) oder *invalid*
Beispiel:

```
var a=URL.getPath(
"http://www.dpunkt.de/wap/bsp#nr3");
// liefert a = "wap/bsp#nr3"
```

getParameters(*Url*)

Diese Funktion liefert die in der absoluten oder relativen URL angegebenen GET-Parameter. Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: URL-Parameter (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getParameters(
"http://www.dpunkt.de/bsp;3;2?x=1&y=4");
// liefert a = "3;2"
var b=URL.getParameters("../bsp;3;2?x=1&y=4");
// liefert b = "3;2"
```

getQuery(*Url*)

Hiermit kann der Parameter-String (query) aus einer absoluten oder relativen URL extrahiert werden. Der Parameter-String enthält alle per GET an den Server übermittelten Parameter sowie deren Werte (durch `&` abgetrennt). Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Parameter-String (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getQuery(
"http://www.dpunkt.de/bsp?x=1&y=4");
// liefert a = "x=1&y=4"
var b=URL.getQuery("http://www.dpunkt.de/bsp/wap");
// liefert b = ""
```

getFragment(*Url*)

Diese Funktion liefert ein in der absoluten oder relativen *Url* angegebenes Fragment `#frag`, sofern vorhanden. Ansonsten wird eine leere Zeichenkette zurückgegeben. Ist *Url* ungültig, so ist das Ergebnis *invalid*.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: Fragment (String), leere Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.getFragment(
"http://www.dpunkt.de/bsp?param=value#2");
// liefert a = "2"
```

getBase()

Diese Funktion liefert die absolute URL der momentanen WMLScript-Ausführungseinheit (=Basis-URL), ohne Berücksichtigung eines eventuell angehängten Fragmentlokators `#frag`.
Ergebnis: Absolute URL as Zeichenkette

Beispiel:
```
var a=URL.getBase();
// liefert "http://www.abcxyz.de/demo.scr"
```

getReferer()

Diese Funktion liefert die relative URL (relativ zur Basis-URL) des Objekts, welches das momentane WMLScript aufgerufen hat. Typischerweise erhält man hier den Dateinamen des aufrufenden WML-Dokuments (siehe auch `getBase`).
Ergebnis: **Beispiel**:
```
var a=URL.getReferer(); // liefert z.B. "menu.wml"
```

resolve(*Url1*,*Url2*)

Diese Funktion erzeugt eine absolute URL, die sich als Verknüpfung der Basis-URL *Url1* und einer eingebetteten URL *Url2* gemäß [RFC2396] zusammensetzt. Ist *Url2* bereits in absoluter Form, so wird diese unverändert zurückgegeben.
Parameter:

- *Url1*: eine Basis-URL (String)
- *Url2*: eine eingebettete URL (String)

Ergebnis: Zeichenkette (String) oder *invalid*
Beispiel:
```
var a = URL.resolve(
"http://www.dpunkt.de","news.wml");
// liefert a = "http://www.dpunkt.de/news.wml"
```

escapeString(*String*)

Mit dieser Funktion werden alle in *String* auftretenden Sonderzeichen durch zweistellige Escape-Sequenzen der Form `%hh` ersetzt. Enthält die Eingabe Sonderzeichen, die nicht Bestandteil des US-ASCII-Zeichensatzes sind, so wird *invalid* zurückgegeben (siehe auch Funktion *unescapeString*).
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: Ergebnis-Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.escapeString(
"http://www.dpunkt.de/bsp?par=val#2");
// liefert a =
"http%3a%2f%2fwww.dpunkt.de%2fbsp%3fpar%3dval%232"
```

unescapeString(*String*)

Mit dieser Funktion werden alle in *String* auftretenden Escape-Sequenzen der Form `%hh` durch ihre entsprechenden Sonderzeichen ersetzt. Zeichenketten ohne Escape-Sequenzen werden nicht auf gültige URL-Syntax geprüft. Enthält die Eingabe Sonderzeichen, die nicht Bestandteil des US-ASCII-Zeichensatzes sind, so wird *invalid* zurückgegeben (siehe auch Funkt. `escapeString`).
Parameter:

- *String*: eine beliebige Zeichenkette (String)

Ergebnis: Ergebnis-Zeichenkette oder *invalid*
Beispiel:

```
var a=URL.escapeString(
"http%3a%2f%2fwww.dpunkt.de%2fbsp%3fpar%3dv%232");
// liefert a = "http://www.dpunkt.de/bsp?par=v#2"
```

loadString(*Url*,*contentType*)

Mittels dieser Funktion können spezielle Inhaltsformate von einer URL geladen und als String in einer Variablen zur weiteren Verarbeitung abgelegt werden. Treten Fehler beim Ladevorgang auf, so wird ein protokollspezifischer Fehlercode zurückgegeben.
Parameter:

- *Url*: eine URL-Zeichenkette (String)
- *contentType*: eine Zeichenkette, die genau einen gültigen, vorhandenen Inhaltstyp beschreibt. Das Präfix muß `text/` sein, der Untertyp ist beliebig.

Ergebnis: Zeichenkette (String), Fehlercode (Integer) oder *invalid*
Beispiel:

```
var meineUrl =
"http://www.dpunkt.de/vcards/meine.vcf";
var meineKarte =
URL.loadString(meineUrl, "text/x-vcard")
```

15.5 Die `WMLBrowser`-Bibliothek

Im WAP sind Skript-Interpreter und WML-Browser zwei gleichgestellte, unabhängige Komponenten der WAE (im WWW sind Skriptsprachen meist in den HTML-Browser eingebettet). Um dennoch den Kontext des WML-Browsers mitverfolgen zu können, werden in dieser Bibliothek einige Funktionen zum Browser-Zugriff zur Verfügung gestellt.

`getVar(Name)`

Liefert den momentanen Wert der im Browser festgehaltenen Variable *Name*, sofern diese existiert, ansonsten eine leere Zeichenkette.
Parameter:

- *Name*: ein Variablenbezeichner (String)

Ergebnis: String oder *invalid*
Beispiel:

```
var a=WMLBrowser.getVar(benutzer);
// liefert a = "mueller", sofern diese Variable
derart belegt ist
```

`setVar(Name,Value)`

Mit dieser Funktion können Variablen im Browserkontext definiert und belegt werden. Eine erfolgreiche Durchführung gibt den Wahrheitswert *true* zurück, ansonsten wird mit *false* geantwortet.
Parameter:

- *Name*: ein Parameter-Bezeichner (String)
- *Value*: den Wert dieses Parameters (String)

Ergebnis: Wahrheitswert oder *invalid*
Beispiel:

```
var a = WMLBrowser.setVar("benutzer", "mayer");
// sollte a=true liefern
```

`go(Url)`

Hiermit wird der Microbrowser veranlasst, eine durch *Url* angegebene Adresse anzusteuern (siehe auch die `<go>`-Aktion in WML,

die Sie in Abschnitt 8.2 auf Seite 70 beschrieben finden). Dies findet jedoch erst statt, nachdem der WMLScript-Interpreter seine Arbeit beendet und die Kontrolle wieder an den Browser zurückgegeben hat. Die Übergabe einer leeren Zeichenkette als URL signalisiert dem Browser, keine Inhalte zu laden.
Vorsicht: Die Funktionen *go()* und *prev()* dieser Bibliothek überschreiben einander. Nur das Ergebnis der zuletzt ausgeführten Funktion bleibt gültig.
Parameter:

- *Url*: eine URL-Zeichenkette (String)

Ergebnis: leere Zeichenkette oder *invalid*
Beispiel:

```
var karte="http://wap.dpunkt.de/index.wml#start";
WMLBrowser.go(karte);
```

prev()

Mit dieser Funktion wird der Microbrowser zu der vorherigen WML-Card zurückgesteuert. Dies findet jedoch erst statt, nachdem der WMLScript-Interpreter seine Arbeit beendet und die Kontrolle wieder an den Browser zurückgegeben hat.
Vorsicht: Die Funktionen *go()* und *prev()* dieser Bibliothek überschreiben einander. Nur das Ergebnis der zuletzt ausgeführten Funktion bleibt gültig.
Ergebnis: **Beispiel**:

```
WMLBrowser.prev();
// geht zurück zur vorherigen Card
```

newContext()

Diese Funktion löscht, analog dem WML-Attribut `newcontext`, den aktuellen Kontext des Microbrowsers. Eine leere Zeichenkette wird zurückgegeben.
Ergebnis: **Beispiel**:

```
WMLBrowser.newContext();
// Kontext des Browsers wurde zurückgesetzt
```

getCurrentCard()

Diese Funktion liefert die kleinste relative URL der WML-Card, welche momentan vom Microbrowser verarbeitet wird. Liegt das Deck dieser Karte außerhalb der Basis-URL der WMLScript-Ausführungseinheit, so wird eine absolute URL zurückgegeben.
Ergebnis: **Beispiel**:

```
var a=WMLBrowser.getCurrentCard();
// liefert z.B. a = "deck#menucard"
```

refresh()

Diese Funktion erzwingt eine Aktualisierung des Kontextes im Browser, analog zu dem WML-Task `refresh`. Eine leere Zeichenkette wird zurückgegeben.
Ergebnis: **Beispiel**:

```
WMLBrowser.setVar("benutzer", "mayer");
WMLBrowser.refresh()
// neue Variable ist nun im Browser "sichtbar".
```

15.6 Die `Dialogs`-Bibliothek

Diese Bibliothek liefert einige nützliche Funktionen zur Realisierung von vorgefertigten Dialogen mit dem Benutzer.

prompt(*Nachricht*,*Vorbelegung*)

Diese Funktion erzeugt einen kurzen Dialog, der die *Nachricht* ausgibt und eine Benutzereingabe erwartet. Durch die Angabe der *Vorbelegung* kann eine Eingabe als Eingabeoption vorgegeben werden. Rückgabewert der Funktion ist die vom Benutzer eingegebene Zeichenkette.
Parameter:

- *Nachricht*: eine (kurze) Mitteilung (String)
- *Vorbelegung*: eine Textvorbelegung (String)

Ergebnis: Zeichenkette (String) oder *invalid*
Beispiel:

```
var a = "gast";
var b=Dialogs.prompt("Login-name: ",b);
```

confirm(*Nachricht,Annahme,Ablehnung*)

Diese Funktion erzeugt einen kurzen Dialog, der die *Nachricht* ausgibt und den Benutzer vor die Alternativen Annahme oder Ablehnung stellt. Wählt der Benutzer die *Annahme*, so wird `true` zurückgegeben, wählt er die *Ablehnung*, so ist `false` die Rückgabe.
Parameter:

- *String*: eine Zeichenkette (String)
- *Annahme*: Text zur Annahme(String)
- *Ablehnung*: Text zur Ablehnung (String)

Ergebnis: Wahrheitswert oder *invalid*
Beispiel:

```
Dialogs.confirm(
"sind Sie sicher ?", "Ja", "Abbruch");
```

alert(*Nachricht*)

Hiermit wird die *Nachricht* als Meldung ausgegeben und eine Bestätigung durch den Benutzer abgewartet. Liefert eine leere Zeichenkette zurück.
Parameter:

- *Nachricht*: eine Mitteilung (String)

Ergebnis: leere Zeichenkette oder *invalid*
Beispiel:

```
WMLBrowser.alert(
"Vorsicht: die gegebene PLZ ist zu kurz");
```

15.7 Die Crypto-Bibliothek

Die seit Version 1.2 in WAP spezifizierte Bibliothek für kryptographische Methoden [53] beinhaltet derzeit nur eine einzige Funktion. Weitere Funktionen, z.B. zur symmetrischen oder asymmetrischen Verschlüsselung, sind in späteren Versionen zu erwarten.

signText(*String,Optionen,KeyIDType,KeyID*)

Einige Anwendungen erfordern eine Authentifikation des Benutzers, die durch eine sogenannte *digitale Unterschrift* erfolgen kann.

Ein Unterzeichner leistet seine Unterschrift, indem er am Ende einer Transaktion einen vorgegebenen Text mit seinem *privaten Schlüssel* codiert. Jedermann kann nun durch Entschlüsseln der Daten mit dem *öffentlichen Schlüssel* des Unterzeichners die Signatur auf Echtheit prüfen.
Die gegebene Funktion fordert den Benutzer auf, die Zeichenkette *String* einer digitalen Signatur zu unterziehen. Nachdem der Benutzer seinen privaten Schlüssel, z.B. durch Freigabe des WIM (WAP Identity Module), dargelegt hat, wird die Zeichenkette chiffriert und als Rückgabewert der Funktion ausgeliefert. Der Benutzer hat selbstverständlich die Möglichkeit, eine Signatur und somit die Authentifizierung abzulehnen.
Parameter:

- *String*: eine zu signierende Zeichenkette (String)
- *Optionen*: hiermit können zusätzliche Angaben im Ausgabewert der Funktion festgelegt werden (bitweise in eine Integerzahl codiert)
- *KeyIDType*: gibt den Typ (beliebig oder an einen Benutzer gebunden oder an ein Zertifikat gebunden) des zu verwendenden Schlüssels vor (Integer)
- *KeyID*: ist der Schlüssel durch den vorhergehenden Parameter an einen Hash gebunden, so wird hier dieser Wert mitgeliefert (String)

Ergebnis: String oder *invalid*. Der Rückgabestring enthält im Normalfall die signierte Eingabe im Format *base-64 encoding*. Lehnt der Benutzer eine Authentifikation ab, so lautet der Ausgabetext *»error:userCancel«*; ist kein passender Schlüssel oder gültiges Zertifikat vorhanden, so wird *»error:noCert«*ausgegeben.
Beispiel:

```
var hash="\x37\xA2\x98\xB4\x03\xC2\x5E\x29\x60\xA3
\x3F\x29\x96\x98\x44\xB0\x91\x08\xB8\x70";
var b = Crypto.signText("Signier mich !",0,1,hash);
// liefert (falls der Benutzer mitmacht) den mit
// einem zu hash passenden Schlüssel
// signierten Text.
```

15.8 Proprietäre Bibliotheken

Neben den im WAP standardisierten Kernbibliotheken gibt es noch die Möglichkeit, auf proprietäre Funktionspakete zurück-

zugreifen. Hierbei muß jedoch die vorliegende WMLScript-Implementierung von der Existenz einer solchen Bibliothek wissen.

Ein Beispiel für eine derartige, herstellerspezifische Bibliothek ist die `Debug Library` von Nokia. Diese ist in den von Nokia angebotenen WAP-Entwicklungsumgebungen integriert und liefert erweiterte Funktionen zum Debuggen selbsterstellter WMLScript-Programme. Mittels der gelieferten Funktionen kann eine sogenannte *Debug-Datei* erstellt und mit Informationen über den Programmablauf (z.B. Variablenwerte) gefüttert werden. Die Bibliothek umfasst drei Funktionen:

`openFile`: öffnet eine Debug-Datei zum Lesen, Schreiben oder Erweitern

`closeFile`: schließt die Debug-Datei

`printLn`: schreibt eine angegebene Zeichenkette in die Debug-Datei. Ein Beispiel:

```
Debug.printLn("Schleife X betreten");
```

Teil III

WAP

16 WAP in der Praxis

Mit dem Wissen der vorangegangenen Kapitel haben Sie nun alles Handwerkszeug beisammen, um Ihr eigenes WAP-Angebot auf die Beine zu stellen. Nach den zahlreichen Beispielen des Buches und einigen Testprojekten erwacht vielleicht in Ihnen das Interesse, einen echten WAP-Dienst aufzubauen. Dieses Kapitel soll Ihnen dabei behilflich sein, ein brauchbares und somit erfolgreiches Angebot zu entwerfen und umzusetzen.

16.1 Voraussetzungen an WAP-Angebote

Bitte kommen Sie nicht in die Versuchung, eine »WAP-Version« Ihrer Web-Homepage zu erstellen – es sei denn, Sie bieten auf Ihrer Site puren Nutzwert an. Dieser ist nämlich bei WAP-Angeboten deutlich stärker gefordert als im bunten und unterhaltsamen Web. Bevor Sie also irgendetwas »ins WAP stellen«, d.h. Informationen und Dienste im Internet für WAP-fähige Endgeräte verfügbar machen, sollten Sie sich einige Gedanken über Ihr Angebot und Ihre Zielgruppe machen.

Seien Sie sich bei der Ideenfindung stets bewußt: »surfen« im WAP macht keinen Spaß. Die Benutzung eines WAP-Endgerätes ist relativ teuer, wenig attraktiv und meist sehr umständlich im Vergleich zum heimischen PC. Der Benutzer wird also Ihr Angebot besuchen, weil er Information benötigt und keinesfalls, weil er auf der Suche nach Zerstreuung und Unterhaltung ist oder gar genau Ihre WAP-Site besonders »schön gestylt« findet. Er wird eine einfache und schnelle Navigation durch Ihr Angebot zu schätzen wissen und jede Effekthascherei als kosten- und zeitfressenden Ballast verurteilen. Bieten Sie ihm daher eine möglichst optimale Benutzerschnittstelle. Die hohe Kunst der Implementierung in WML und WMLScript liegt also darin, Ihre Informationen und Dienste in einer kompakten und konzentrierten Darstellung aufzubereiten und dennoch vollständig auszuliefern. Stellen Sie häufig angeforderte Informationen nach »vorne«, verzichten Sie auf Überflüssi-

ges ohne Nutzwert und minimieren Sie Navigationspfade durch kritisches Testen und Optimieren der Menüstruktur.

WAP-Angebote können als mehr oder weniger WAP-geeignet klassifiziert werden – je nachdem, ob sie einen brauchbaren Nutzen unter den technischen Gegebenheiten darstellen. Als *WAP-geeignet* können gelten:

- Angebote, die an die Restriktionen der WAP-Technologie (siehe Seite 197) angepasst werden können,
- Angebote mit geringer Interaktivität,
- Angebote mit geringen Benutzereingaben,
- Angebote mit Anspruch auf Mobilität.

Auf der anderen Seite gelten als *WAP-ungeeignet*:

- repräsentative Angebote,
- multimediale Angebote,
- Angebote, die keinen Mehrwert durch mobile Verfügbarkeit erhalten.

16.2 WAP-Anwendungen

Je nachdem, welche Zielgruppe mit einem Angebot bedient wird und welche Absicht hinter dem Angebot steht, kann man WAP-Anwendungen in Dienstklassen unterteilen. Hierbei ist zu beachten, daß die Grenzen zwischen den verschiedenen Typen nicht immer klar gezogen werden können, d.h. eine spezielle Anwendung kann auch durchaus in mehreren Klassen einen berechtigten Platz finden.

16.2.1 Klassifizierung von WAP-Anwendungen

Jeder WAP-Dienst läßt sich in mindestens eine der folgenden Klassen einordnen:

- *Informationsdienste* liefern allgemein nachgefragte Informationen in den verschiedensten Themengebieten. Beispiele für solche Informationen sind Sportergebnisse, Nachrichten, Lottozahlen, Wetter, etc.

- *Auskunftsdienste* sind individuell angeforderte Informationen wie z.B. Telefonnummern oder Fahrplanauskunft.
- *Finanzanwendungen* bilden insbesondere im WAP eine eigene Klasse, da sie eine der häufigsten Anwendungen darstellen. Im Gegensatz zu anderen Diensten sind die Finanzdienste fast immer kostenpflichtig. Beispiele sind Börsenauskunfts- und -ordersysteme, Bezahlsysteme und Kontoführung.
- *m-Commerce* (das *m* steht für *mobile*) umfasst alle verkaufenden und verkaufsfördernden Dienste, meist sind es beschränkte Ergänzungsdienste zu e-Commerce-Systemen im Web.
- *Mobilitätsdienste* werden von Anwendungen erbracht, die insbesondere die individuelle Mobilität fördern und unterstützen. Eine wichtige Beispielanwendung ist die Routenplanung.
- *Dienste für WAP* sollen die WAP-Benutzung selbst bereichern. Beispiele hierfür sind Bookmark-Dienste, E-mail-Konten, Passwort-Safe (hierbei handelt es sich um einen nicht unumstrittenen Dienst, der Ihnen die immer zahlreicher werdenden Passwörter und PINs archiviert), WAP-Portale und WAP-Suchmaschinen.
- *Private Anwendungen* bieten Dienste, die nur einer geschlossenen Benutzergruppe zur Verfügung stehen. Hierbei handelt es sich typischerweise um Anwendungen im Intranet oder im *Virtual Private Network (VPN)*. Die Möglichkeit, die Lagerverfügbarkeit von Rohstoffen in Produktionsbetrieben abzurufen, sei hier als Beispiel stellvertretend für zahlreiche Anwendungsmöglichkeiten genannt.
- *Sonstige Dienste* sind alle Anwendungen, die in keine der oberen Klassen direkt eingeordnet werden können, wie z.B. Behörden-Infos, Paketverfolgung, Fernsteuerungen, Unterhaltung oder Kleinanzeigen.

Die Anzahl angebotener WAP-Seiten wächst täglich an. Neben viel Unbrauchbarem gibt es einige Angebote, bei denen sich eine nähere Betrachtung lohnt. Die folgenden Unterabschnitte nehmen für einige Anwendungskategorien jeweils ein existentes, besonders gelungenes und/oder diskussionswürdiges WAP-Angebot

genauer unter die Lupe. Alle vorgestellten Dienste werden kostenlos angeboten.

16.2.2 Informationsdienste

Die Tagesschau der Ersten Programme ist eines der beliebtesten WAP-Angebote im Netz. Unter der URL *wap.tagesschau.de* lassen sich ständig aktuelle Schlagzeilen (ca. 10 bis 15) abrufen. Jede Schlagzeile ist anwählbar und mit einem Kurzartikel hinterlegt.

Abbildung 16.1
wap.tagesschau.de

Pluspunkte dieses Dienstes sind seine Nützlichkeit, Schlichtheit und Klarheit. Die sehr flache Navigation mit nur 2 Ebenen macht es fast unmöglich, sich in dem Angebot zu verlieren. Ohne unnötigen Ballast erhält der Besucher die gewünschten Informationen. Der Browser erhält die Übersichtsseite aller News als Deck mit einer Card, jeweils zwei aufeinanderfolgende Kurzartikel sind als zwei Cards in je einem weiteren Deck untergebracht. Dieser WAP-Dienst kann Ihnen als Vorbild dafür dienen, wie Sie den Benutzer vor unangenehmen Tastatureingaben bestmöglich schützen können: bieten Sie Informationen pur!

16.2.3 Auskunftsdienste

Im Web sehr beliebt ist der elektronische Auskunftsdienst der Deutschen Bahn. Unter *wap.hafas.de* hat das Unternehmen einen Teil seines Dienstes auch im WAP verfügbar gemacht. Die Zielgruppe hierfür sind Reisende, die sich unterwegs über mögliche Verkehrsverbindungen informieren möchten oder müssen. Will der Benutzer eine Zugverbindung erfragen, so braucht er lediglich vier Angaben zu machen: Ausgangsort, Zielort, Datum und Zeit.

Bei den beiden letzten Angaben ist das System besonders komfortabel ausgelegt, da Datum und Zeit mit der aktuellen Stunde vorbelegt sind. Die Ausgabe liefert die ab dem gewählten Zeitpunkt nächsten drei Verbindungen unter Angabe der Abfahrts- und Ankunftszeit sowie der Anzahl, wie oft der Fahrgast umsteigen muß. Zu jeder Verbindung können nun noch weitere Informationen abgefragt werden: Alle Stationen der Reise, Gesamtdauer und Informationen zu den Zügen zwischen den Stationen wie Zugnummer und Abfahrtsgleis. Sogar die Information über das Vorhandensein eines Bordrestaurants wird geliefert.

Abbildung 16.2
wap.hafas.de

Die WAP-Umsetzung ist eine sinnvolle Reduzierung des WWW-Angebotes. Reisewillige werden im heimischen Umfeld stets den PC einem WAP-Gerät vorziehen, also muß das Angebot insbesondere Personen ansprechen, die sich bereits auf ihrer Reise befinden. Bei diesem Angebot kommt der Benutzer schnell und gezielt zu den gesuchten Informationen.

Die Brauchbarkeit des Angebotes steht außer Frage, da beispielsweise schon innerhalb eines verspäteten Zuges alternative Verkehrsverbindungen ermittelt werden können, ohne daß man sich auf die Suche nach einem Zugbegleiter machen muß.

Als Besonderheit kann der Benutzer aus vier Kategorien von WAP-Endgeräten und Simulatoren seine gewünschte Darstellung auswählen. Auch eine Sprachauswahl zwischen Deutsch und Englisch steht dem Reisenden zur Verfügung. Der hohe Nutzwert des Dienstes fordert die Nachahmung in Ihrem WAP-Angebot heraus.

16.2.4 Finanzdienste

Auch in Deutschland werden Aktien als Geldanlage immer beliebter. Wer auch unterwegs seine Börseninfos, Aktienkurse und De-

potwerte im Blick behalten möchte, der findet unter *wap.ariva.de* einen kostenlosen und dennoch qualitativ hochwertigen Finanzdienst. Hier kann der Anleger sich laufend über den Wert seines Depots, Kurswerte einzelner Aktien (nach Name oder Wertpapierkennummer auswählbar) oder Listen (DAX, NEMAX-Allshare, ISDEX, RTS, Branchenindizes) informieren. Als Bonbon gibt es sogar noch grafische Charts von Kursentwicklungen über wählbare Zeiträume. Aktuelle Kurzmeldungen aus der Börsenwelt sowie eine Kurzaufstellung der Tops und Flops am Neuen Markt frischen das Angebot auf.

Abbildung 16.3
wap.ariva.de

Der Dienst von Ariva ist übersichtlich, aktuell und zielgruppenorientiert. Die Abfrage des eigenen Depots benutzt eine personalisierte Komponente, die ein gelungenes Zusammenspiel zwischen Web und WAP demonstriert. Während der Benutzer sich auf der Website des Anbieters bequem sein individuelles Wertpapierdepot »zusammenklicken« kann, wird er im WAP lediglich einige aktuelle Informationen über dieses Depot abrufen wollen. Der WAP-Zugang zum Depot ist durch eine Benutzerkennung mit Passwort geschützt. Etwas unausgereift erscheint die Ausgabe der grafischen Charts, da diese zum Testzeitpunkt mit nahezu keinem WAP-Simulator darstellbar waren. Dies, obwohl der Benutzer hier zwischen zwei Darstellungsgrößen eine gewisse Möglichkeit zur Browseranpassung erhält. Der Verzicht auf diese Anzeige fällt jedoch leicht, da *unterwegs* eine strategische Analyse ohnehin kaum stattfinden wird. Die Ausführung von Kauf- oder Verkaufsorder bleiben übrigens ausschließlich *gebührenpflichtigen* WAP-Diensten vorbehalten, die meist von Anlagebanken gestellt werden.

Neben der Nützlichkeit des Dienstes sollte der WAP-Entwickler bei diesem Angebot insbesondere das Depot als interessante

Umsetzung im Auge behalten. Für viele WAP-Dienste sollte gelten, daß diese als mobile Ergänzung zu Web-Diensten fungieren. Belassen Sie umfangreiche Formulareingaben im Web. Mehr noch: Bieten Sie dem WAP-Nutzer einen *personalisierten* Dienst, dessen individuelle Komponenten er sich bequem im WWW zusammenstellen und zügig im WAP abrufen kann!

16.2.5 m-Commerce

Auf den ersten Blick etwas kurios erscheinen Online-Shops im WAP. Ein relativ ausgereiftes Beispiel für diese Spezies finden Sie bei *wap.booxtra.de*, dem Shop eines Online-Händlers für Bücher. Das WAP-Angebot bietet nahezu alle Funktionen eines kompletten Online-Shops: eine Suchmöglichkeit (nach Autor, Titel oder nach der ISBN), einen Warenkorb, Bestellfunktionen sowie zahlreiche Buchtipps und Informationen zu Büchern verschiedener Kategorien.

Abbildung 16.4
wap.booxtra.de

Dieses Angebot muß etwas differenzierter betrachtet werden: Auf der einen Seite ist die WML-Umsetzung des Shops gut gelungen, auf der anderen Seite wurde an den Interessen der vermeintlichen Zielgruppe vorbeiprogrammiert. Das Angebot ignoriert, daß ein wesentlicher Faktor des eCommerce das *Einkaufserlebnis* ausmacht. Zahlreiche Punkte im Hauptmenü zielen auf ein »Stöbern« in der Angebotsvielfalt des Kataloges ab, worauf wohl kaum ein Shop-Besucher per WAP Lust haben wird. Dies ist ganz klar eine Leidenschaft des im heimischen Wohnzimmer befindlichen Web-Surfers. Dennoch stimmt die grobe Richtung des Angebotes von Booxtra, es müssen jedoch die speziellen Bedürfnisse des m-Commerce berücksichtigt werden.

Bieten Sie dem Besucher die Möglichkeit, schnell die *Lagerverfügbarkeit* (z.B. ob ein bestimmter Artikel schon eingetroffen ist) zu überprüfen. Geben Sie ihm die Möglichkeit, unterwegs einen *Preisvergleich* in Ihrem Angebot durchzuführen, insbesondere wenn Sie Artikel anbieten, die häufigen Preisschwankungen ausgesetzt sind (Abruf von Tagespreisen). Hat er bestellt, so will er vielleicht per WAP den *Bestellstatus* oder den *Transportstatus* abfragen. Verschonen Sie ihn mit Warenkörben und langen Produktinformationen – der WAP-Nutzer wird das Produkt seines Interesses bereits kennen! Bei der Bestellung selbst können Sie sich dann auch wieder ein Scheibchen von Booxtra abschneiden: Bestellt wird durch Authentifikation registrierter Benutzer (die Registrierung findet im Web statt) oder per Telefon – natürlich WAP-typisch durch einen WTA-Aufruf realisiert.

16.2.6 Mobilitätsdienste

Zu den Mobilitätsdiensten zählen Angebote, die insbesondere dem beweglichen Teilnehmer in seiner mobilen Situation hilfreich sind. Auch das oben genannte Angebot der Bahn hätte in dieser Kategorie plaziert werden können, als Paradebeispiel zur Unterstützung individueller Mobilität soll hier jedoch das Angebot auf *wap.ptv.de* dienen. Das von der PTV AG eingerichtete WAP-Portal bietet neben aktuellen Verkehrsmeldungen und einem Hotelführer einen leistungsfähigen Routenplaner für Deutschland oder Europa. Die bundesweite Routensuche kann entweder besonders WAP-freundlich durch bloße Eingabe der Postleitzahlen des Start- und Zielortes oder punktgenau – mit etwas mehr Eingabeaufwand für Ort, Teilort und Straße – durchgeführt werden. Die sich meist über mehrere Decks ausbreitende Ausgabe liefert Streckenlänge, Gesamtfahrtzeit und eine Etappenliste mit Entfernungs- und Richtungsangaben.

Der Dienst bietet hohe Zuverlässigkeit und Genauigkeit der ermittelten Routen und zählt zu den nützlichsten und beliebtesten Angeboten im WAP. Natürlich erhält man nicht annähernd den Komfort moderner Routenplaner-Software des heimischen PCs, brauchbare Ergebnisse bekommt der Benutzer jedoch auch hier. Vor allem die allgegenwärtige Verfügbarkeit des Dienstes macht einen zusätzlichen Wert aus. Eine Personalisierung, z.B. durch Eingabe fest eingestellter Wegepunkte (z.B. die Wohnungsadresse des Benutzers), könnte hier noch eine weitere Steigerung hervorrufen.

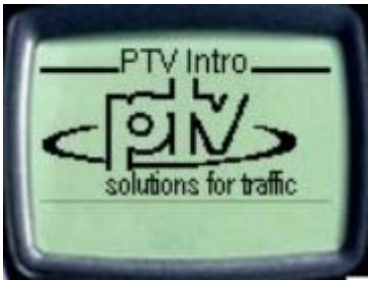


Abbildung 16.5
wap.ptv.de

An dem Dienst besonders interessant: Das Angebot kann von den wachsenden Bandbreiten der Funknetze profitieren und seinen Komfort durch Übermittlung geographischer Darstellungen weiter ausbauen. Bei zukünftigen, breitbandigen Netzen stellt dann auch eine Übertragung grafischer Karteninformationen keine technische Hürde mehr dar.

Das folgende Kapitel bringt Ihnen die Entwicklung und einige technische Grundprinzipien von Funknetzen näher und wagt einen Blick in die Übertragungstechnologien der nächsten Generation.

17 Mobilfunknetze – die WAP-Grundlage

Sie erhalten in diesem Kapitel einen Einblick in die faszinierende Welt der drahtlosen, mobilen Kommunikation von gestern, heute und morgen. Bitte erwarten Sie an dieser Stelle keine umfassende Einführung in die Technologien der Mobilkommunikation, lediglich ein grober Gesamtüberblick über diese Materie soll Ihnen das Umfeld von WAP etwas näherbringen. Der Schwerpunkt des Kapitels liegt daher in den Netzen, die für WAP relevant sind. Insbesondere wird das in Europa vorherrschende und Ihnen wahrscheinlich von der Abkürzung her geläufige GSM dargestellt. Einen sehr umfassenden, technisch tiefgreifenden Einblick in das Thema Funknetze liefert z.B. [27]. Für einen überschaubaren und dennoch sehr vollständigen Einblick in die Mobilkommunikation mit dem Schwerpunkt »mobiles Internet« sei dem Leser [19] wärmstens empfohlen.

17.1 Mobilfunknetze, -systeme und Zugriffsverfahren

Mobilfunknetze werden seit Beginn der 80er Jahre in den Industrienationen eingesetzt. Zu Beginn ein Privileg der wohlhabenden Bevölkerung, hat sich die mobile Telefonie in den 90er Jahren zu einem bezahlbaren Konsumgut für alle Bevölkerungsschichten entwickelt. Nicht zuletzt die gewaltigen Erlöse von knapp 100 Mrd. DM der UMTS-Lizenzversteigerung in Deutschland deuten das Potential dieses Marktes an. Eine schnelle und weite Verbreitung von Mobiltelefonen in den USA hat diesen Kontinent einst zur führenden Nation in diesem Gebiet gemacht. Da ein großer Markt jedoch in der Regel träger reagiert als ein kleinerer, hinken die teils veralteten Netzsysteme in den USA heute den technologisch ausgereifteren Systemen in den europäischen Ländern hinterher. Es ist jedoch nicht nur das in Europa geläufige GSM-

Mobiltelefon als Anwendungsplattform für WAP möglich, die Spezifikation des Protokolls sieht in [42] zahlreiche Geräte und Mobilfunknetze für den Einsatz von WAP vor.

Dieses Kapitel stellt Ihnen die wichtigsten Typen von Netzen und Technologien vor. Hierbei besteht die Absicht, interessante Themen nicht durch komplizierte, technische Betrachtungen ihrer Attraktivität zu berauben und auch stets ein Auge auf das zentrale Thema WAP zu behalten. Der folgende Abschnitt wird eine Ordnung in die Vielfalt der Funknetze bringen und hierbei grundlegende, physikalische Verfahren vorstellen.

17.1.1 Merkmale von Funknetzen

Funknetze zur (Daten-)Kommunikation können durch verschiedene Merkmale charakterisiert werden. Sie können klassifiziert werden nach ...

- ... *der Ausbreitung*: Hier unterscheidet man nach *lokalen*, *zellulären* oder *orbitalen* Netzen.

  Bei *lokalen Netzen* handelt es sich um Funknetze von geringer Ausdehnung (unter 1000 Metern), beispielsweise das System für Schnurlostelefone DECT (Digital Enhanced Cordless Telecommunications) oder auch drahlose LANs. Diese Systeme arbeiten meist in einem »freien«, d.h. nicht reservierten Frequenzbereich um die 2,4 GHz. Die WAP-Spezifikation sieht auch für DECT eine Möglichkeit der Implementierung vor.

  Das heute gebräuchliche Mobiltelefon arbeitet in der Regel in *zellulären Landfunknetzen*. Hierbei wird der zur Verfügung stehende Raum in Zellen unterteilt und jede Funkzelle mit einer Antenne, der sogenannten *Basisstation* ausgestattet. In Abbildung 17.1 ist eine derartige Aufteilung dargestellt, in der Praxis variiert die Zellengröße jedoch abhängig von der Bevölkerungsdichte. In Ballungsräumen kann die Kapazität einer Zelle schon bei einer Reichweite von wenigen hundert Metern an ihre Grenzen stossen, während in ländlichen Gebieten mit wenig Nutzlast ein Zellenradius von bis zu 50 km möglich ist. Der Benutzer baut nun innerhalb seiner Funkzelle eine Verbindung zu dieser Basisstation auf und kann sich innerhalb der Zelle frei bewegen. Moderne, heutige Systeme gestatten auch einen Übergang in eine benachbarte Zelle, *Handover* genannt, ohne die Verbindung abzubrechen

und somit eine netzweite Bewegungsfreiheit. Da zellenbasierte Landfunknetze mit Abstand die höchsten Benutzerzahlen vorweisen, zielt auch WAP hauptsächlich auf mobile Endgeräte dieser Systeme ab.

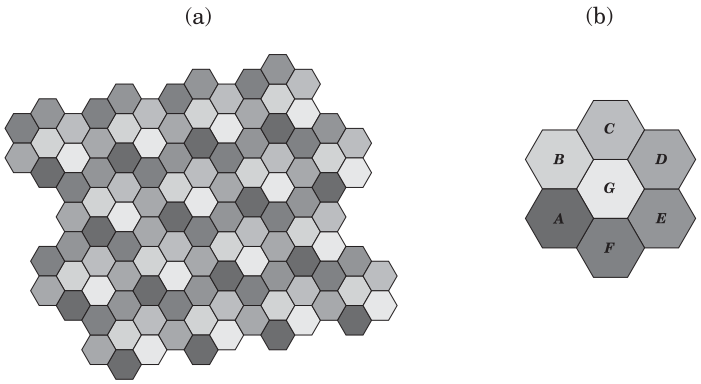


Abbildung 17.1 *Anordnung der Funkzellen bzw. Cluster*

Die *orbitalen* Netze werden durch Einsatz von Satelliten ermöglicht und spielen für WAP keine Rolle. Haupteinsatzgebiet von Satelliten sind neben Ortung und Überwachung die sogenannten *Broadcast*-Dienste (z.B. Fernsehen), die sich nicht an einen einzigen Empfänger wenden. Telefonie aus dem All ist aufgrund der enormen Kosten kaum markttauglich, wie 1999 das Scheitern des Milliardenprojektes *Iridium* (ein Projekt mit 72 LEO-Satelliten und einem Volumen von ca. 14 Mrd. DM) eindrucksvoll bewies.

- ... *der Dauer*: Funknetze können permanent existieren oder ad hoc entstehen. Ein Beispiel für ein spontan aufgebautes Netz ist die direkte Funkverbindung zwischen zwei sich im Luftraum begegnenden Flugzeugen. Auch die *Bluetooth*-Technologie basiert auf spontanen Netzen mit sehr geringer Ausdehnung (unter 10 Metern). Alle Netze mit fest installierten Antennen und auf vorbestimmten Bahnen kreisende Satelliten zählen zu den permanten Funknetzen. Nur diese spielen in der heutigen Zeit für WAP eine Rolle, da das WAP-Gateway einen Fixpunkt im Kommunikationsnetz darstellt.

- ... *der Berechtigung*: Da Frequenzbereiche ein knappes Gut darstellen, wird die Vergabe von Frequenzen meist durch staatliche Stellen reguliert und bestimmte Frequenzbereiche für verschiedene Netzbetreiber reserviert. Diese Frequenzen

dürfen dann exklusiv von diesem Betreiber belegt werden. Auch bei den Rundfunkfrequenzen für Fernsehen und Radio findet dieses Verfahren Anwendung. Auf der anderen Seite werden bestimmte Freqenzbereiche, meist jenseits der 2-GHz-Grenze, zur allgemeinen Benutzung freigegeben. Lediglich die Sendeleistung der Geräte wird durch Vorschriften beschränkt, damit eine faire und störungsfreie Belegung der Ressource durch viele Benutzer ermöglicht wird.

- ... *dem Modulationstyp*: Man unterscheidet beim Modulationstyp zwischen analogen und digitalen Funknetzen. Bei analogen Systemen werden die Kommunikationsfrequenzen direkt auf das Trägersignal aufmoduliert. Die neueren, digitalen Systeme müssen ihre binär vorliegenden Informationen vor dem Versand in analoge Signale umwandeln. Der Experte kann hier weiter nach der *Modulationsart* fragen, also nach dem Verfahren zur Umwandlung der digitalen Signale. Ohne weiter auf die Techniken einzugehen, sollen hier die *Amplitudenumtastung*, *Frequenzumtastung* und die *Phasenumtastung* genannt werden. Fest steht in jedem Fall, daß digitale Funknetze den analogen in nahezu jeder Hinsicht überlegen sind und heutige Installationen ausschließlich digitaler Natur sind.

- ... *dem Kanalzugriffsverfahren*: Einzelne Frequenzbänder sind eine kostbare Ressource, die möglichst effizient auf die Teilnehmer verteilt werden muß. Um dieses Ziel zu erreichen, kann ein Frequenzband mehreren Benutzern zugewiesen werden. Diese müssen jedoch auf irgendeine Art auseinandergehalten werden, um gegenseitige Störungen auszuschließen. Ein derartiges Verfahren der Mehrfachnutzung nennt man *Multiplex*. Den verschiedenen Multiplexverfahren ist der folgende Unterabschnitt gewidmet.

17.1.2 Kanalzugriffsverfahren

Das einfachste Multiplexverfahren ist das *Raummultiplex*. Dies bedeutet, daß eine Frequenz von mehreren Benutzern gleichzeitig belegt werden kann, wenn deren Reichweiten weit genug auseinanderliegen. Ein Radiosender in Hamburg kann beispielsweise auf der gleichen Frequenz senden wie ein Münchner Radiosender; beide Sender betreiben Raummultiplex. Werfen Sie nun einen erneuten Blick auf die rechte Seite (b) der Abbildung 17.1. Einzelne

Zellen (hier 7) werden zu sogenannten *Clustern* zusammengefasst, innerhalb derer jede Zelle eine eigene Frequenz besitzt. Durch periodische Anordnung der Cluster wie in Bildteil (a) können gleiche Frequenzbänder nun immer wieder vergeben werden, da niemals zwei Zellen gleicher Frequenz direkt nebeneinander liegen. Kanalzugriff mit Raummultiplex wird auch als *SDMA-Verfahren* (Space Division Multiple Access) bezeichnet.

Mit *Frequenzmultiplex* bezeichnet man alle Verfahren, die das Frequenzspektrum in disjunkte Teilbänder aufteilt. Ein Frequenzband wird also beispielsweise für zwei Benutzer in zwei kleinere Bänder weiter unterteilt. Ein anschauliches Beispiel ist die mehrfache, gleichzeitige Benutzung einer Straße durch nebeneinanderliegende Fahrbahnen. Das Verfahren ist im Mobilfunk unter der Abkürzung *FDMA* (Frequency Division Multiple Access) geläufig.

Eine Ressource kann auch durch das sogenannte *Zeitmultiplex* mehreren Benutzern zur Verfügung gestellt werden. Zeitmultiplex ermöglicht es beispielsweise Betrieben, nur einen Kopierer für mehrere Mitarbeiter einer Abteilung zu betreiben. Für Mobilfunknetze bedeutet dieses sogenannte *TDMA*-Verfahren (Time Division Multiple Access), daß die gesamte Zeit der Kommunikation in mehrere Abschnitte, sogenannte *Zeitrahmen* aufgeteilt wird. In jedem Rahmen stehen nun eine bestimmte Anzahl *Zeitschlitze* (Time Slots) zur Verfügung, die durch unterschiedliche Benutzer belegt werden können. Die Länge eines Zeitrahmens bewegt sich im Millisekundenbereich, so daß bei einem Telefonat keine Unterbrechungen spürbar werden und auch Datenübertragungen als kontinuierlicher Strom auftreten. Bei TDMA unterscheidet man selbst wieder statische und mehrere dynamische Verfahren, die mit wachsender Komplexität die Übertragung noch effizienter gestalten, indem die Gefahr von Kollisionen durch Mehrfachbelegung von Slots minimiert wird.

Ein ausschließlich auf digitale Netze beschränktes Verfahren zum Mehrfachzugriff ist das *Codemultiplex*, in der Fachwelt als *CDMA* (Code Division Multiple Access) bekannt. In diesem System können alle Benutzer das gleiche Frequenzband belegen, jedoch wird das *Nutzsignal* für jeden Benutzer unterschiedlich codiert. Ein gerne zitiertes Beispiel ist das Zimmer gefüllt mit Personen unterschiedlicher Nationalitäten und Sprachen. Sprechen alle Personen durcheinander und jeder in seiner Muttersprache, so wird ein Zuhörer nur die Stimme in seiner eigenen Sprache aufnehmen und verstehen. Die Stimmen aller anderen Personen werden als unverständliches Hintergrundgeräusch empfunden. Beim

Kanalzugriff im Mobilfunk verwenden die Teilnehmer eine spezielle Form der Codierung für die einzelnen Verkehrskanäle, die eine eindeutige Unterscheidung zu allen anderen Teilnehmern hervorruft.

Das CDMA-Verfahren gilt als das effizientere Verfahren im Vergleich zu TDMA. In der Praxis kommen immer mehrere Multiplexverfahren zum Einsatz, GSM verwendet z.B. SDMA durch Clusterbildung sowie FDMA und TDMA beim Medienzugriff.

17.2 Mobilfunk in der ersten Generation

Mobile Telefonie gibt es in Deutschland schon seit 1958. In diesem Jahr wurde das frequenzmodulierte *A-Netz* in Betrieb genommen. In diesem Netz wurden die Verbindungen noch von Hand vermittelt und die Besitzer der Mobiltelefone konnten zwar Gepräche initiieren, nicht jedoch angerufen werden. Immerhin erreichte schon dieses System in Deutschland eine bundesweite Netzabdeckung von 80% und über 10.000 Benutzer. Der Betrieb wurde 1977 eingestellt.

Schon 5 Jahre vor Ende des A-Netzes, also 1972, ging das *B-Netz* als Nachfolgesystem an den Start. Dieses Netz war in der Lage, die Vermittlung im Selbstwählverkehr durchzuführen. Auch war es in diesem Netz möglich, mobile Benutzer anzurufen. Da dieses System aber keinen automatischen Zellenübergang (Handover) unterstützte, mußte dem Anrufer der Aufenthaltsort seines Gesprächspartners bekannt sein. 1985 gab es in Deutschland über 25.000 Kunden und die 158 Basisstationen erzielten eine vollständige Netzabdeckung. Stillgelegt wurde das Netz im Jahre 1994.

1985 folgte das dritte und letzte analoge System in Deutschland: das *C-Netz*. Das analoge Netz arbeitet mit digitaler Signalisierung bei leitungsvermittelnden Wählverbindungen und unterstützt erstmalig eine, wenn auch sehr einfache Form der Datenübertragung mit 2,4 kbit/s mittels spezieller V.24/V.28-Modems. Während seine beiden Vorgänger noch im 160-MHz-Bereich operierten, nutzt das zellenbasierte C-Netz nun einen Träger von 450 MHz. Durch eine stetige Reduzierung der Kosten und des Gewichtes der Endgeräte konnte das System den Massenmarkt erobern: zu seiner besten Zeit (1995) besaß das flächendeckende C-Netz eine Anzahl von etwa 850.000 Kunden, die Zahlen sind jedoch zugunsten digitaler GSM-Netze stark rückläufig. Die Abschaltung

des heute noch in Betrieb befindlichen Netzes ist beschlossene Sache und soll noch vor dem Jahr 2001 vollstreckt werden.

In anderen europäischen Ländern werden oder wurden ebenfalls analoge Systeme ähnlich dem C-Netz, welches übrigens außerhalb Deutschlands nur in Portugal verwendet wird, eingesetzt. Erwähnt werden sollen das in Skandinavien entwickelte *Nordic Mobile Telephone (NMT)*-System, das in zahlreichen europäischen Ländern vorherrschte, sowie das *TACS-System* in Großbritannien und Italien. In den USA startete 1983 das *Advanced Mobile Phone System (AMPS)*. Das zelluläre System arbeitet im 850-MHz-Band und wurde nicht nur in den USA zu einem durchschlagenden Erfolg. Da AMPS auch heute noch ein in 90 Ländern weltweit genutzter Standard ist, wurde dieses Netz als einziges *analoges* System in der WAP-Spezifikation berücksichtigt. Alle anderen WAP-Träger basieren auf digitalen Netzen, welche die zweite Generation mobiler Kommunikation darstellen.

17.3 Die zweite Generation des Mobilfunks

Die Gegenwart und nahe Zukunft ist von den Mobilfunknetzen der zweiten Generation geprägt. Auch wenn heute noch, insbesondere in Amerika, analoge Netze im Einsatz sind – die Tage dieser Systeme sind gezählt. Abgesehen von der höheren Komplexität bieten digitale Systeme in jeder Hinsicht klare Vorteile: Qualität und Kapazität der Übertragung ist höher und eine Vielfalt neuer Dienste, z.B. Kurzmitteilungen, werden unterstützt. Digitale Funknetze im TDMA- und CDMA-Verfahren spielen heute eine dominierende Rolle auf dem riesigen Markt der Mobilkommunikation. In der WAP-Spezifikation wurden alle wichtigen gegenwärtigen Digitalnetze als WAP-Träger aufgenommen. Neben klassischen mobilen Telefonnetzen finden sich hier auch Netze für Mitteilungsdienste (Pager), Systeme für Schnurlostelefone und Bündelfunknetze. Bei Bündelfunk handelt es sich um betriebliche Netze, in denen eine geschlossene Benutzergruppe, z.B. Taxifahrer, einen einzigen Kanal nutzt. Diese Netze sind lokal beschränkt und dienen meist nur der Sprachkommunikation.

Folgende Aufstellung zeigt, für welche digitalen Netze WAP bereits spezifiziert wurde. Fast immer sind sogar mehrere Dienste eines Mobilfunknetzes als WAP-Träger definiert. Dies muß jedoch nicht bedeuten, daß für jedes System und für jeden Dienst bereits eine Implementierung existiert.

- Global System for Mobile Communications (GSM)

- Der digitale AMPS-Nachfolger ANSI-136 (IS-136)
- Das moderne CDMA-System IS-95
- Personal Digital Cellular (PDC)
- Integrated Digital Enhanced Network iDEN
- Die Pager-Netze FLEX und REFLEX
- Personal Handyphone System PHS
- Das Funkdatennetz DataTAC
- Terrestrial Trunked Radio (TETRA) (Bündelfunk)
- Digital Enhanced Cordless Telecommunications (DECT)
- Mobitex-Netze

Von tragender Bedeutung und mit bereits vorhandener WAP-Infrastruktur ausgestattet sind jedoch nur die drei global marktführenden Telefonsysteme GSM (Europa), IS-136 (USA) und PDC (Japan). Auf alle oben gelisteten Systeme im einzelnen einzugehen würde der Absicht dieses Buches nicht gerecht werden. Daher sollen in diesem Abschnitt nur die zwei in Europa vorherrschenden Systeme und ihr Bezug zu WAP etwas genauer betrachtet werden: GSM und DECT.

17.3.1 GSM

Das *Global System for Mobile Communications (GSM)* ist ein europaweiter, digitaler, sicherer und überdies weitestgehend ISDN-kompatibler Mobilfunkstandard. 1992 ging in Deutschland das D1-Netz als erste GSM-Installation in Betrieb, D2 folgte ein Jahr später und 1995 belegte das E-Netz das neu hinzugekommene 1800-MHz-Band (hier wird GSM auch als *DCS 1800* bezeichnet). Da bei der Entwicklung von GSM – im Gegensatz zu beispielsweise IS-136 (USA) – keine Rücksicht auf eine Kompatibilität zu exisitierenden, analogen Netzen genommen werden musste, entstand ein sehr modernes, effizientes und zukunftsträchtiges Landfunknetz. GSM arbeitet im 900- und 1800-MHz-Band und moduliert seine Bits mit einer speziellen Form der Frequenzumtastung, dem *Gaussian Minimum Shift Keying (GMSK)* auf das Trägersignal. Beim Medienzugriff kommen TDMA (8 Zeitschlitze zu je

0,566 ms pro Zeitrahmen) und FDMA zum Einsatz. Der Netzaufbau besteht neben den Antenneninstallationen (Base Transceiver System, BTS) aus einem komplexen, hochleistungsfähigen und hochverfügbaren Subsystem zur Vermittlung, Benutzerverwaltung und Diagnose. Das GSM-System bietet dem Benutzer eine Fülle von heutigen und zukünftigen Diensten, von denen fünf als WAP-Träger in die WDP-Beschreibung aufgenommen wurden:

1. *Leitungsvermittelte Datenübertragung (Circuit Switched Data, CSD)* ist die zum heutigen Zeitpunkt einzige Möglichkeit für GSM-Endgeräte das WAP zu nutzen. Hierbei agiert das Endgerät ähnlich wie ein Modem: das Gerät baut eine Leitungsverbindung zu einem WAP-Zugangspunkt (WAP-Provider) auf und nutzt diese Punkt-zu-Punkt-Wählverbindung als Kommunikationskanal. Nicht nur die sehr magere Transferrate von 9,6 kbit/s, auch die relativ hohen Verzögerungen beim Verbindungsaufbau beeinträchtigen hier den Komfort. Ein Wechsel des Trägerdienstes in naher Zukunft ist also Grundvoraussetzung für einen endgültigen Durchbruch von WAP bei den Benutzern.

   Ein zukünftiger, leitungsvermittelter Datendienst von GSM, der sogenannte *High Speed Circuit Switched Data (HSCSD) Service* erreicht mittels einer Bündelung mehrerer Verkehrskanälen durch Mehrfachbelegung von Zeitschlitzen immerhin eine Datenrate von 76,8 kbit/s, wird aber in der WDP-Spezifikation (noch) nicht berücksichtigt.

2. Der *Kurzmitteilungsdienst (Short Message Service, abgekürzt als SMS)* dient der alphanumerischen Punkt-zu-Punkt-Nachrichtenübertragung zwischen einer Kurzmitteilungszentrale und einem mobilen Endgerät [6]. SMS wird sowohl zum Bezug allgemeiner Informationen, z.B. Wetterlage, als auch zum individuellen Nachrichtenaustausch zwischen Endgeräten eingesetzt. Die maximale Länge einer Textnachricht beträgt 160 Zeichen pro Kurzmitteilung. Da nur Signalisierungskanäle genutzt werden, ist eine simultane Nutzung der Basisdienste (Telefonieren und SMS) möglich.

   SMS arbeitet im Halbduplexbetrieb und basiert nicht auf einem Sitzungsbetrieb. Um hieraus den für WAP erforderlichen, sitzungsorientierten Trägerdienst im Vollduplexbetrieb bereitzustellen, ist ein erheblicher Zeitaufwand notwendig. Diese zeit- und somit auch kostenintensive Abbildung machen SMS zu einem nur wenig geeigneten WAP-

Träger und lassen andere Dienste wie beispielsweise USSD über GSM, den Sie gleich kennenlernen, geeigneter erscheinen.

3. Der *Zellenmitteilungsdienst (SMS Cell Broadcast, SMSCD)* stellt einen Punkt-zu-Mehrfachpunkt-Nachrichtendienst dar [7]. Das Dienstezentrum sendet mehreren Mobilfunkteilnehmern einheitliche Mitteilungen. Einzelne Empfänger können selektiert werden, die Länge der Mitteilungen ist auf 93 Zeichen begrenzt. In der WAP-Praxis wird dieser Dienst kaum eine Rolle spielen, da insbesondere bei WAP eine Punkt-zu-Punkt-Kommunikation gefordert ist und die Übertragungsleistung noch geringer als bei SMS ausfällt.

4. *Unstructured Supplementary Services Data (USSD)* ist ein Nachrichtendienst zur gegenseitigen Übermittlung von Textzeilen, die meist als Steuerbefehle an das Endgerät oder an den Diensteanbieter interpretiert werden. Der Dienst ist im GSM-Standard in mehreren Entwicklungsstufen definiert: In der ersten USSD-Stufe [8] wird die Interaktion vom Mobiltelefon initiiert, ab der zweiten Stufe ([9] und [10]) können USSD-basierte Transaktionen auch von *Anwendungen* eingeleitet werden. USSD ist ein Dienst ähnlich dem SMS, da hier ebenfalls Textnachrichten (allerdings bis zu einer Länge von bis zu 182 Zeichen) über die Signalisierungskanäle gesendet werden. Im Gegensatz zu SMS werden die Nachrichten nicht gepuffert (*store and forward*), da der Dienst *sitzungsorientiert* arbeitet. Eine errichtete USSD-Sitzung bleibt permanent, bis diese vom Nutzer, von der Anwendung oder durch ein Timeout geschlossen wird.

   Zeichenketten in USSD werden stets mit dem Zeichen `*` eingeleitet und enden mit einem `#`. Beispielsweise liefert der String `*#06#` (danach muß die »Hörer-Abnehmen-Taste« zum Senden gedrückt werden) die eindeutige Geräte-Identifizierungsnummer (IMEI) des Mobiltelefones. Sie können diese Zeichenfolge direkt an Ihrem GSM-Mobiltelefon ausprobieren. Ein weiteres Beispiel ist `**21*NUMMER#`, womit alle eingehenden Gespräche an die mit `NUMMER` angegebene Rufnummer umgeleitet werden.

   USSD über GSM ist ein Halbduplex-Dienst, der keine Zieladressen übermittelt. Da das WAP im Vollduplex-Betrieb arbeitet, muß dieses USSD-Manko transparent gemacht werden. Hierfür ist in [35] das *USSD Dialogue Control Proto-*

col (UDCP) spezifiziert. Das UDCP siedelt sich also im mobilen Endgerät und im GSM-Netzknoten zwischen dem WDP- und dem USSD-Protokoll an. Es adaptiert Zieladressen, versteckt die Halbduplex-Komplexität und erledigt das Beibehalten bzw. Wiederherstellen eines Dialoges.

Einige Punkte sprechen für USSD als geeigneten Träger für das WAP: USSD kann von allen GSM-Telefonen ohne Änderung an der Hardware genutzt werden. Da der Dienst sitzungsorientiert arbeitet, ist die Antwortzeit geringer als bei SMS und ermöglicht so bis zu siebenmal schnellere 2-Wege-Transaktionen als der Mitteilungsdienst. Seit der 2. Stufe von USSD bieten die Dienstbefehle überdies zahlreiche Möglichkeiten der Interaktion mit dem Nutzer. Da die USSD-Befehle über das sogenannte *Home Location Register (HLR)*, eine bei GSM zentralisierte Datenbank mit semipermanenten und temporären Teilnehmerdaten, geleitet werden, funktioniert der Dienst auch beim Roaming.

5. Der tatsächliche Durchbruch für WAP wird vermutlich mit der Umstellung auf *paketvermittelte Datenübertragung (General Packet Radio Service, GPRS)* als Trägerdienst eingeläutet. Hierbei handelt es sich um einen *paketorientierten*, drahtlosen Transportdienst in GSM-Netzen. Hauptmerkmale von GPRS sind [27]:

   - Eine hohe Geschwindigkeit mit (theoretisch) maximal 177,2 kbit/s ist bei gleichzeitiger Belegung aller 8 Zeitschlitze möglich. In der Praxis liegt die Datenrate je nach Anzahl der Benutzer in einer Funkzelle jedoch nur zwischen 10 kbit/s und 57 kbit/s.
   - Der Dienst kann unverzüglich in Anspruch genommen werden, es muß keine Wählverbindung aufgebaut werden.
   - Da Restriktionen von SMS und CSD entfallen, werden neue und bessere Anwendungen möglich.
   - Es sind GPRS-taugliche mobile Endgeräte zur Nutzung dieses Dienstes erforderlich (heutige Geräte und Netzkomponenten beherrschen GPRS meist nicht).
   - Da die Funkkanäle bei einer GPRS-Sitzung nicht permanent in Anspruch genommen werden, kann ein einzelnes Frequenzband von einer hohen Anzahl GPRS-Nutzern gleichzeitig belegt werden. Allerdings sinkt je-

doch auch die individuelle Bandbreite mit der tatsächlichen Anzahl gleichzeitig kommunizierender Teilnehmer innerhalb einer Funkzelle.

Die paketorientierte Datenübertragung macht GPRS prädestiniert für TCP/IP-Anwendungen wie E-mail, Web-Browsen, Informationsdienste, Remote LAN Access, Chat und viele weitere Applikationen. Mit einem paketvermittelten Dienst ist es dann auch möglich, die Abrechnung der WAP-Nutzung nicht mehr nach Verbindungszeit, sondern nach tatsächlich übertragener Datenmenge durchzuführen. WAP-Geräte sind dann quasi »ständig online« – eine wichtige Grundlage für interessante Push-Dienste.

Größtes Problem bei GPRS ist das erforderliche *hohe Investitionsvolumen*, da mit den mobilen Endgeräten auch die Komponenten der Netzbetreiber GPRS-fähig gemacht werden müssen. Mit einer Etablierung am Markt von GPRS wird daher nicht vor Anfang 2002 gerechnet [gprsW3], erste GPRS-taugliche Geräte sollen ab dem Jahr 2000 verfügbar werden. Auch existiert noch einige Unsicherheit bei den Entwicklern, da sich der GPRS-Standard immer noch laufender Änderungen unterziehen muß. Eine Erweiterung zu *Enhanced GPRS* ist derzeit in Bearbeitung, beeinflußt durch *Enhanced Data Rates for GSM Evolution (EDGE)*, eine Erweiterung von GSM in Richtung UMTS.

17.3.2 DECT

Das zweite Trägernetz für WAP, welches in diesem Kapitel etwas genauer betrachtet wird, ist *Digital Enhanced Cordless Telecommunications (DECT)*. Im Gegensatz zu den terrestrischen Zellularnetzen spricht man hier auch von *schnurlosen* Netzen. Der Standard definiert ein zellulares, auf Funkübertragung basierendes System für die Inhouse-Kommunikation mit tragbaren Sprach- und Datenendgeräten (10 bis max. 250 mWatt Sendeleistung) für Distanzen bis zu maximal 200 Metern. Auch drahtlose LANs mit einer Transferrate von ca. 750 kbit/s lassen sich mittels DECT realisieren.

Für die Geräte stehen 10 Trägerkanäle in einem Frequenzbereich von 1880 - 1900 MHz zur Verfügung, von denen jeder Kanal durch TDMA-Zugriff mit zweimal 12 Slots (für die Duplexverbindung) insgesamt 12 Nutzkanäle liefert. Moduliert wird, wie bei

GSM auch, mit dem GMSK-Verfahren. Die Ressourcen können für kombinierte Sprach- und Datenkommunikation aufgeteilt werden. Die Protokolle wurden allerdings auf Sprachkommunikation optimiert und werden daher den Anforderungen der Datenkommunikation nicht voll gerecht. Dennoch werden bei DECT gleich drei Dienste im WDP als WAP-Plattform dargestellt:

- Verbindungsorientierter DECT-Datendienst
- Paketvermittelte Datenübertragung
- Kurzmitteilungsdienst (SMS)

WAP über DECT kann in lokalen Funknetzen durchaus eine wichtige Rolle spielen, da die technischen Gegebenheiten das Netz als »WAP-freundliches« Trägermedium auszeichnen. Der Einsatz von kommunikationsarmen Intranet-Anwendungen in Betrieben wird auch dort möglich sein, wo die Arbeiter typischerweise nicht mit einem PC oder Terminal ausgestattet sind. Ein Arbeiter an einer Kfz-Produktionsstraße könnte z.B. über ein DECT-Telefon per WAP den Lagervorrat abrufen und ggf. Nachbestellungen über den Zentralrechner initiieren. Die Lager-Verfügbarkeit von Produkten ist auch im Einzelhandel eine wichtige Information, die ein Verkäufer ohne Gang zu einem Terminal oder Rückfrage sofort per WAP-Telefon abfragen könnte. Das DECT-WAP-Telefon kann allgemein als Mini-Terminal fungieren und Zugang zum Firmenrechner eröffnen. Ein weiteres Plus für DECT ist die Flexibilität aufgrund der Lokalität: während beispielsweise die Umstellung des GSM-Netzes auf GPRS Unsummen verschlingt, kann ein DECT-System einfach nach den modernsten Erkenntnissen hergestellt und vertrieben werden. Vorhandene DECT-Installationen sind davon nicht betroffen und es entstehen keine nennenswerten Kosten bei der Umstellung.

Die aktuelle Generation der Mobilfunknetze und -dienste zeigt einen klaren Trend zu datenorientierten, digitalen Netzen auf. Fort vom klassischen Telefon und hin zum universellen, mobilen Datenendgerät wird seit einigen Jahren an einem länderübergreifenden Standard für ein hochleistungsfähiges, universelles Mobilfunknetz gearbeitet: UMTS.

17.4 Die dritte Generation: UMTS

Die momentane Entwicklung der Mobilfunknetze fokussiert ein länderübergreifendes Konzept, welches zellulare, schnurlose und

orbitale Funksysteme in ein einheitliches System namens *IMT-2000* (IMT steht für *International Mobile Telecommunications*) münden läßt. Weltweit sollen in Zukunft für dieses System die Frequenzbänder 1885 – 2025 MHz und 2110 – 2200 MHz reserviert werden. Eine globale Harmonisierung für dieses Frequenzspektrum dürfte jedoch aufgrund der zahlreichen, unterschiedlichen Systeme und Regulierungsbestrebungen der einzelnen Staaten eine Wunschvorstellung bleiben. Dennoch hat man es in Asien und Europa geschafft, zumindest annähernd übereinstimmende Bänder für IMT-2000 zu reservieren,

Der Standard des *European Telecommunications Standardisation Institute (ETSI)*, dem Normierungsinstitut für den gesamteuropäischen Telekommunikationsmarkt, als Vorschlag für dieses System ist das *Universal Mobile Telecommunications System, UMTS*. In Deutschland wurden von der *Regulierungsbehörde für Telekommunikation und Post* die Frequenzbänder 1920 – 1980 MHz und 2010 – 2170 MHz für UMTS reserviert und im Sommer 2000 für insgesamt knapp 100 Milliarden DM päckchenweise an fünf Mobilfunkanbieter versteigert. UMTS stellt aus europäischer Sicht eine Erweiterung von GSM dar und integriert zahlreiche vorhandene Netze und Dienste, wie z.B. DECT und *Personal Communications Network (PCN)* – ein in Großbritannien vorgeschlagenes Konzept als zukünftiges System für individuelle Kommunikation. Die Übertragungsrate ist vom Urbanisierungsfaktor und von der maximalen Fortbewegungsgeschwindigkeit abhängig und übersteigt mit 144 kbit/s auf dem Land und 2 Mbit/s in Ballungszentren sogar noch die meisten privaten Kommunikationsanschlüsse im heutigen Festnetz. Die hohe Bandbreite ermöglicht ein breit gefächertes Dienstangebot:

- *Leitungsvermittelte Trägerdienste* für Sprache, Audiodaten und transparente Benutzerdaten.
- *Paketvermittelte Trägerdienste* bei denen mehrere Transportdienste wie z.B. Breitband-ISDN-Dienste einen verbindungslosen Kanal zum Datentransfer stellen.
- *Teledienste* wie z.B. Telefonie, Paging, Telefax, Audio- und Videoübertragung, Mobilitätsdienste (Navigation oder Ortung), Kurznachrichtendienste und viele mehr.
- *Zusatzdienste* (Rufnummernidentifikation, Gruppenkommunikation, Abrechnungsinformationen etc.).

- *Mehrwertdienste* wie z.B. ein personalisiertes Dienstangebot oder kostenpflichtige Bandbreitenreservierung.

Die Unterstützung von *Dienstgüte-Parametern* wie beispielsweise eine maximale Verzögerung oder eine maximale Bitfehlerrate bringen den Komfort heutiger Festnetze in die Funknetze von morgen ein. Insbesondere dem mobilen Internetzugriff – und somit auch WAP – kommt eine Unterstützung von *asymmetrischen Datendiensten* zugute. Hierbei wird berücksichtigt, daß bei üblichem »Surf-Verhalten« weit mehr Daten zum Benutzer transportiert werden als von diesem abgehen.

Der Weg zu UMTS ist jedoch kein Schnitt mit der Vergangenheit. Vielmehr wird das vorhandene Mobilfunknetz sukzessive um immer weitere Technologien und Dienste ausgebaut, um schließlich alle Anforderungen des UMTS-Standards zu erfüllen. Ein wichtiger Vorstoß für GSM in diese Richtung ist die Einführung des oben beschriebenen paketvermittelten Trägerdienstes GPRS. Den nächsten Schritt in Richtung UMTS-Standard stellt die GSM-Erweiterung EDGE dar. Hiermit kann eine Datenrate von immerhin 384 kbit/s unter Beibehaltung der GSM-Frequenzen erzielt werden. Die Migration kann schrittweise vollzogen werden und somit wird der teuren, vorhandenen Infrastruktur des GSM-Netzes eine Gnadenfrist eingeräumt und der Wechsel relativ kostengünstig durchgeführt. In grob geschätzt 5 bis 10 Jahren wird dann unsere Kommunikationslandschaft von mobilen Kleinstgeräten geprägt sein, deren Anwendungen dank der hohen Bandbreite von UMTS-Netzen vielfältigste, multimediale Daten durch den Äther sausen lassen.

Bis dahin müssen sich die Anwender jedoch noch einige Jahre mit den eingeschränkten Leistungsmerkmalen heutiger Funknetze begnügen. Für den heute schon akuten Bedarf an mobilem Internetzugriff durch diese langsamen Netze wurde das auf diese Einschränkungen zurechtgeschnittene WAP geschaffen. Im nächsten Kapitel wird Ihnen dieses schon mehrfach erwähnte Protokoll nun von Grund auf vorgestellt.

18 Die WAP-Architektur

18.1 Was ist WAP?

Das *Wireless Application Protocol*, kurz WAP, hat sich zum Ziel gesetzt, das weltumspannende Internet um die Benutzer von kleinen, mobilen, drahtlosen Endgeräten zu erweitern. Diesen Teilnehmern soll ein ortsunabhängiger Zugriff auf Ressourcen des Internets ermöglicht werden. Hierzu wurde ein Standard ins Leben gerufen, der auf die Bedürfnisse der mobilen Benutzer zurechtgeschnitten wurde. Als Terminals bzw. Endgeräte fungieren vorwiegend Mobiltelefone, die mit einer speziellen Software, einem sogenannten *WAP-Client*, ausgestattet sind. Bedient werden diese Geräte durch gewöhnliche Webserver.

Warum benötigt man überhaupt ein »zweites Web«? Weshalb greift man mit den Endgeräten nicht direkt auf die Informationen des vorhandenen Internets zurück? Die Antwort liegt in den *Einschränkungen*, denen heutige Mobilfunknetze und -geräte unterliegen. Mobile Endgeräte sind den geläufigen Web-Terminals in zahlreichen Punkten unterlegen:

- Tragbare Endgeräte besitzen eine *eingeschränkte Benutzerschnittstelle*. Die Geräte sind durch kleine, niedrigauflösende und meist monochrome Displays nur für die Darstellung geringer Informationsmengen geeignet. Auch die Benutzereingaben durch mehrfachbelegte Telefontastaturen oder Stifteingabe sind nur in geringem Unfang zumutbar.

- Die kompakten Geräte verfügen nur über *geringe Rechenleistung und Speicherkapazität*. Leistungsfähige Hardware findet keinen Platz in den immer kleiner werdenden Begleitern.

- Eine *begrenzte Stromversorgung* über Akkus oder Batterien erfordert einen eminent sparsamen Umgang mit der verfügbaren Energie und schränkt so die Benutzung dieser Geräte erheblich ein.

Auch die Luftschnittstelle hinkt als Übertragungsmedium den klassischen Leitungen hinterher:

- Die heutigen Mobilfunknetze besitzen eine vielfach *geringere Bandbreite* als z.B. Verbindungen über das ISDN-Telefonnetz oder sogar Glasfaser.
- Verbindungen über Funknetze weisen *hohe Verzögerungen* auf.
- Öffentliche Funknetze bieten eine wesentlich *geringere Verfügbarkeit und Stabilität*. Es muß häufiger mit Verbindungsunterbrechungen gerechnet werden.

Im klassischen Internet bzw. im World Wide Web existieren Inhaltsformate wie z.B. Applets, Flash-Animationen, hochauflösende Bilder oder Bildanimationen, bei denen das Display der mobilen Endgeräte hoffnungslos überfordert wäre. Auch sind nahezu alle Webseiten auf eine Bedienung per Maus ausgelegt. Überdies besitzen die Formate und Protokolle (z.B. HTML oder HTTP) einen hohen Grad an Redundanz (eliminierbaren Informationen), die eine nicht wirklich benötigte Bandbreite in Anspruch nimmt. Aufgrund dieser Faktoren bildet das klassische Web für die heutigen mobilen Endgeräte eine unüberwindbare Hürde. Es musste also ein Standard geschaffen werden, der den Besonderheiten der tragbaren Terminals Rechnung trägt: das WAP.

Alle im WAP definierten Standards sind auf die oben genannten Restriktionen im Mobilfunk optimiert. Es werden neue, für die Geräte darstellbare Inhaltsformate definiert. Alle Protokolldaten und Inhaltsformate werden vor ihrer Übertragung durch die Luftschnittstelle in irgendeiner Form komprimiert, so daß kein überflüssiges Byte übermittelt werden muß. Den häufig auftretenden Verbindungsabbrüchen durch z.B. Funklücken oder Austausch der Stromversorgung tritt ein (im HTTP-Protokoll unbekanntes) Zustandsmodell entgegen. Das WAP stellt jedoch keine umfassende Neuentwicklung dar: so gut es geht werden vorhandene Internet-Standards berücksichtigt und verwendet. Insgesamt kann WAP daher als ein hochspezialisiertes Internet betrachtet werden, das die erschwerten Bedingungen der Mobilität bestmöglich kompensiert.

18.2 Das WAP-Forum

Im Dezember 1995 wurde die Firma *Unwired Planet* (heute unter dem Namen *phone.com* bekannt) in der Absicht gegründet, einen mobilen Zugang zum Internet zu entwickeln [18]. Ein gutes halbes Jahr später formierte sich dieses Unternehmen gemeinsam mit Nokia, Ericsson und Motorola zu dem sogenannten *WAP-Forum* und veröffentlichte im September 1997 die erste Version von WAP. In den darauffolgenden drei Jahren traten über einhundert weitere Firmen und Organisationen dem Forum bei und es folgten drei weitere Versionen (V1.1 bis V1.3) der WAP-Spezifikation.

Obwohl die Bezeichnung »Wireless Application Protocol« die Existenz *eines einzigen* Protokolls suggeriert, so umfaßt WAP ein ganzes Paket von Protokollen und Spezifikationen. Die Gesamtheit dieser Dokumente wird als *WAP Specification Suite* bezeichnet.

Das WAP-Forum verabschiedet alle 6 Monate eine neue Version der Specification Suite. Da WAP eine offene Spezifikation ist, sind alle Dokumente der aktuellen und vergangenen WAP-Versionen öffentlich zugänglich. Die Dokumente liegen im PDF-Format auf den Webseiten des WAP-Forums (www.wapforum.org) für jedermann zum Herunterladen bereit. Der Umfang der gesamten Spezifikation wächst mit jeder neuen Version und liegt bei WAP V1.2 bereits jenseits der 1.300 Seiten-Marke (genaugenommen 1.323 Textseiten).

Die verschiedenen Versionen der WAP-Spezifikationen sind mit Ausnahme der allerersten Version abwärtskompatibel ausgerichtet. Von einer Kompatibilität zur Version 1.0 hat man zugunsten einer verbesserten Architektur und einer Kompatibilität zu XHTML in den Nachfolgeversionen Abstand genommen.

18.3 Die WAP-Spezifikationen

Im WAP-Forum entwickeln verschiedene Arbeitsgruppen (Working Groups) an zukünftigen Standards in mehreren Bereichen (z.B. Übertragung, Anwendungen oder Sicherheit) des WAP. Ein veröffentlichtes Dokument einer Arbeitsgruppe wird je nach Status einer von drei möglichen Klassen von Spezifikationen zugeordnet:

1. *Verabschiedete (approved) Spezifikationen*: Dokumente mit diesem Status – und ausschließlich diese - bilden die offizielle WAP-Spezifikation. Derartige Standards wurden sorgfältig

geprüft und gelten als stabile und optimale Lösung einer wohldefinierten und begründeten Aufgabenstellung.

2. *Vorgeschlagene (proposed) Spezifikationen*: Hier stehen technische Dokumente, die von den Mitgliedern des Forums vor der Aufnahme in die WAP-Spezifikation noch weiter validiert und geprüft werden müssen. Der Standard gilt in diesem Status schon als vollständig und stabil. Vorgeschlagene Spezifikationen verlassen diese Klasse meist innerhalb von drei Monaten.

3. *Prototyp-Spezifikationen*: In dieser Klasse stehen Kandidaten für die Aufnahme in die WAP-Spezifikation. Die definierten Standards gelten als theoretisch ausgereift, müssen sich jedoch noch der öffentlichen Diskussion bzw. Tests an Prototypen unterziehen, bevor sie eine Verabschiedung vom Forum erhalten oder verworfen werden. Die Aufenthaltsdauer in diesem Status liegt üblicherweise bei sechs Monaten.

Die Inhalte zahlreicher Dokumente der WAP Specification Suite sowie der WAP-Protokollstapel wird Thema der nächsten beiden Kapitel sein. Der Rest dieses Kapitels soll Ihnen die grundlegende Funktionsweise von WAP deutlich machen und Ihnen die Elemente einer WAP-Infrastruktur vorstellen. Die Betrachtung der von einem Server initiierten Kommunikation (Push) wird das Kapitel abschließen.

18.4 Die Kommunikation im WAP

WAP-Objekte, z.B. WML-Dokumente, können auf existierenden Webservern abgelegt werden. Es müssen außer dem Eintrag der neuen Inhaltstytpen (MIME-Typen, siehe auch Seite 251) keine Änderungen am Server vorgenommen werden. Fordert ein mobiler Client nun ein derartiges Objekt an, so wird dieses mittels der üblichen Transportmechanismen des Internets bis zu einem *WAP Proxy*, auch *WAP-Gateway* genannt, befördert. Dieses komprimiert das Objekt (sofern es nicht schon in komprimierter Form vom Server abgeliefert wird) in ein schlankes, binäres Format und sendet es über das Funknetz an den aufrufenden Teilnehmer weiter.

WML-Dokumente werden beispielsweise vor der Übertragung im Mobilfunknetz in das sogenannte *Binary WML*-Format überführt. Das Verfahren aus [29] beschreibt eine kompakte, binäre Darstellung *beliebiger* XML-Dokumente, die auf einen Bruchteil

ihrer Ausgangsgröße komprimiert werden. Die Codierung arbeitet hauptsächlich mit Substitution bekannter Tag-Bezeichner und Strukturelemente durch knappe Hexadezimalwerte, sogenannte *Token*. Da WML selbst auch eine XML-Anwendung darstellt, wurde dieses Verfahren in die WAP-Spezifikation integriert, es kann jedoch auch in vielen anderen Fällen von bandbreitenbeschränkter XML-Übertragung mit fest definierten DTDs eingesetzt werden.

Abbildung 18.1
Codierung nach Binary XML

```
Ein kleines XML-Dokument:
-------------------------
<?xml version="1.0" ?>
<!DOCTYPE XYZ [
  <!ELEMENT XYZ (CARD)+>
  <!ELEMENT CARD (#PCDATA | BR)*>
  <!ELEMENT BR EMPTY>
  <!ENTITY nbsp " ">
]>
<XYZ>
  <CARD>
  X &amp; Y<BR/>
  X = 1
  </CARD>
</XYZ>

codiert nach Binary-XML:
------------------------
02  01  03  00  47  46  03  ' ' 'X' ' ' '&' ' '
'Y' 00  05  03  ' ' 'X' 00  02  81  20  03  '='
00  02  81  20  03  '1' ' ' 00  01  01
```

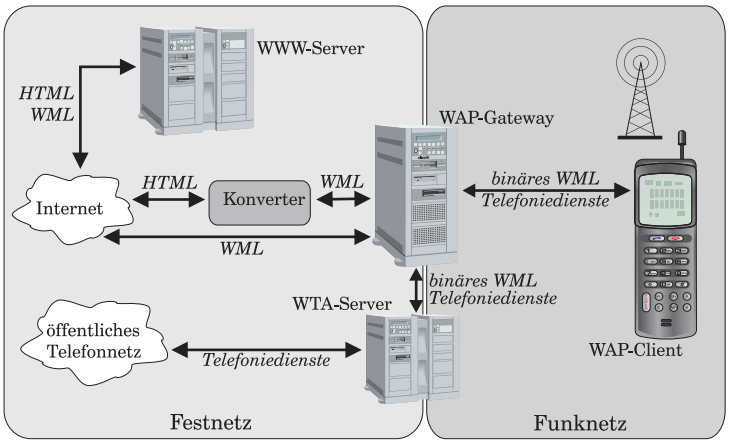
Eine kleine Demonstration (aus [29]) einer XML-Codierung in das Binary-Format zeigt Abbildung 18.1. Das Beispiel soll an dieser Stelle nicht die Funktionsweise der Transformation erläutern, sondern Ihnen lediglich einen Eindruck vermitteln, wie effizient das Verfahren arbeitet. Weitere Maßnahmen im WAP zur Reduzierung der Übertragungsmenge sind optimierte Protokolle, Zwischenspeicherung (Caching), schlanke Bildformate und Bytecode-Übersetzung von Scriptprogrammen.

Um den WAP-Geräten auch HTML-Dokumente zugänglich zu machen, existieren einige Ansätze für eine automatische Übersetzung von HTML nach WML. Ein derartiger Filter kann direkt

in einem auf WML spezialisierten Webserver oder - wie meist realisiert - an den WAP-Proxy gekoppelt sein. Bis heute gibt es jedoch keine Implementierung, die zufriedenstellende Ergebnisse liefert. Ob jemals ein brauchbarer Algorithmus gefunden wird ist zweifelhaft, da bei der WML-Erzeugung zu viele Einbußen gemacht werden müssen und eine vernünftige Reduktion der HTML-Dokumente sehr stark an den Kontext der Seite gebunden ist.

Das Szenario in Abbildung 18.2 verdeutlicht die möglichen Kommunikationspfade zwischen Content-Server und mobilem Client.

Abbildung 18.2
WAP-Kommunikation



Neben der Übertragung typischer WAP-Objekte aus dem Internet können WAP-Geräte per Binary WML auch auf die Dienste des Telefonnetzes zugreifen und somit die Internet-Anwendungen mit Telefonfunktionen vermischen. Beispielsweise können somit auch Verweise (Links) auf Telefonnummern mit direkter Anwahlmöglichkeit der Nummer realisiert werden. Grundlage dieser Integration bildet die im WAP spezifizierte Telefonschnittstelle (*Wireless Telephony Application Interface, kurz WTAI).* Bei der Spezifikation der WTA-Funktionen mußten jedoch Kompromisse bezüglich der Portabilität von WAP gemacht werden: hier entsagt die WTA-Spezifikation dem Paradigma der totalen Plattformunabhängigkeit, da unterschiedliche Mobilfunknetze verschiedene Telefoniedienste anbieten. Neben dem »kleinsten Nenner« wurden zusätzliche, funknetzspezifische Dokumente ([47], [48] und [49]), die sogenannten *Addendums*, in die Specification Suite aufgenommen.

18.5 Das WAP-Gateway

Für sämtliche Anwendungen und Inhaltsobjekte ist in WAP eine eigens entwickelte Anwendungsumgebung, die *Wireless Application Environment (WAE)*, definiert. Innerhalb dieser Umgebung wird stets vom Vorhandensein eines WAP-Gateways zwischen Funknetz und Internet ausgegangen. Ein WAP-Gateway verfolgt drei Hauptziele:

1. Übersetzung zwischen den WAP-Protokollen und den Internetprotokollen (HTTP).
2. Gateway zwischen Fest- und Funknetz mit Adressenabbildung (z.B. wird eine IP-Adresse auf eine eindeutige Rufnummer (MSISDN) abgebildet).
3. Optimierung der Inhaltsobjekte (z.B. werden WMLScript-Programme in Bytecode übersetzt).

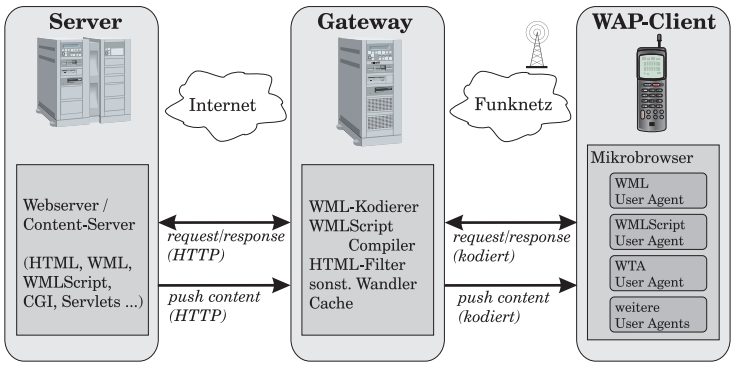


Abbildung 18.3
Das WAP-Gateway

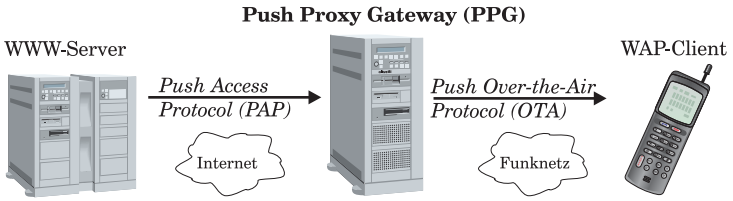
Das Gateway erledigt im Auftrag des WAP-Endgerätes die Kommunikation mit dem Server und liefert ausschließlich binär codierte Objekte an die WAE aus. Hierbei muß das Gateway ggf. Codierungen von WML nach Binary WML vornehmen, WMLScript in Bytecode kompilieren oder Bildformate umwandeln. Darüberhinaus kann dieser auch einen Filter besitzen, um aus Inhalten von HTML-Dokumenten WML-Dokumente zu erstellen (eine direkte Transformation ist *nicht* möglich, da HTML-Dokumente ungleich mächtigere Darstellungsmöglichkeiten besitzen). Auch kann das Gateway zusätzlich mit einem Cache-Speicher zur Zwischenspeicherung von Inhalten ausgestattet sein. Da umgekehrt

das mobile Endgerät seine Anfragen in Binary WML stellt, muß das Gateway dessen Nachrichten wieder decodieren und weiter verarbeiten. Insgesamt zeigt sich das Gateway also dafür verantwortlich, daß die Kommunikation über die Funkschnittstelle in einer optimierten Form stattfindet. Abbildung 18.3 auf der vorherigen Seite verdeutlicht die Stellung des Gateways bei der Übertragung von WAP-Objekten.

18.6 Push-Dienste

Unter einem *Push-Dienst* versteht man die Übermittlung einer Nachricht vom Server an den Client, ohne daß hierfür eine explizite Anforderung des Clients vorausging. Im Gegensatz zum klassischen *Pull-Dienst*, bei dem der Server stets auf eine Client-Anfrage *reagiert*, wird bei einem Push die Übertragung vom Server selbst *initiiert*. Die Ihnen geläufigste Form von Push-Dienst ist die Signalisierung eines ankommenden Telefongespräches oder der Empfang einer Kurzmitteilung (SMS). Dank einer in WAP erweiterten *Push-Architektur* [31] kann dieser Dienst auch für die Übermittlung weiterer Dienste wie Verkehrsinformationen, Signalisierung von E-mail-Eingängen, Warnmeldungen, Rundschreiben oder aber auch für die Vermarktung eingesetzt werden. Im Gegensatz zu einfachen Kurzmitteilungen können die Push-Meldungen alle Möglichkeiten der WAP-Inhaltsformate (z.B. WML mit Hyperlinks) voll ausschöpfen.

Abbildung 18.4
Server-Push über ein Gateway



In Abbildung 18.4 sieht man nun zwischen Server und Client anstelle eines allgemeinen Gateways einen sogenannten *Push Proxy Gateway* angesiedelt, dessen Funktionen in [34] festgelegt sind. Eine vom Server zu übertragene Push-Meldung wird hierbei mittels des *Push Access Protocol, PAP* [30] bis zu diesem Gateway transportiert. Das Format der Meldungen selbst entspricht einem in [32] festgelegten Format und besteht aus einem speziellem WAP-Header und einem Rumpf, der ein beliebiger MIME-Typ

sein kann. Das PAP benutzt HTTP-POST (Sie kennen die POST-Methode vielleicht schon aus der HTML-Formularverarbeitung) zur Nachrichtenübertragung.

Neben Protokoll- und Adressumsetzung muß das Push-Gateway nun auch den oder die Empfänger ausfindig machen, ggf. Meldungen zwischenspeichern, umcodieren und die Nachricht mittels des *Push OTA (Over-the-Air) Protocol* [33] an den Client ausliefern. Das Push-OTA-Protokoll setzt direkt auf den Sitzungsdienst WSP (siehe nächstes Kapitel) von WAP auf und kann so bestätigte oder unbestätigte sowie verbindungslose oder sitzungsorientierte Push-Übertragungen durchführen.

Neben den *reinen* Push-Diensten gibt es im WAP noch die sogenannten *Push-Pull-Dienste*. Hier erhält der Client nicht die komplette Nachricht zugestellt, sondern lediglich eine Verweisadresse, wo er sich bei Bedarf die Nachricht (per Pull) anfordern kann. Der im WAP hierfür spezifizierte Dienst *service indication* [36] bietet über eine XML-codierte Mitteilung eine URL an, die mit einer Kurzbeschreibung sowie mit Erstellungs- und Verfallsdatum behaftet ist. Der Benutzer kann sich dann bei Interesse oder Bedarf die dort angebotenen Informationen abholen. Wesentlich knapper und ohne Interaktion mit dem Benutzer arbeitet das sogenannte *service loading* [37]: hierbei erhält der Client ebenfalls eine URL (in XML verpackt) per Push zugestellt, der Agent im Client entscheidet nun aber selbständig, ob und wann die angebotene URL aufgerufen wird. Bei sofortigem Aufruf der URL erscheint also dieser Dienst wie ein klassischer Push-Dienst, obwohl die Nachricht per Pull angefordert wurde.

19 Die Protokolle des WAP

In diesem Kapitel werden die einzelnen Funktionsschichten des WAP-Protokolls tiefer durchleuchtet. Nach einem allgemeineren Abschnitt zur Begriffsbestimmung und Einführung geschichteter Protokolle werden die einzelnen WAP-Schichten mit Ihren Schnittstellen vorgestellt. Hierbei soll WAP »von unten nach oben«, d.h. angefangen bei dem Datagrammdienst über das Mobilfunknetz über Sicherungsschicht, Transaktionsschicht und Sitzungsschicht bis zur obersten Ebene der WAP-Anwendungen durchwandert werden.

Das Kapitel ist besonders interessant für Entwickler, die eine Programmierung eines WAP-Gerätes oder eines WAP-Gateways planen. Auch wenn Sie als solcher nicht um die Lektüre der

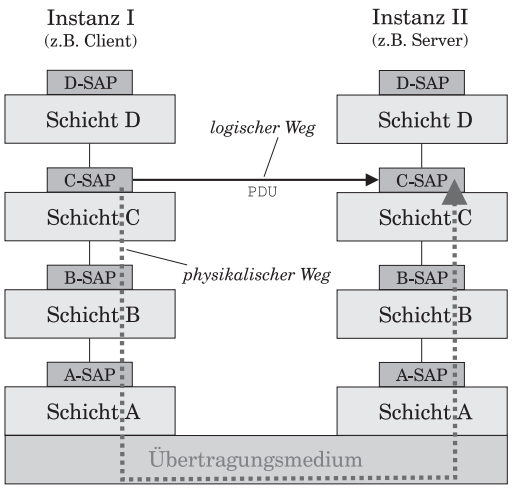
Dokumente des WAP-Forums herumkommen, so können Ihnen die folgenden Abschnitte eine gute Vorbereitung auf diese Spezifikationen liefern und zum allgemeinen Verständnis der WAP Specification Suite beitragen.

19.1 Grundlegendes zu Protokollschichten

Das WAP-Protokoll ist, wie auch viele andere komplexe Protokolle, in mehrere funktionale Abschnitte, den sogenannten *Protokollschichten*, aufgeteilt. Hierbei wird die Gesamtdienstleistung in eine Hierarchie von transportorientierten und anwendungsorientierten Funktionsschichten zerlegt, welche einzelne, in sich geschlossene Teilaufgaben erfüllen. Jede Schicht bietet jeder höhergelegenen Schicht einen Dienst an ihrem sogenannten *Dienstzugangspunkt (Service Access Point, SAP)* in Form von fest definierten *Dienstelementen* an. Diese Elemente werden auch als *Dienstprimitive* bezeichnet. Zu übertragende Datenmengen werden also stets in Form von *Parametern* eines Primitivs ausgeliefert. Eine einzelne Schicht definiert sich also einzig aus der Menge der von ihr angebotenen Dienstprimitiven und verbirgt die Implementierung der Diensterbringung vor den darüberliegenden Schichten.

Die komplette Hierarchie der Protokollschichten bezeichnet man als *Protokollstapel*.

Abbildung 19.1
Kommunikation in Schichtprotokollen



Eine Kommunikation zweier Instanzen, also »Geräte« im gleichen Protokoll, zeigt Abbildung 19.1. Die beiden Instanzen sind jeweils durch ihre Protokollstapel repräsentiert. In diesem Beispiel sei das Gesamtprotokoll in vier Schichten A bis D aufgeteilt. Die unterste Verbindungsschicht stellt das physikalische Medium (z.B. Glasfaserkabel oder GSM-Netz) dar. Die dargestellte Kommunikation zeigt beispielhaft den Austausch einer zur Schicht C gehörenden *Dateneinheit (Protocol Data Unit, PDU)* von Instanz I nach Instanz II. Allgemein kann jede Schicht einer Instanz nur mit der gleichen Schicht einer anderen Instanz kommunizieren, wobei die Daten das gesamte System unterhalb dieser Schicht durchlaufen. Der schwarze Pfeil deutet den *logischen* Weg der Transaktion an, während der gestrichelte Pfeil den tatsächlichen, *physikalischen* Datenkanal aufzeigt.

Stellen Sie sich zur Veranschaulichung exemplarisch zwei Faxgeräte als zwei Instanzen zu einem gemeinsamen Faxprotokoll vor. Obwohl jedes gesendete Fax in der untersten Schicht als Frequenzsignal auf einem Kupferkabel vorliegt, unterhält sich eine höhere Schicht mit ihresgleichen in Form von Pixeldaten. Diese Ebene kann davon ausgehen, daß darunterliegende Schichten Kommunikationsparameter aushandeln, Übertragungsfehler korrigieren oder Frequenzen aufmodulieren. In den folgenden Ab-

schnitten bestehen die beiden Instanzen übrigens ausschließlich aus WAP-Client und WAP-Server.

Um den zeitlichen Aspekt der Kommunikation in einer Protokollschicht festzuhalten, kann zu dem oben gezeigten Beispiel ein *Zeitablauf-Diagramm* zwischen Dienstnutzer und Diensterbringer erstellt werden. Die waagrechten Pfeile zeigen und benennen die einzelnen aufgerufenen bzw. ausgelösten Primitive. Abbildung 19.2 zeigt ein solches Diagramm mit einer einzigen Transaktion.

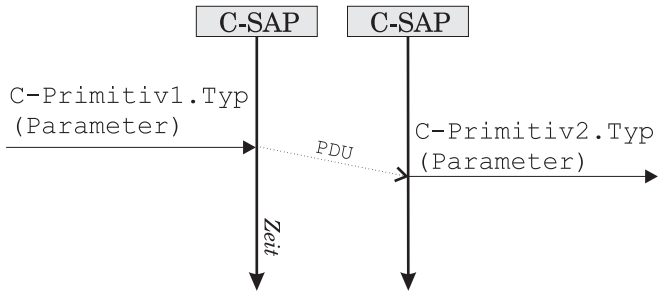


Abbildung 19.2
Zeitablauf-Diagramm

Für Primitive gibt es vier *Dienstelementtypen*, welche die Funktion eines Primitiv beschreiben [22]:

1. *Anforderung (Request, .req)* Dienstanforderung einer Einheit vom Dienstnutzer.
2. *Anzeige (Indication, .ind)* Anzeige vom Diensterbringer am Partner-SAP.
3. *Antwort (Response, .res)* Antwort des Dienstnutzers.
4. *Bestätigung (Confirmation, .cnf)* Bestätigung des Diensterbringers.

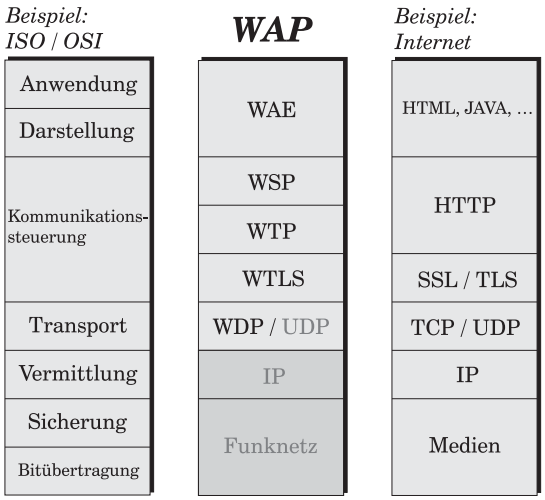
Der Typ wird, durch einen Punkt getrennt, an den Bezeichner des Primitivs angehängt, meist schreibt man noch eine Kennung des Dienstzugangspunktes mit einem Bindestrich vor den Bezeichner. Beispielsweise ist `C-PrimitivX.req` der Anforderungstyp des Primitivs `PrimitivX` am Zugangspunkt `C`.

Nach diesen allgemeingültigen Betrachtungen haben Sie nun das Handwerkszeug beisammen, um zu WAP zurückzukehren und den Protokollstapel der WAP-Spezifikation auseinanderzunehmen.

19.2 Der WAP-Protokollstapel

Der Protokollstapel von WAP wird in der *Wireless Application Protocol Architecture Specification* [28] vorgestellt und besteht insgesamt aus fünf Schichten. In Abbildung 19.3 (Mitte) wird der Protokollstapel zum Vergleich mit dem generischen ISO/OSI-Referenzmodell (links) und dem Schichtenmodell des Internets (rechts) herangezogen. Das *ISO/OSI-Referenzmodell* beschreibt kein spezielles Protokoll; es handelt sich hierbei um ein generisches Modell zur allgemeinen Beschreibung offener Systeme. Der Stapel der Internetprotokolle ist dem WAP-Modell sehr ähnlich, augenscheinlich wird HTTP jedoch im WAP durch zwei andere Protokolle (Transaktion und Sitzung) ersetzt.

Abbildung 19.3 *Der WAP-Protokollstapel*



Die im Modell dargestellten Schichten (die dunklen Bereiche gehören nicht der WAP-Spezifikation an) decken unterschiedliche, funktionale Bereiche ab:

- *Anwendungsschicht (WAE)*
  Die Anwendungsumgebung von WAP, im Original als *Wireless Application Environment (WAE)* bezeichnet, bietet die Grundlage für zahlreiche drahtlose Anwendungen, wie z.B. WML-Browser, WTA-Funktionen, WMLScript etc. (s.u.). Die hierzu vergleichbaren Komponenten im Modell des Internets sind z.B. HTML, JavaScript und Java-Applets.

- *Sitzungsschicht (WSP)*
  Das *Wireless Session Protocol (WSP)* definiert einen Sitzungszustand zwischen Server und Client, der eine Fortsetzung der Kommunikation auch nach einem Verbindungsabbruch oder nach Suspendierung der Verbindung ermöglicht. Dieser Komfort ist gerade bei einem Trägermedium wie dem relativ unzuverlässigen Funknetz sehr willkommen und unterscheidet WAP vom zustandslosen HTTP-Protokoll.

- *Transaktionsschicht (WTP)*
  Das *Wireless Transaction Protocol (WTP)* klassifiziert Transaktionen nach dem Grad der Zuverlässigkeit und führt diese aus. Sie teilt den Strom unstrukturierter Datagramme in (asynchrone) Transaktionen auf und bietet weitere Möglichkeiten, diese zu verarbeiten. Zusammen mit WSP liegen diese beiden Schichten auf Höhe der HTTP-Schicht im Internetprotokoll.

- *Sicherungsschicht (WTLS)*
  Das Protokoll *Wireless Transport Layer Security (WTLS)* wird optional eingesetzt, um sichere Sitzungen durch Chiffrierung der Datagramme zu erzeugen. Es werden Funktionen zur Verschlüsselung, Authentifikation, Integrität und zum Schlüsselmanagement bereitgestellt. In der Architektur des Internets sind in der dortigen Sicherungsschicht *SSL (Secure Sockets Layer)* bzw. *TLS (Transport Layer Security)* angesiedelt.

- *Transportschicht (WDP)*
  Das *Wireless Datagram Protocol (WDP)* liefert einen Datagrammdienst, der die darunterliegende Plattform transparent macht und somit eine Plattformunabhängigkeit der Anwendungen gewährleistet. Diese Schicht gemeinsam mit den darunterliegenden Trägermedien entsprechen der Transportschicht im Internet mit TCP/IP, UDP/IP und den Übertragungsmedien. Bietet das Trägermedium bereits das Internetprotokoll IP, so entfällt das WDP zugunsten des *User Datagram Protocol (UDP)* [16].

- *Trägermedien (Plattformen)*
  Hierbei handelt es sich um die verwendeten Mobilfunknetze und Protokolle der Telefonanbieter. Es existieren zahlreiche, unterschiedliche Technologien wie z.B. SMS, USSD oder CDMA. Die meisten der verbreiteten Systeme sind als WAP-Plattform im WDP berücksichtigt (siehe auch Seite 187).

Die weit über 30 Spezifikationen der WAP Specification Suite setzen sich zusammen aus den fünf Spezifikationen der Protokollschichten sowie aus zahlreichen ergänzenden Spezifikationen zu jeder Schicht. Über die Hälfte der Dokumente betreffen das WAE.

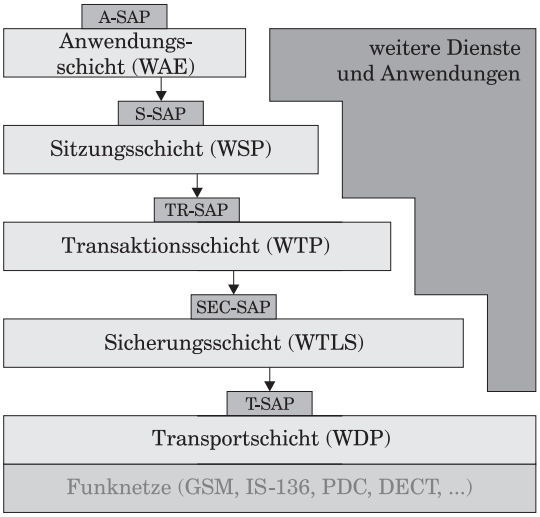


Abbildung 19.4 *SAPs im WAP-Protokollstapel*

Den vollständigen Protokollstapel einschließlich der Dienstzugangspunkte (SAPs) zeigt Abbildung 19.4. Wie im rechten Teil angedeutet kann auch nur ein Subsystem des Stapels zur Anwendung kommen, z.B. können bestimmte Anwendungen außerhalb der WAE direkt auf der Sicherungsschicht aufsetzen.

19.3 Die Transportebene

Die WAP Specification Suite in der Version 1.2 liefert zum Thema »Medienzugriff« insgesamt vier Dokumente:

- Wireless Datagram Protocol [42],
- Wireless Control Message Protocol Specification [41],
- WDP/WCMP Gateway Adaptation [38],
- WAP over GSM USSD Specification [35].

Die erste dieser vier Spezifikationen beschreibt die Transportschicht (WDP), die anderen drei Dokumente bilden Ergänzungen

zu dieser Schicht. Das WDP kennt mit `T-DUnidata` nur ein einziges Dienstprimitiv zur Übertragung von Datagrammen sowie ein weiteres Primitiv `T-DError` zur Fehlerindikation. Von den Elementtypen kommen hier nur die ersten zwei zur Anwendung: Anforderung `(.req)` und Anzeige `(.ind)`. Die beiden Dienstprimitive sind am Zugangspunkt *T-SAP* verfügbar und kommen jeweils in Begleitung von bis zu sechs Parametern:

- `Destination Address (DA)`: die Zieladresse, z.B. eine Rufnummer, IP-Adresse oder X.25-Adresse.
- `Destination Port (DP)`: die Anwendungsadresse der angeforderten Instanz bei der Zieladresse.
- `Source Address (SA)`: die Absenderadresse der dienstanfordernden Anwendung, z.B. eine Rufnummer, IP-Adresse oder X.25-Adresse.
- `Source Port (SP)`: die zur SA gehörende Anwendungsadresse.
- `User Data (UD)`: die vom WDP-Protokoll transportierten Datagramme (Nutzdaten).
- `Error Code (EC)`: Fehlerursache/Fehlerbeschreibung (dieser Parameter kommt nur beim Primitiv `T-DError` vor).

Im Falle einer Anforderung `(.req)` sind sämtliche Parameter erforderlich, bei einer Anzeige `(.ind)` sind Zieladresse und -port optional. Wird ein Fehler über `T-DError` angezeigt, so können auch hier alle Parameter verwendet werden, einzig der Fehlercode ist jedoch zwingend vorgeschrieben. Eine Fehlerindikation kann z.B. dann auftreten, wenn eine höhere Schicht eine Anfrage an einen ungültigen Port der Zieladresse signalisiert.

Abbildung 19.5 demonstriert eine gültige sowie eine fehlerhafte Dienstanforderung am T-SAP (die gewellten Unterbrechungen in den vertikalen Pfeilen heben den semantischen Bezug auf, d.h. es werden aus Platzgründen zwei oder mehr unabhängige Diagramme in eine Darstellung gepackt). Im ersten Fall wird die Anforderung angezeigt, im zweiten Fall durch eine Fehlermeldung zurückgewiesen.

Über ein IP-basiertes Netzwerkprotokoll wird WDP durch das im Internet verwendete *User Datagram Protocol (UDP)* ersetzt. Dieses wird in [16] spezifiziert und ist *nicht* Bestandteil der WAP Specification Suite.

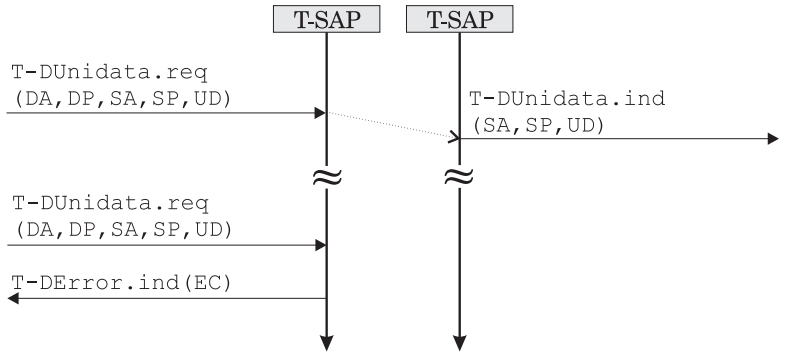


Abbildung 19.5 *Datagrammdienst (gültig bzw. fehlerhaft)*

19.3.1 Ausnahmebehandlung durch das WCMP

Im Internet benutzen Router und Hosts das *Internet Control Message Protocol (ICMP)* [17], um sich gegenseitig über Störungen zu unterrichten und Statusinformationen auszutauschen. Beim WAP erfüllt das *Wireless Control Message Protocol (WCMP)* diese Aufgabe [41]. Es ergänzt das WDP um eine effiziente Fehlerbehandlung und liegt daher ebenfalls in der Trägerschicht des Protokollstapels. Die Implementierungen des WCMP sind genau wie beim WDP abhänging vom darunterliegenden Träger, eine Mindestmenge an WCMP-Meldungen bildet jedoch einen gemeinsamen Kern, der eine gewisse Kompatibilität zwischen den Implementierungen garantieren soll. Dieser Kern beinhaltet 6 Meldungstypen, denen eine oder mehrere Meldungen angehören:

- *Destination Unreachable* (keine Route gefunden, Kommunikation nicht erlaubt, Adresse unerreichbar, Port nicht ansprechbar),
- *Parameter Problem* (fehlerhafte Angabe im Header),
- *Message Too Big* (UD enthält zu viele Zeichen),
- *Reassembly Failure* (Fehler beim Verknüpfen von Datagrammen),
- *Echo Request* (Meldungsanforderung),
- *Echo Reply* (Rückmeldung).

WCMP-Fehlermeldungen dürfen nicht andere WCMP-Meldungen beantworten. Auch darf nur eine einzige Meldung bezüglich eines speziellen, fehlerhaften Datagrammes gesendet werden. In IP-

basierten Netzen wird ICMP anstelle des WCMP verwendet, analog zur Ersetzung von WDP durch UDP.

Eine Untermenge des sogenannten *Short Message Peer-to-Peer Protocol (SMPP)* dient WAP als Tunnelprotokoll bei der Übertragung über SMS-Dienste einiger Netze oder USSD bei GSM. Ab der Version 1.2 von WAP werden in der *WDP/WCMP Gateway Adaptation* [38] die erforderlichen Elemente spezifiziert, die zur Übertragung von WDP- und WCMP-Daten zwischen dem WAP Proxy Server und dem Funk-Gateway benötigt werden. Diese SMPP-Anpassung hat die Aufgabe, vom Trägerdienst zu abstrahieren und auch über diese Dienste eine Plattformunabhängigkeit zu gewährleisten.

19.4 Sicherheit im WAP

Zur Übermittlung sicherheitsrelevanter Daten bietet WAP zahlreiche kryptographische Mechanismen, die auf die Erfordernisse der geringen Bandbreite optimiert sind. Methoden zur Verschlüsselung, zur Gewährleistung der Integrität, der Authentifikation und ein Schlüsselmanagement ermöglichen eine geschützte Punkt-zu-Punkt-Verbindung und somit den Einsatz von sicherheitsbedürftigen Anwendungen.

Die WAP-Spezifikation der Version 1.3 widmet dem Thema »Sicherheit« drei Dokumente:

- Wireless Transport Layer Security Specification (WTLS) [52],
- WAP Identity Module Specification (WIM) [43],
- WMLScript Crypto Library Specification [53].

Die ersten beiden Spezifikationen bieten Sicherheitsfunktionen, die unmittelbar auf den Datagrammdienst (WDP) aufsetzen und werden in den folgenden Abschnitten diskutiert. Das dritte Dokument erweitert WMLScript um eine kryptographische Bibliothek (siehe Abschnitt 15.7) und unterstützt somit eine Sicherheit auf Anwendungsebene.

19.4.1 Sicherheit auf Transportebene mit WTLS

Die *Wireless Transport Layer Security Specification (WTLS)* bildet den »drahtlosen Bruder« zu den im Internet verwendeten Sicherungsprotokollen *Secure Sockets Layer (SSL)* bzw. *Transport Layer Security (TLS)* und hat zahlreiche Funktionen aus diesen

Protokollen übernommen. Sie stellt die Sicherungsschicht im Protokollstapel von WAP dar und ist somit direkt über der Transportschicht (WDP) angesiedelt. Alle darüberliegenden Schichten können die Dienste des WTLS an dessen Dienstschnittstelle *SEC-SAP* wahlweise nutzen oder diese Schicht ignorieren und in einer ungesicherten Verbindung direkt auf die Transportschicht zugreifen. Die Spezifikation bietet Funktionen zur Verschlüsselung, Authentifikation, Integrität sowie ein Schlüsselmanagement. Hiermit können gesicherte Punkt-zu-Punkt-Verbindungen aufgebaut werden.

WTLS ist auf die speziellen Bedürfnisse der drahtlosen Kommunikation zurechtgeschnitten und unterstützt daher

- langlebige sichere Sitzungen,
- optimierte Handshake-Prozeduren
- und kryptographische Funktionen, welche die eingeschränkte Rechenleistung und Speicherkapazität der tragbaren Endgeräte berücksichtigen.

WTLS-Dienstprimitive

Am SEC-SAP stehen den darüberliegenden Protokollschichten insgesamt sieben Dienstprimitive zur Verfügung:

- `SEC-Unidata`: Dieses Primitiv wird zum Austausch von Daten innerhalb einer sicheren Sitzung verwendet. Die übergebenen Parameter sind identisch mit den fünf Parametern von `T-DUnidata` der WDP-Schicht:
  - Zieladresse (`DA`),
  - Zielport (`DP`),
  - Anfrageadresse (`SA`),
  - Anfrageport (`SP`)
  - und die eigentlichen Benutzerdaten (`UD`).

  In den Abbildungen 19.7 und 19.8 markiert dieses Primitiv das Ende des Sitzungsaufbaus und somit den Beginn des gesicherten Datenaustausches.
- `SEC-Create`: Hiermit wird der Aufbau einer sicheren Verbindung initiiert. Neben Start- und Zieladressen sowie den Ports (`DA,DP,SA,SP`) werden noch weitere Informationen als Parameter übergeben:

- ein (asymmetrisches) Verfahren für den Schlüsselaustausch (RSA, ECDH oder DH) (`KES`),
- eine (symmetrische) Verschlüsselungsmethode (wie z.B. RC5, DES, 3DES oder IDEA zur Datenverschlüsselung sowie die Hash-Algorithmen SHA oder MD5 zur Nachrichtenauthentifikation) (`CS`),
- eine Kompressionsmethode (derzeit wurde noch keine Methode spezifiziert) (`CM`),
- die Art der Sequenznumerierung zum Schutz vor Playback-Attacken (implizit, explizit oder keine) (`SNM`),
- die Häufigkeit einer neuen Schlüsselvergabe innerhalb einer Sitzung. Ist diese Größe n, so wird alle $2n$ Nachrichten ein neuer Sitzungsschlüssel generiert (`KR`),
- eine pro Server eindeutige Sitzungsnummer (`SID`).

Für die oben in Klammern erwähnten asymmetrischen und symmetrischen Verfahren existieren mehrere Varianten bezüglich der Schlüssellänge. Eine dazu noch weitergehende Beschreibung würde den Rahmen dieses Buchs sprengen, daher soll an dieser Stelle auf [20] verwiesen werden. Hier findet der Leser neben den genannten nahezu alle in der Kryptographie relevanten Verfahren ausführlich behandelt.

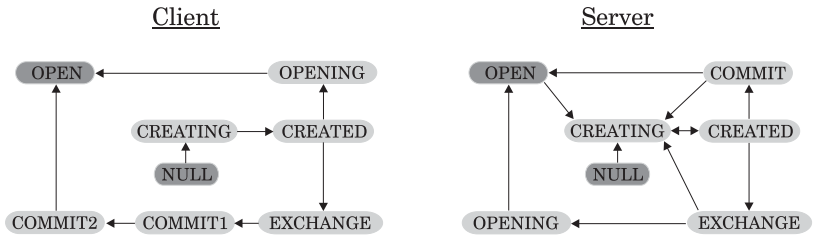
- `SEC-Exchange`: Dieses Primitiv wird benutzt, wenn der Server einen Schlüsselaustausch oder eine *Authentifizierung* anfordert. Als einziger Parameter wird das Zertifikat des Clients (`CC`) übergeben.
- `SEC-Commit`: Mit diesem Primitiv fordert eine der Seiten den Übergang in den sicheren Verbindungsstatus an, wenn der Handshake-Vorgang abgeschlossen ist. Parameter werden keine übergeben.
- `SEC-Terminate`: Hiermit wird eine Sitzung beendet. Der Grund des Abbruchs und dessen Einstufung (»fatal« bei Sitzungsabbruch oder »kritisch« bei Verbindungsabbruch) werden als Paramter übermittelt.
- `SEC-Exception`: Das Primitiv alarmiert den Verbindungspartner über auftretende Warnungen. Die Ursache der Warnung wird als Parameter übergeben.

- `SEC-Create-Request`: Dieses Dienstprimitiv wird vom Server verwendet, um bei dem Client eine neue Handshake-Prozedur anzufordern. Als Parameter werden die Adressen und Ports beider Teilnehmer übermittelt.

Die sichere Sitzung

Da WTLS ein *asynchrones* Protokoll ist, gibt es für Client und Server unterschiedliche *Sitzungszustände*. Abbildung 19.6 zeigt für beide Parteien die (vereinfachten) Zustandsdiagramme. Zur besseren Übersicht ist in den Diagrammen nicht dargestellt, daß zusätzlich jeder Zustand einen Übergang zu sich selbst (der Sitzungszustand wird beibehalten) und ebenfalls jeder Zustand einen Übergang in den Zustand NULL (die Sitzung kann jederzeit von jedem Partner abgebrochen werden) besitzt.

Abbildung 19.6
Zustände im WTLS



Den Zustandsdiagrammen kann man entnehmen, daß jeweils vom Zustand CREATED eine *Verzweigung* abgeht. Dies bedeutet, daß das WTLS *zwei* Methoden zum Aufbau einer sicheren Sitzung kennt: entweder

1. mittels der *kompletten Handshake-Prozedur* oder
2. durch einen *optimierten oder verkürzten Handshake-Vorgang*.

Ein *optimiertes Handshake* baut eine neue Sitzung auf, jedoch ist der Server bereits im Besitz des Client-Zertifikats und muß dieses daher nicht anfordern. Dieses verkürzte Handshake dient zum Aufbau einer neuen Verbindung, die auf einer existierenden sicheren Sitzung basiert (z.B. nach Verbindungsabbruch). Beim *kompletten Handshake* müssen sich die Partner durch Austausch der *Zertifikate* erst »kennenlernen«. Die Abbildung 19.7 demonstriert die Sequenz am SEC-SAP beim Verbindungsaufbau mit kompletter Handshake-Prozedur.

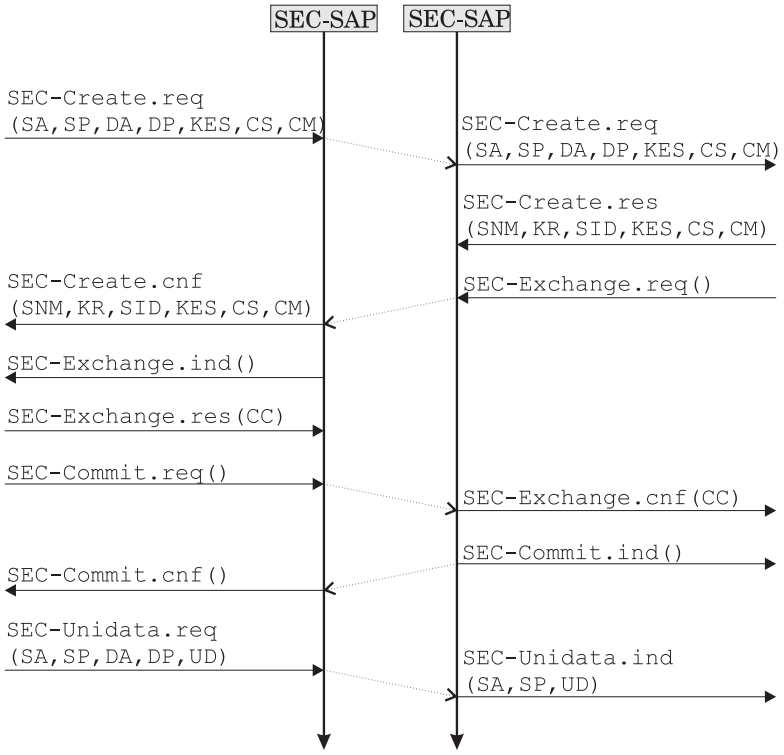


Abbildung 19.7
Aufbau einer sicheren WTLS-Verbindung (komplettes Handshake)

Nur 9 statt 13 Primitive benötigt der optimierte oder verkürzte Handshake-Vorgang bis hin zum ersten, gesicherten Datenaustausch. Wie Abbildung 19.8 zeigt, findet bei dieser Methode kein Schlüsselaustausch statt.

Innerer Aufbau des WTLS-Protokolls

Das WTLS-Protokoll selbst kann bei genauerer Betrachtung wiederum in zwei Protokollschichten aufgeteilt werden: ein sogenanntes *Record-Protokoll* und vier Clients in einer darüberliegenden Schicht: ein *Handshake-Protokoll*, ein *Überwachungsprotokoll (alert protocol)*, ein *Anwendungsdaten-Protokoll (application data)*, und das Protokoll zum *Wechsel der Verschlüsselungsmethode (change cipher)* (Abbildung 19.9).

Das *Record-Protokoll* ist für die optionale Datenkompression, Authentifikation und Verschlüsselung zuständig und liefert die Daten an die Transportschicht aus. Umgekehrt werden empfan-

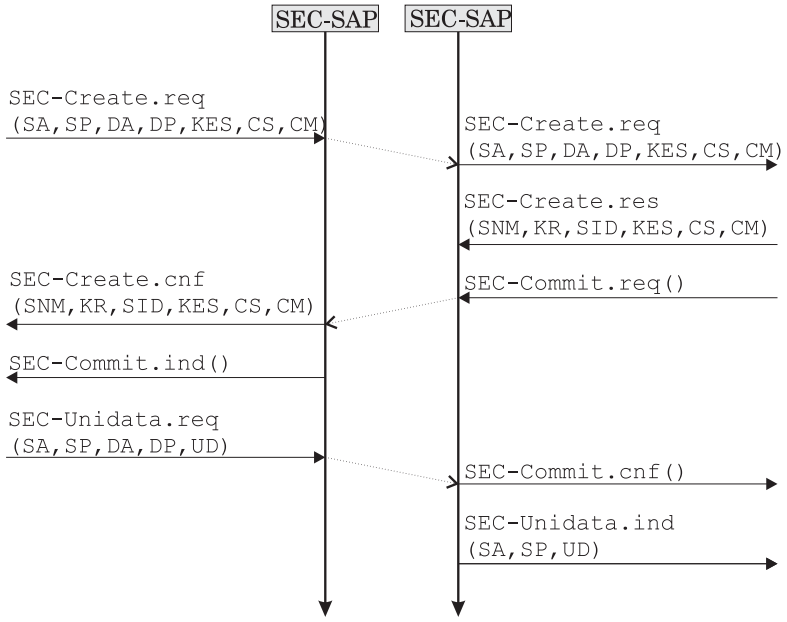


Abbildung 19.8 *Aufbau einer sicheren WTLS-Verbindung (optimiertes Handshake)*

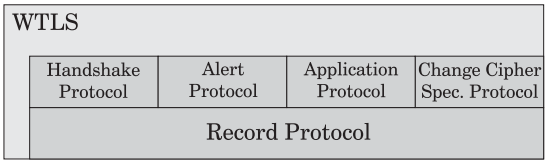


Abbildung 19.9 *Komponenten der WTLS-Schicht*

gene Nachrichten entschlüsselt, verifiziert und dekomprimiert an die darüberliegenden Clients weitergegeben.

Das *Handshake-Protokoll* ist für das Zustandekommen einer sicheren Sitzung verantwortlich. Hier authentifizieren sich die Partner (optional), einigen sich auf eine Protokollversion und Algorithmen zur Chiffrierung und nutzen Public-Key-Verfahren, um gemeinsame Schlüssel auszutauschen.

Das *Datentransfer-Protokoll* erledigt innerhalb einer sicheren Sitzung die Übergabe der Nutzdaten zwischen dem Record-Protokoll und der darüberliegenden Schicht im Protokollstapel.

Das *Überwachungsprotokoll* ist für das Versenden von Warnmeldungen (alert messages) zuständig. Diese Meldungen bestehen aus drei Komponenten: einer Warnstufe (drei Stufen: `warning`, `critical` oder `fatal`), einer Beschreibung (z.B. `deco-`

de_error, no_connection oder unknown_key_id) und einer 4-Byte Prüfsumme, um eine Zuordnung der Warnmeldung zu ermöglichen. Meldungen der Stufe »fatal« führen zum sofortigen Ende der sicheren Verbindung.

Das *Protokoll zum Wechsel der Verschlüsselungsmethode (change cipher)* besteht aus einem einzigen Byte mit konstantem Wert eins und wird von Client oder Server dazu verwendet, die Gegenpartei darüber zu informieren, daß alle nachfolgenden Nachrichten ab sofort mit den neu vereinbarten Verschlüsselungsmethoden geschützt werden. Das Signal und somit auch der Wechsel der Verschlüsselung findet innerhalb des Handshake-Vorgangs statt.

19.4.2 Das Identitätsmodul WIM

Eine Schwäche der vorhandenen WTLS-Spezifikation bis zur WAP-Version 1.1 ist die fehlende Möglichkeit, Client-Zertifikate zur Benutzeridentifikation von außen einzubringen und manipulationssicher anzuwenden. In der WAP-Version 1.2 wurde hierfür nun eine adäquate Lösung vorgestellt.

Das *WAP Identity Module (WIM)* [43] ist eine abgesicherte Einheit zur Erweiterung der Sicherheitsfunktionen der WTLS- und Anwendungsschicht. Sie wird insbesondere zur Speicherung und Verarbeitung von Informationen (Schlüssel) genutzt, die zur Identifikation und Authentifizierung der Benutzer benötigt werden. Ziel des WIM ist es, kryptographische Funktionen des Handshake-Vorgangs und private Schlüssel in eine gekapselte Umgebung zu verlagern. Abbildung 19.10 zeigt die Einordnung des WIMs in die WAP-Architektur.

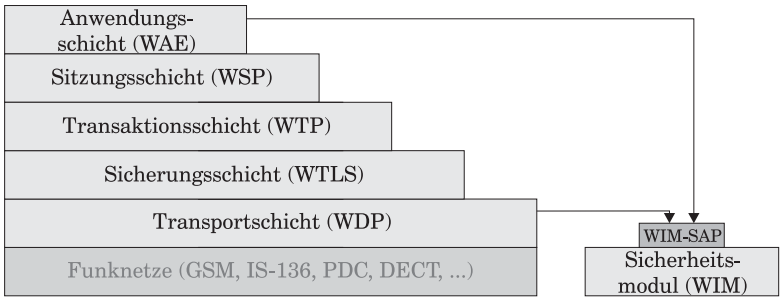


Abbildung 19.10
Zugriff auf das WIM

Insbesondere hat man die sogenannten *Smart Cards* zur Implementierung des WIMs in Betracht gezogen. Entweder dedizier-

te Chipkarten oder die im Mobiltelefon bereits vorhandene *Subscriber Identity Module (SIM)*-Karte stehen zur Auswahl. Sämtliche WIM-Anwendungen sind so konzipiert, daß sie mit den heute vorhandenen Technologien in Chipkarten implementiert werden können.

Die wichtigsten Aufgaben des WIMs sind:

- Ausführung kryptographischer Algorithmen während eines laufenden Handshake-Vorgangs,
- Absicherung langlebiger, sicherer Sitzungen,
- manipulationssichere Kapselung privater Schlüssel (Zertifikate).

An seiner Schnittstelle WIM-SAP bietet WIM zahlreiche Dienstprimitive, die sich in fünf Klassen aufteilen und an dieser Stelle nicht im einzelnen dargestellt werden sollen [43]. Die fünf Einsatzbereiche dieser Primitive sind (in Klammer steht die Anzahl definierter Dienstprimitive dieser Klasse):

- *Gerätesteuerung (2)*: Öffnen und Schließen des WIM,
- *Verifikation (5)*: Identifikation der WIM-Benutzer,
- *Datenzugriff (4)*: Datei öffnen/schließen und Daten (binär) lesen/schreiben,
- *Kryptographie (8)*: Erzeugen, Signieren und Anwenden der Schlüssel,
- *Ausnahmen (1)*: Signalisierung von Ausnahmezuständen.

Eine interessante Perspektive des WIM ist die Möglichkeit, das Modul auch jenseits von WAP zur Anwendung kommen zu lassen. Es kann auch WAP-fremden Anwendungen seine Sicherheitsfunktionen anbieten, so können z.B. Zertifikate zur Client-Authentifikation bei SSL und TLS oder zur Signatur von S/MIME-Nachrichten im WIM abgelegt werden.

Problematisch in der derzeitigen Spezifikation der Sicherungsschicht ist die Tatsache, daß eine Punkt-zu-Punkt-Sicherheit nur vom mobilen WAP-Client bis zum Gateway und nicht bis zum angesprochenen Server gewährleistet werden kann. Zwischen Gateway und Server müssen andere Mechanismen wie z.B. TLS eingesetzt werden, um die gesamte Übertragungsstrecke abzusichern. Zur Sicherung der gesamten Strecke sollte man daher die Verschlüsselung auch auf Anwendungsebene durchführen. Die Version 1.4 von WAP läßt in diesem Punkt auf Abhilfe hoffen.

19.5 Transaktionen im WTP

Der noch ungeordnete Strom von Datagrammen, der vom WDP bzw. WTLS kommt, wird durch das *Wireless Transaction Protocol (WTP)* [51] in eine strukturierte Abfolge von Transaktionen überführt. Als *Transaktion* kann hier eine konkrete Anfrage an den WAP Server/Proxy, z.B. nach einer WML-Seite mit der zugehörigen Antwort, betrachtet werden. Hierbei unterscheidet das WTP für derartige Transaktionen *drei Klassen* von Transaktionen.

19.5.1 Transaktionsklassen

- *Klasse 0: unzuverlässige, unbeantwortete Transaktionen*
  Transaktionen dieser Klasse bestehen aus einer einzigen, vom Initiator versendeten Nachricht. Der Empfänger schickt keine Empfangsbestätigung, die Transaktion ist also unmittelbar nach dem Versenden der Nachricht wieder beendet. Die Nachrichtenübertragung ist unzuverlässig und zustandslos, sie stellt somit eine *unzuverlässige Push-Übertragung* von Informationen dar.

  Transaktionen der Klasse 0 sollten die Ausnahme bleiben, da für derartige Übertragungen der Transportdienst des WDP direkt genutzt und somit das Netz effizienter ausgelastet werden kann. ✘

- *Klasse 1: zuverlässige, unbeantwortete Transaktionen*
  Push-Nachrichten, bei denen der Initiator eine Bestätigung über den Erhalt der Nachricht einfordert, fallen als Transaktionen in diese Klasse. In diesem Szenario ist die Transaktion erst dann beendet, wenn der Initiator die Empfangsbestätigung erhalten hat oder ein Transaktionsabbruch von einem Partner erfolgt ist. Bei Ausbleiben der Bestätigung innerhalb einer bestimmten Frist sendet der Initiator seine Nachricht erneut an den Empfänger. Die Bestätigung kann automatisch von der empfangenden WTP-Einheit erfolgen und somit den Absicherungsvorgang für den Benutzer transparent machen, oder es gibt alternativ die Möglichkeit, *explizit* eine Bestätigung durch den Benutzer (user acknowledgement) anzufordern.

- *Klasse 2: zuverlässige, beantwortete Transaktionen*
  In diese Klasse fallen alle Transaktionen eines typischen *Client/Server-Szenarios*. Hier gibt es genau eine Anfrage und

genau eine Antwort, die mit jeweils einer Empfangsbestätigung gegen Transportverluste abgesichert sind. Hier kann die Antwort selbst als Empfangsbestätigung dienen oder der Server sendet, um bei längerem Zeitbedarf unnötige Wiederholungssendungen zu vermeiden, eine *Zwischenbestätigung (hold on)* vor dem eigenlichen Versenden der Antwort. Auch hier können die Bestätigungen alternativ durch explizite Benutzereingaben erfolgen.

19.5.2 Funktionen des WTP

Um den gewünschten Transaktionsdienst zu erhalten, erfüllt das WTP zahlreiche, verschiedene Aufgaben. Die folgende Auflistung enthält alle in [51] dargestellten Funktionen des WTP.

- *Nachrichtenübertragung*
  Die elementare Aufgabe des WTP liegt in der Nutzdatenübertragung zwischen den Partnern. Die Übertragung wird durch ein `TR-Invoke.req`-Primitiv initiiert und über einen Parameter (`C`) einer der drei genannten Transaktionsklassen zugeordnet. Dann werden, je nach Klasse, die verschiedenen Funktionen des WTP zur Durchführung einer Transaktion genutzt.

  Eine Transaktion gliedert sich in bis zu 5 Abschnitte:

  1. Verbindungsaufnahme durch einen Initiator.
  2. Kontrolle der Transaktionsidentifikatoren.
  3. Vorläufige Bestätigung (hold on) des Empfängers.
  4. Antwort mit den angeforderten Daten.
  5. Abschließende Bestätigung des Initiators über den Erhalt der Antwort.

  Während Transaktionen der Klasse 2 alle genannten Schritte vollziehen, entfallen Schritt drei und vier bei Transaktionen der Klasse 1. In der Klasse 0 findet nur der erste Vorgang, die Verbindungsaufnahme, statt.

- *Abbruch von Transaktionen*
  Durch Emission des Dienstprimitivs `TR-Abort.req` kann ein Benutzerabbruch einer laufenden Anfrage erfolgen. Der Abbruch kann direkt von der Anwendung (z.B. durch Benutzereingabe), durch ein negatives Ergebnis (z.B. wenn der

WTP-Benutzer keine Antwort erzeugen kann) oder durch einen Protokollfehler ausgelöst werden. Hierbei ist zu beachten, daß bei Aufruf dieser Funktion die ungewollte Antwort schon unterwegs sein kann und somit der Abbruch nur die Bandbreite des Netzes unnötig belastet, ohne etwas zu bewirken.

- *Übertragungswiederholung*
  Mit der wiederholten Übertragung bis zu einer erfolgten Bestätigung kann eine zuverlässige Übertragung über ein unzuverlässiges Medium erzielt werden. Wenn ein Datenpaket verschickt und nach Ablauf einer bestimmten Zeitspanne noch nicht beantwortet wurde, so wird die ausbleibende Antwort als Datenverlust interpretiert und die Übertragung erneut gestartet. Die Anzahl der Neuübertragungen ist begrenzt, so daß nach einer maximalen Anzahl vergeblicher Versuche die Transaktion verworfen und der Benutzer über den endgültigen Verlust informiert wird.

- *Explizite Benutzerbestätigung*
  Im Normalfall erfolgt das Versenden einer Empfangsbestätigung automatisch von der empfangenden WTP-Schicht der WAP-Instanz. Um eine stärkere Form der Bestätigung zu erreichen, gibt es im WTP die Möglichkeit, eine Bestätigung anzufordern, die vom Benutzer aus der *Anwendungsschicht* explizit durch Emission eines *Response-Primitivs* abgegeben werden muß. Hierdurch wird garantiert, daß die Bestätigung vom Dienstnehmer des Partners und nicht nur von dessen WTP-Einheit kommt.

  Auch wenn diese Funktion in der Spezifikation als *optionale* WTP-Funktion definiert ist, so sollte sie stets in einer Instanz implementiert sein, da die weiter unten beschriebene Sitzungsschicht WSP auf genau diese Funktion zugreift. ✘

- *Informationen mit der letzten Bestätigung*
  Um ergänzende Informationen wie z.B. gemessene Übertragungsgeschwindigkeiten zur Qualitätsbestimmung der Verbindung zu einer Transaktion zu übermitteln, besitzt der WTP-Benutzer die Möglichkeit, diese in genau der letzten Bestätigung der Transaktion abzulegen. In Transaktionen der Klasse 2 kann der Initiator diese Informationen an den Partner (meist an den Server) zurückgeben. Die Informationen werden hier im Parameter (`ExitInfo`) des Dienstpri-

mitivs `TR-Result` abgelegt. Bei Transaktionen der Klasse 1 kann der antwortende Partner diese Informationen über das Primitiv `TR-Invoke` an den Initiator übermitteln.

- *Verknüpfung und Aufteilung der Protokolldaten*
  Um die Effizienz der Funkübertragung zu steigern, ist es möglich, mehrere PDUs in ein größeres, sogenanntes *Datagramm-Paket (SDU)* des Trägerdienstes zu verpacken. Umgekehrt können aus einem empfangenen SDU wieder die ursprünglichen PDUs extrahiert werden.

- *Asynchrone Transaktionen*
  Das WTP sollte in der Lage sein, mehrere Transaktionen nebenläufig und unabhängig voneinander durchzuführen. Der 16-Bit-Identifikator ermöglicht es daher, daß bis zu 2^{15} Transaktionen simultan bearbeitet werden können (das höchste Bit repräsentiert die Richtung der Transaktion).

- *Erzeugung von Transaktionsidentifikatoren*
  Um eine Nachricht ihrer Transaktion zuzuordnen, wird jeder Transaktion eine eindeutige Nummer, auch *Transaktionsidentifikator (TID)* genannt, zugeordnet. Diese 16-Bit große Zahl wird vom Initiator bei jeder neuen Transaktion, unabhängig vom angesprochenen Server, um Eins inkrementiert. Somit ist der Partner in der Lage, ältere von neueren Transaktionen zu unterscheiden und Duplikate zu ignorieren. Da auch die Richtung in einem Bit (Anfrage=0, Antwort=1) des Identifikators berücksichtigt wird, kann auch der Partner Transaktionen initiieren, ohne mit den erhaltenen Anfragen zu kollidieren.

- *Verifikation der Transaktionsidentifikatoren*
  Eine Nachricht kann aufgrund Paketduplizierung der Transporteinheiten mehrfach beim Empfänger eintreffen. Um Duplikate zu erkennen und herauszufiltern, wird der Transaktionsidentifikator (TID) jeder Nachricht in einem dreigeteilten Vorgang kontrolliert: Im ersten Schritt übermittelt der Initiator den TID an den Empfänger mit der Protokoll-Dateneinheit `Invoke`. Es erfolgt eine Rückfrage des Empfängers an den Initiator, ob auch tatsächlich eine Transaktion mit dieser TID auf eine Antwort wartet. Im dritten Schritt bestätigt (`Ack`) oder verneint (`Abort`) der Initiator diese Anfrage. Wird bestätigt, so kann die Transaktion durchgeführt werden, andernfalls wird sie verworfen.

- *Bereitstellung von Transportinformationen (TPIs)*
  Der Kopf der WTP-Protokolldaten (PDUs) spaltet sich in einen festen und einen variablen Anteil. Der variable Teil ist entweder leer oder er enthält Transportinformationen (*Transport Information Items (TPI)*) zur Übertragung von selbstbeschreibenden Informationen variabler Länge. Hiermit wurde das Protokoll für zukünftige Erweiterungen offengehalten und erlaubt spätere, ergänzende Definitionen sowie eine inkonsistente Benutzung weiterer Transportparameter.

- *Fehlerbehandlung*
  Tritt während einer Transaktion ein Fehler im WTP auf, so muß diese Transaktion abgebrochen und der WTP-Benutzer darüber informiert werden. Hierzu dient die Protokolldateneinheit `Abort`. Eine Möglichkeit der Fehler*behebung* ist derzeit nicht berücksichtigt.

- *Vermeidung von Versionskonflikten*
  Um eine einheitliche Kommunikationsbasis zu garantieren, überträgt der Initiator bei einer Anfrage die Versionsnummer seiner WTP-Implementierung. Der Partner vergleicht diese mit der eigenen Versionsnummer und bricht, falls die empfangene Nummer höher (neuer) ist als die eigene, die Transaktion durch Emission eines `Abort` PDU ab. Eine Abwärtskompatibilität zu früheren Versionen muß in jedem Fall gewährleistet sein.

- *Segmentierung und Reassemblierung von Nachrichten*
  Wenn die Länge einer Nachricht die maximale Paketgröße einer Transporteinheit übersteigt, so muß die Nachricht in kleinere Teile aufgesplittet (segmentiert) werden. Umgekehrt kann man zahlreiche kleine Nachrichten effizienter übertragen, wenn diese zu einem größeren Paket zusammengeschnürt und mit einem Transportvorgang übermittelt werden. Viele Netzprotokolle wie IS-136, GSM-SMS/USSD oder IP unterstüzten diese Mechanismen bereits. Eine derartige Funktionalität wurde als Option auch in das Transaktionsprotokoll WTP aufgenommen, um durch eine nun mögliche *selektive Neuübertragung* die Effizienz bei der Übertragung großer Nachrichten steigern zu können. Wird diese Funktion unterstützt, so muß bei Paketverlust nur das betroffene Paket und nicht die gesamte Nachricht erneut übertragen werden.

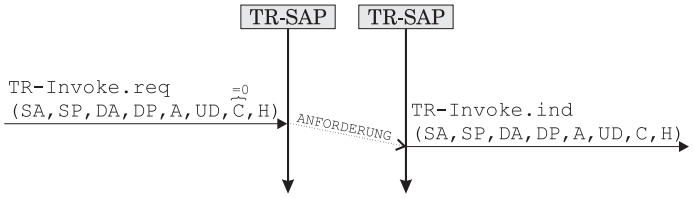
19.5.3 Protokolldienste des WTP

Eine über das WTP transportierte Dateneinheit, die *Protocol Data Unit (PDU)*, gliedert sich in einen *Datenkopf (header)* und einen *Datenteil (data)*. Der Datenkopf selbst kann weiter in einen Teil fest definierter Länge und in einen variablen Teil aufgespalten werden, der Datenteil kann auch leer sein. Insgesamt kennt das WTP sieben verschiedene Typen von PDUs, die zwischen den beiden Dienstzugangspunkten (TR-SAPs) zweier WTP-Instanzen transportiert werden können (Tabelle 19.1).

Am Zugangspunkt *TR-SAP* stehen den darüberliegenden Protokollschichten drei verschiedene Dienstprimitive zur Verfügung:

- *TR-Invoke*: Mit diesem Primitiv beginnt jede Transaktion. Die übergebenen Parameter sind neben den vier obligatorischen Adressparametern DA, DP, SA, SP (siehe Seite 213) auch noch der Bestätigungstyp (implizit oder explizit) (A), die Benutzerdaten (UD), die Transaktionsklasse (C), ein Beendigungsstatus (X) (optional bei Transaktionsende) und ein Transaktions-Handle (H) zur eindeutigen Identifikation der Transaktion. Abbildung 19.11 zeigt die Benutzung dieses Primitivs bei einer Transaktion der Klasse 0.

Abbildung 19.11 *WTP-Transaktion der Klasse 0*



Auch bei einer automatisch bestätigten Transaktion aus der Klasse 1 kommt ausschließlich dieses Primitiv zum Einsatz (Abbildung 19.12).

Abbildung 19.12 *WTP-Transaktion der Klasse 1*

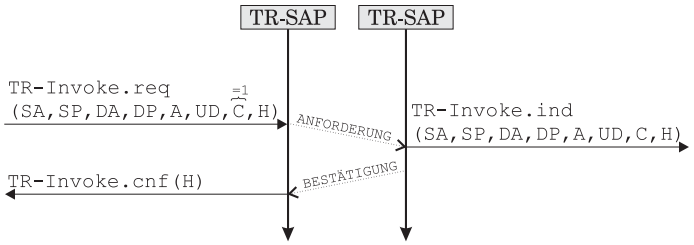


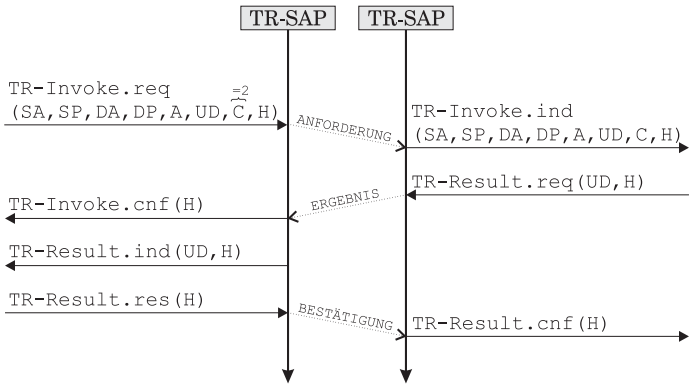
Tabelle 19.1
PDU-Typen im WTP

PDU-Typ	Funktion
`Invoke`	Von diesem Typ ist stets die erste Dateineinheit einer Transaktion und dient der Kontaktaufnahme.
`Result`	Mit dieser Dateneinheit werden die (gewöhnlichen) angefragten Daten an den Initiator übermittelt.
`Ack`	Hiermit werden Bestätigungen (Empfangsbestätigungen, Verifikation) gesendet.
`Abort`	Mit dieser Dateneinheit wird ein Abbruch, z.B. bei fehlgeschlagener Verifikation, signalisiert.
`Segmented Invoke`	Mehrere dieser Dateneinheiten können ein einzelnes Invoke ersetzen, wenn die Funktion »Segmentierung und Reassemblierung« verwendet wird (optional).
`Segmented Result`	Mehrere dieser Dateneinheiten können ein einzelnes Result ersetzen, wenn die Funktion »Segmentierung und Reassemblierung« verwendet wird (optional).
`Negative Ack`	Dieses PDU wird bei selektiver Neuübertragung der Funktion »Segmentierung und Reassemblierung« angewendet, falls bestimmte Datenpakete einer Übertragungssequenz verlorengegangen sind (optional).

✘ Bei expliziter Benutzerbestätigung müsste hier die Bestätigung durch ein `TR-invoke.res`-Primitiv des Empfängers ausgelöst werden.

- *`TR-Result`*: Dieses Primitiv überbringt das Ergebnis einer Anfrage. Neben den Benutzerdaten (`UD`), falls vorhanden, und dem Transaktions-Handle (`H`) gibt es auch hier ggf. einen Beendigungsstatus (`X`) bei Transaktionsende. Eine einfache, erfolgreiche Transaktion der Klasse 2 ohne Zwischenbestätigung und ohne explizite Benutzerbestätigung zeigt Abbildung 19.13.

Abbildung 19.13
WTP-Transaktion der Klasse 2



- *`TR-Abort`*: Mit diesem Primitiv wird ein sofortiger *Transaktionsabbruch* eingeleitet. Neben dem Handle (`H`) kann noch ein beschreibender Abbruch-Code als Parameter beigefügt werden.

Der Transaktiondienst liefert die Grundlage für ein Sitzungsmodell, welches im Vergleich zum gewöhnlichen Internet eine echte Neuerung darstellt. Die im nächsten Abschnitt vorgestellte Sitzungsschicht von WAP liefert den hierfür benötigten Dienst.

19.6 Das Sitzungsmodell

Im Protokollstapel von WAP eine Schicht über dem WTP liegt die Sitzungsschicht. An dieser Stelle beginnt WAP sich bedeutend vom klassischen Internet zu unterscheiden, da das HTTP [3],[11] von seiner Natur aus *zustandslos* agiert und daher keine Sitzungen berücksichtigt. Im Web können Sitzungszustände daher nur

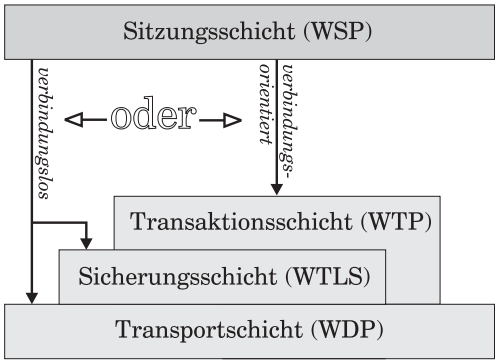
durch Mechanismen aus den Anwendungen heraus künstlich erzeugt werden:

- Durch Übergabe von *CGI-Parametern* in der URL oder in POST-Umgebungsvariablen können die Clients Zustandsinformationen zur nächsten Anfrage transferieren.
- Durch Speicherung von Informationen in sogenannten *Cookies* können Zustände erzeugt werden. Bei Cookies handelt es sich um kleine, gekapselte Speicherhäppchen innerhalb der Client-Software.
- Eine Speicherung von Informationen in *Skript-Variablen (wie z.B. JavaScript) in Kombination mit einer Fensteraufteilung (Frames)* wird in der Praxis z.B. häufig für Client-seitige Warenkorbsysteme eingesetzt.
- Eine *Transformation aller benötigten Zustände in eine einzige Transaktion* wird durch Einsatz von Client-seitigen Anwendungen wie z.B. Java-Applets oder Flash-Animationen erreicht.
- Serverseitig können Sitzungen z.B. durch Anlegen einer *IP-Tabelle* erzeugt werden. Hier wird die IP-Adresse des Clients ermittelt und diese in einer Tabelle mit zusätzlichen Zustandsinformationen (z.B. Inhalt des Warenkorbes, Verknüpfung zu registrierten Benutzern etc.) abgelegt. Die Sitzung muß dann entweder explizit vom Client unterbrochen werden oder wird nach einer bestimmten Zeit ohne Anfrage (Timeout) durch die Server-Software aufgehoben.

Die geringe Bandbreite und hohe Verzögerung der Mobilfunknetze sowie die eingeschränkte Benutzerschnittstelle würden bei einem zustandslosen Protokoll dem Anwender die Wiederaufnahme einer unterbrochenen WAP-Verbindung (man denke hier auch an Unterbrechungen, die z.B. durch eingehende Anrufe oder unvorhergesehende Ereignisse im Umfeld des Benutzers verursacht werden) weitaus mehr Mühe bereiten als beispielsweise ein Verbindungsabbruch während einer Internetbenutzung. Daher wurde in das WAP ein *Sitzungsprotokoll* integriert, welches dem Anwender bzw. der Anwendung ermöglicht, eine unterbrochene WAP-Sitzung zu einem späteren Zeitpunkt im gleichen Kontext fortzusetzen. Ein Zustand zwischen Client und Server kann also über einen längeren Zeitraum hinweg »eingefroren« und später reaktiviert werden. Als angenehmer Nebeneffekt kommt dieses Protokoll einem Einsatz von personalisierten WAP-Diensten entgegen,

da benutzerspezifische Angebote als Einstiegspunkt einer Sitzung gewählt werden können.

Abbildung 19.14
Die Sitzungsschicht



Die Sitzungsschicht wird im *Wireless Session Protocol (WSP)* [45] definiert. Das Protokoll bietet zwei grundlegende Arten von Diensten an: einen *verbindungslosen* und einen *verbindungsorientierten* Dienst (Abbildung 19.14). Letzterer nutzt die Dienste der darunterliegenden Transaktionsschicht (WTP), während der verbindungslose Dienst direkt auf einen durch WTLS gesicherten oder ungesicherten Datagramm-Dienst des WDP zugreift. Generelle Funktionsmerkmale der Sitzungsschicht sind:

- *Sitzungsmanagement (session management)*
  Sitzungen können eröffnet, geschlossen, eingefroren und wiederaufgenommen werden.

- *Aushandlung von Verbindungseigenschaften (Capability negotiation)*
  Client und Server können sich während der Errichtung einer Sitzung entsprechend ihrer Fähigkeiten auf einen gemeinsamen Nenner bezüglich der Protokollparameter bringen. Beispiele hierfür sind maximale SDU-Größe (Service Data Unit) von Client und Server, maximale Anzahl ausstehender Anfragen oder weitere Protokolloptionen.

- *Komprimierung von Inhalten (compact content encoding)*
  Verschiedene typisierte Inhalte werden in einer kompakten, binären Codierung ausgetauscht.

Neben dem allgemeingültig definierten Sitzungsprotokoll WSP (es kann auch über WAP hinaus zum Einsatz kommen) bezeichnet

[28] die speziell auf Browser-Anwendungen zugeschnittenen Dienste als *WSP/B* (das *B* steht hier für Browsing). WSP/B ermöglicht eine Verbindung zwischen WAP-Client und einem standardmäßigen HTTP-Server über einen WAP-Proxy. Weitere Eigenschaften von WSP/B sind:

- *Funktionalität des HTTP/1.1* [11] wie z.B. erweiterte Methoden für Request (Client-Anfrage) und Reply (Server-Antwort), zusammengesetzte Objekte und das Aushandeln von Verbindungseigenschaften. Um die Kommunikation kompakter zu gestalten, benutzt WSP/B eine kompakte binäre Codierung bekannter Inhaltstypen. Es kann also als eine Art »binäres HTTP« betrachtet werden.

- *Austausch von Sitzungsinformationen* durch wechselseitige Übergabe der Sitzungs-Header (Kopfdaten der Anfragen und Antworten). Client und Server tauschen und konservieren diejenigen Header ihrer Anfragen und Antworten, welche über die gesamte Sitzungsdauer konstant bleiben. Diese Header enthalten z.B. Inhaltstyp, Zeichensatz, Sprache und gerätespezifische Restriktionen. WSP/B wertet diese Informationen nicht aus, sondern reicht sie an die Benutzer in der Anwendungsschicht weiter.

- *Abbruch* (Abort) laufender Transaktionen.

- *Push- und Pull-Transfers*: Pull-Transfers beschreiben das klassische Anfrage/Antwort-Szenario, d.h. der Client fordert Daten an und der Server liefert diese aus. Bei Push-Transfers unterscheidet man drei Arten:

  1. Der bestätigte Daten-Push innerhalb eines existierenden Sitzungskontextes.
  2. Der unbestätigte Daten-Push innerhalb eines existierenden Sitzungskontextes.
  3. Der unbestätigte Daten-Push ohne existierenden Sitzungskontext.

- *Versenden asynchroner Anfragen*. Optional ermöglicht WSP/B einem Client, simultan mehrfach Anfragen zu einem Server zu senden. Dies erhöht die Effizienz der Kommunikation, da Anfragen und Antworten zu einer geringeren Anzahl Transfers zusammengefasst werden können. Da obendrein jedes

Ergebnis zum Zeitpunkt seiner Verfügbarkeit gesendet werden kann, verringern sich auch die Verzögerungszeiten innerhalb der Sitzung.

Da sich in der Praxis das WAP und somit auch das WSP derzeit und in absehbarer Zukunft ausschließlich auf Anwendungen im Microbrowser konzentriert, soll im weiteren Text auf eine sprachliche Unterscheidung von »WSP/B« und »WSP« verzichtet und schlicht vom WSP gesprochen werden. Im folgenden wird nun der verbindungslose und der verbindungsoriente Dienst der Sitzungsschicht an deren Dienstzugangspunkt *S-SAP* genauer erörtert.

19.6.1 WSP als verbindungsloser Dienst

Bevor die Erzeugung von Sitzungszuständen vorgestellt wird, soll Ihnen dieser Abschnitt verdeutlichen, wie diese Möglichkeiten *umgangen* werden können. Hierzu existieren drei Dienstprimitive im WSP, mit denen die Verwendung des Transportprotokolles WTP unterlassen wird und direkt über den Datagrammdienst WDP oder über eine gesicherte Verbindung mittels WTLS kommuniziert werden kann. Diese Variante der zustandslosen Verbindung kann dann sinnvoll sein, wenn eine Einsparung von Protokollinformationen (Overhead) und somit eine effizientere Bandbreitennutzung als wichtiger bemessen wird als die Existenz von Zuständen und Verläßlichkeit. Ein Anwendungsbeispiel hierfür ist die periodische Abfrage eines Servers auf dessen Erreichbarkeit: der Verlust eines »Ping« kann verschmerzt werden (Vorsicht: im umgekehrten Fall, d.h. der Server meldet eine Störung, muß höchste Zuverlässigkeit vorherrschen!). Auch Push-Sendungen können durchaus verbindungslos erfolgen, wenn deren Verlust keine nennenswerten Nachteile zur Folge hat.

Die drei Dienstprimitive zur verbindungslosen Übertragung sind:

- Anfrage per `S-Unit-MethodInvoke`.
- Antwort bzw. Ergebnis mit `S-Unit-MethodResult`.
- Daten-Push vom Server zum Client mittels `S-Unit-Push`.

Mit dem Dienstprimitiv `S-Unit-MethodInvoke` wird ein unbestätigter Methodenaufruf an den Server gestellt. Der Server soll die angeforderte Methode ausführen und das Ergebnis zurücksenden. Die Parameter dieser Dienstfunktion sind:

- Die Server-Adresse (SA) und die Adresse des Clients (CA).
- Eine eindeutige Identifikationsnummer der Transaktion (TID) zur Unterscheidung verschiedener Anfragen.
- Die vom Server auszuführende Methode (M). Die Methoden sind identisch mit den Methoden des HTTP/1.1-Protokolles [11], also CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE.
- Die *Request-URI* (Universal Resource Identifier) (RU). Auch diese ist unverändert aus dem HTTP-Protokoll übernommen.
- Optional: Die *Request-Headers* (RQH). Sie sind ebenfalls in ihrer Semantik identisch mit den HTTP-Headers.
- Gestattet M einen *Request-Body* (RQB), so kann auch dieser als Parameter angefügt werden. Auch dieser ist semantisch äquivalent zum HTTP Entity-Body.

Als Reaktion auf den Methodenaufruf wird das Antwortprimitiv S-Unit-MethodResult gesendet. Sie liefert, ebenfalls ohne weitere Bestätigungsmechanismen, das Ergebnis einer Anfrage an den Client zurück. Das Primitiv kommt in Begleitung folgender Parameter:

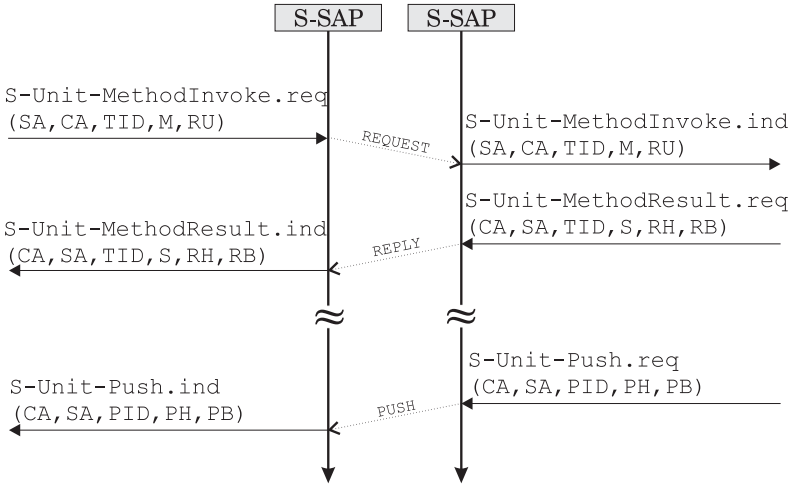
- SA, CA und TID analog zur Anforderungsprimitive.
- Eine Ganzzahl als Statusmeldung (S), deren Codierung den Statusnummern von HTTP entspricht. Beispiele: 200=OK, 404=Ressource nicht gefunden usw.
- Der *Response Header* (RH) als Liste von Attributen analog zu dem HTTP Response Header [11].
- Einen *Response Body* (RB), der im Erfolgsfall (S=200) die angeforderten Informationen und ansonsten eine genauere Fehlerbeschreibung enthält.

Zum unaufgeforderten, unzuverlässigen Push-Versand von Daten an den Client dient die Primitive S-Unit-Push. Neben CA und SA werden drei weitere Parameter mitgeliefert:

- Eine Push-Identifikationsnummer (PID), analog zur TID.
- Der Push Header (PH), analog zu den RH.
- Ein Push Body (PB) für die Inhalte, analog zum RB.

Die Übermittlung von Identifikatoren mit Transaktionen bzw. Push-Nachrichten ist zwingend erforderlich, da die Befehle asynchron abgesetzt und empfangen werden können. Eine Antwort einer späteren Anforderung kann beispielsweise die Antwort einer früher gestellten Anfrage »überholen«, falls deren Daten schneller verfügbar sind. Die Kommunikationsabläufe für Pull- und Push-Transaktionen im verbindungslosen Modus zeigt Abbildung 19.15.

Abbildung 19.15 *Verbindungsloser Datenaustausch im WSP*



Viele WAP-Anwendungen ziehen es jedoch vor, in den Genuß eines Sitzungszustandes zu kommen. Wie das WSP seinem Namen gerecht werden und diese geforderten Sitzungen ermöglichen kann, lesen Sie im nun folgenden Abschnitt.

19.6.2 Verbindungsorientierte Sitzungen mit dem WSP

Um *verläßliche Sitzungszustände* zu erzeugen, setzt das WSP auf das Transaktionsprotokoll WTP auf. Die Aufgabe von WSP kann in fünf *WSP-Einheiten* klassifiziert werden, wie in Tabelle 19.2 dargestellt ist. Die zweite Spalte enthält die in dieser Einheit implementierten Ereignisse und in der dritten Spalte eine Zuordnung zu den in Abschnitt 19.5.1 dargestellten Transaktionsklassen des WTP.

WSP unterstützt einen Mechanismus zum Aushandeln von Verbindungseigenschaften der Kommunikationspartner. Mittels dieser sogenannten *Capability Negotiation* können sich Client und Server auf einen gemeinsamen Nenner bezüglich des unterstützten Funktionsumfanges bringen und sich somit eine gemeinsame

WSP-Einheit	Ereignisse	Klassen
Sitzungsmanagement	Sitzungseröffnung (normal), Sitzungseröffnung (redirect), Sitzungseröffnung (server error), Sitzung beenden	0,2
HTTP-Methodenaufruf	Methodenaufruf	2
Suspendierung und Wiederaufnahme von Sitzungen	Sitzung einfrieren, Sitzung wiederaufnehmen (normal), Sitzung wiederaufnehmen (server error)	0,2
Daten-Push	Push-Versand	0
Daten-Push mit Bestätigung	bestätigter Push-Versand	1

Tabelle 19.2
WSP-Einheiten

Kommunikationsbasis schaffen. Die beiden Parteien einigen sich hier z.B. auf eine Menge verfügbarer Dienste, gewisse Protokollparameter und Methodenbezeichner. Nachdem der Initiator (Client *oder* Server) den Vorgang zu diesem Abgleich gestartet und die Menge seiner verfügbaren Möglichkeiten (*Requested Capabilities*) dargelegt hat, kann der Gegenpart (Responder) nun eine ihm angepasste Untermenge von Funktionen (*Negotiated Capabilities*) hieraus auswählen und dem Initiator übermitteln. Diese Untermenge gilt dann als Standard-Funktionsumfang bis zum nächsten Abgleich. Bei der Übertragung dieser Menge können auch zwischengeschaltete Dienstanbieter (z.B. Proxy-Server) ebenfalls Einfluß auf den Funktionssatz ausüben und den Umfang auf ihre eigenen Fähigkeiten weiter eingrenzen. Die folgenden Parameter müssen beim Sitzungsaufbau durch Capability Negotiation ausgehandelt werden:

- die Liste der verwendeten Alias-Adressen,
- die vom Client maximal akzeptierte SDU-Größe (Service Data Unit),
- die vom Server maximal akzeptierte SDU-Größe,
- die Menge von gemeinsam benutzten Bezeichnern für erweiterte Methoden,
- die Menge der weiteren Code-Pages-Namen, die von beiden im Header unterstützt werden, z.B. »Content-Length« oder »Bearer-Indication«,
- eine Menge weiterer Protokolloptionen, die eine Existenz weiterer Dienstangebote wie Push, bestätigter Push, Wiederaufnahme von Sitzungen und Header zur Bestätigung anzeigen.

Die Capability Negotiation findet während des *Sitzungsaufbaus* statt. Betrachtet man diese Phase als Zustand, so existieren weitere Zustände für die beteiligten Kommunikationspartner. Insgesamt können sich die Beteiligten in sechs (Server) bzw. sieben (Client) *Sitzungszuständen* befinden:

- NULL = in keinem Sitzungszustand befindlich,
- CONNECTING = eine Sitzung wird gerade aufgebaut,
- CONNECTED = in einem (aktiven) Sitzungszustand befindlich,
- CLOSING = eine Sitzung wird gerade beendet,
- SUSPENDING = eine Sitzung wird gerade suspendiert (nur Client),
- SUSPENDED = im eingefrorenen (passiven) Sitzungszustand befindlich,
- RESUMING = eine suspendierte Sitzung wird gerade wiederaufgenommen.

Bild 19.16 zeigt die vereinfachten Zustandsdiagramme. Reflexive Übergänge (alle außer SUSPENDED) und Übergänge zu NULL (alle) sind der Übersichtlichkeit wegen nicht dargestellt. Man erkennt sofort das Fehlen eines SUSPENDING-Zustandes auf Serverseite, d.h. ausschließlich dem Client ist es erlaubt, den Sitzungszustand zu suspendieren.

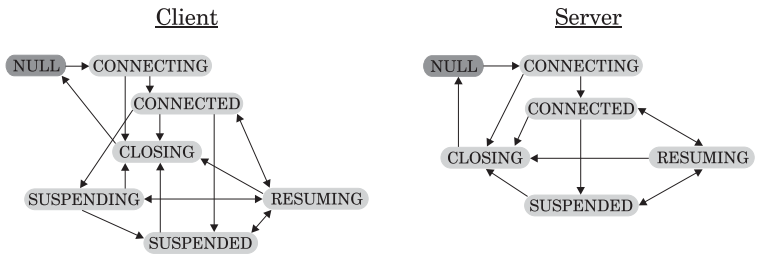


Abbildung 19.16 *Sitzungszustände im WSP*

Auch die einzelnen Transaktionen zwischen Client und Server können sich in Zuständen, den sogenannten *Transaktionszuständen*, befinden (Abbildung 19.17):

- NULL = in keinem Transaktionszustand befindlich,
- REQUESTING = ein Dienst wird gerade angefordert,
- WAITING = das Ergebnis einer Anforderung wird erwartet (Client),
- COMPLETING = das Ergebnis trifft ein (Client),
- PROCESSING = die angeforderte Methode wird ausgeführt (Server),
- REPLYING = das Ergebnis wird gerade zurückgesendet (Server),
- ABORTING = die Transaktion wird gerade abgebrochen.

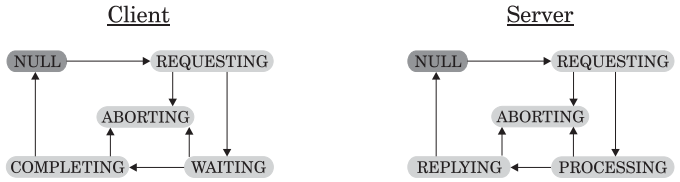


Abbildung 19.17 *Transaktionszustände im WSP*

Definiert man Zustände für Transaktionen, so muß dies analog auch für Push-Vorgänge geschehen (Abbildung 19.18). *Push-Zustände* sind:

- NULL = in keinem Zustand befindlich,

Abbildung 19.18
Push-Zustände im WSP

- PUSHING = eine Nachricht wird gerade übermittelt (Server),
- RECEIVING = eine Push-Nachricht wird gerade empfangen (Client),
- ABORTING = der Empfangsvorgang wird gerade abgebrochen (Client).

Am S-SAP (Session - Service Access Point) stellt auch diese Schicht den darüberliegenden Protokollschichten ihre Dienste über Dienstprimitive zur Verfügung. Diese Primitive und ihre Parameter werden nun einzeln aufgeführt und erläutert. Vorab liefert Tabelle 19.3 eine Übersicht aller verfügbaren WSP-Primitive. In dieser Aufstellung *nicht* aufgeführt sind die drei oben behandelten Primitive des *verbindungslosen* WSP-Dienstes.

Tabelle 19.3
Dienstprimitive im verbindungsorientierten WSP

`S-Connect`	`S-Disconnect`
`S-Suspend`	`S-Resume`
`S-MethodInvoke`	`S-MethodResult`
`S-MethodAbort`	`S-Exception`
`S-Push`	`S-Confirmed-Push`
`S-Push-Abort`	

Für den *Aufbau einer WSP-Sitzung* steht das Dienstprimitiv `S-Connect` zur Verfügung. Hiermit wird ein Sitzungsaufbau initiiert und bestätigt. Auch eine vom Client angeregtes Capability Negotiation (siehe Seite 236) wird in dieses Primitiv implementiert. Die dazugehörigen Parameter sind:

- die obligatorische Server-Adresse (`SA`) und Client-Adresse (`CA`),
- Client-Headers beim Request (`CH`) bzw. Server-Headers (`SH`) beim Result als Liste von Attributen, analog zu den HTTP Request- bzw. Response-Headers [11],

- die *Requested Capabilities* des Client (RC), wie sie bei der Capability Negotiation benötigt werden,
- und die daraufhin vom Server zurückgegebenen *Negotiated Capabilities* (NC).

Um eine *Sitzung zu schließen* und den Partner über die Schließung zu informieren, wird S-Disconnect von einem Kommunikationspartner oder dem Dienstanbieter eingesetzt. Damit niemand in einem undefinierten Sitzungszustand »hängenbleibt«, müssen alle unvollständigen Transaktionen explizit abgebrochen (abort) werden, bevor das Primitiv bei der Gegenstelle wirksam wird (.ind). Danach kann keine weitere Aktion bezüglich der beendeten Sitzung mehr erfolgen. Die Parameter von S-Disconnect sind:

- Eine Begründung in codierter Form (Reason Code) (R).
- Sollte R als Grund eine Umleitung zu einem anderen Server angeben, so muß die neue Server-Adresse (RA) und ein Sicherheitsparameter (RS) angegeben werden. Letzter bestimmt, ob eine bestehende sichere Sitzung auf der neuen Adresse fortgesetzt werden darf oder eine neue sichere Sitzung aufgebaut werden muß.
- Ergänzend zu R können Error Headers (EH) und ein Error Body (EB) als weitere Informationsquelle über Fehler genutzt werden, wenn das Sitzungsende durch einen Fehler hervorgerufen wurde.

Abbildung 19.19 zeigt die Errichtung mit anschließender Beendigung einer WSP-Sitzung.

Analog zum Errichten und Beenden existieren auch zwei Primitive zum *Suspendieren* und *Fortsetzen* einer WSP-Sitzung. Mit S-Suspend wird eine Sitzung eingefroren. Vor der Suspendierung werden alle ausstehenden Methodenaufrufe und Push-Transaktionen unverzüglich durch Abbruch beendet. Eine Suspendierung kann durch den Server, den Client oder eine dazwischenliegende Instanz ausgelöst werden. Kann ein Partner, z.B. aufgrund eines sofortigen Verbindungsabbruches, nicht mehr über die Suspendierung informiert werden, so wird dies durch die Aufforderung zur Wiederaufnahme der Sitzung durch den Partner nach Wiederherstellung der Verbindung implizit nachgeholt. Beendet ein Teilnehmer eine Sitzung gänzlich während einer Suspendierung, so kann der Partner erst wieder bei verfügbarer Verbindung durch

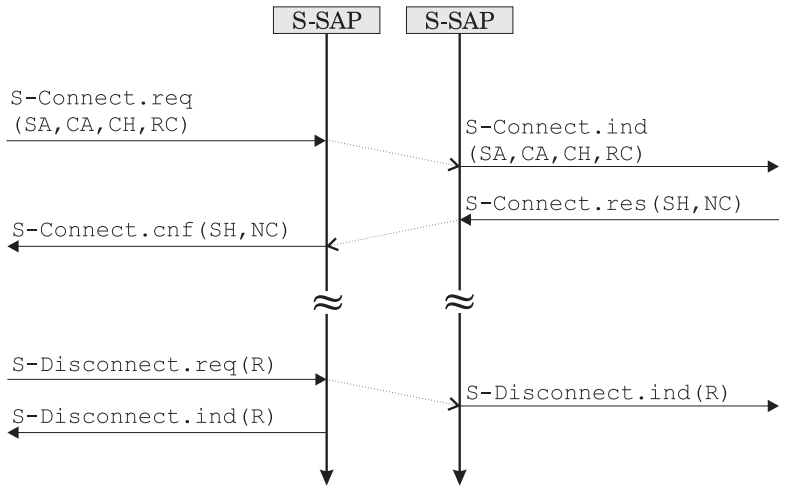


Abbildung 19.19 *Errichten und Beenden einer WSP-Sitzung*

den Dienstanbieter darüber informiert werden. Einziger Parameter dieses Primitivs ist die Angabe eines Grundes (`R`) (Reason) für die Suspendierung.

Zur Reanimation einer suspendierten Sitzung wird das Primitiv `S-Resume` eingesetzt. Als Parameter werden beigefügt:

- Die Server-Adresse (`SA`) und Client-Adresse (`CA`) zur Identifikation.
- Optional: Client-Headers beim Request (`CH`) bzw. Server-Headers (`SH`) beim Result als Liste von Attributen analog zu den HTTP Request- bzw. Response-Headers.

Die Abbildung 19.20 zeigt eine Sitzung, die vom Client (z.B. aufgrund geringer Restenergie im Akku) eingefroren und zu einem späteren Zeitpunkt (z.B. nach Akkuwechsel) wieder aufgenommen wird.

Das Dienstprimitiv `S-Exception` mit ihrem einzigen Parameter *Exception Data* (`ED`) dient der *Berichterstattung* über besondere Ereignisse, die jedoch weder mit einer bestimmten Transaktion verbunden sind noch eine Aufhebung der Sitzung zur Folge haben. Der Parameter liefert Informationen über das Ereignis, welches z.B. ein Wechsel des Mobilfunknetzes (Roaming) oder ein Wechsel der Dienstgüte (Quality of Service, QoS) sein kann.

Zum HTTP-ähnlichen *Methodenaufruf* dient das Dienstprimitiv `S-MethodInvoke`. Hiermit wird der Server aufgefordert, eine gewünschte Funktion aufzurufen und ggf. das Ergebnis an den

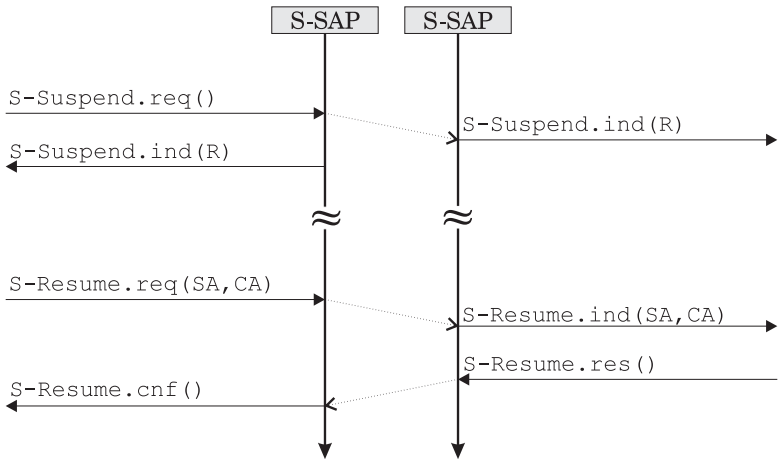


Abbildung 19.20 *Suspendierung und Wiederaufnahme einer WSP-Sitzung*

Client zurückzuliefern. Eine Reihe von Parametern begleiten den Aufruf:

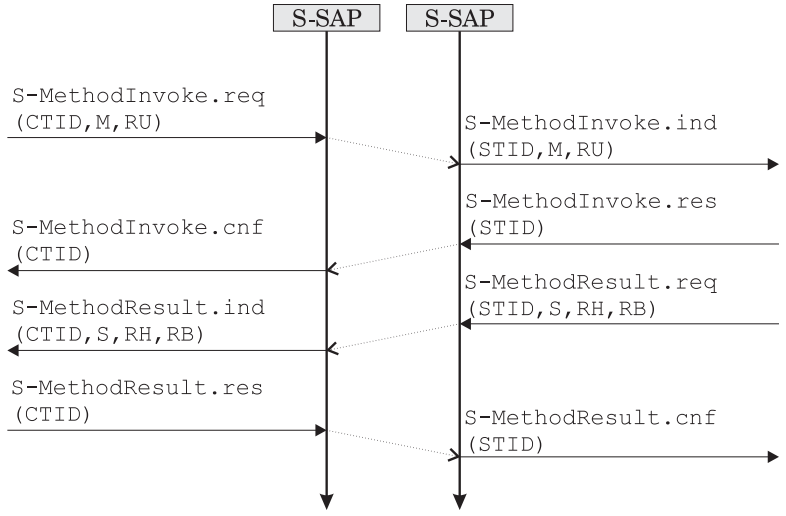
- Eine Nummer zur Identifikation der Transaktion des Clients (`CTID`) bei Anforderung bzw. des Servers (`STID`) bei Signalisierung. Hiermit wird zwischen verschiedenen, aktuell laufenden Transaktionen unterschieden.
- Die vom Server auszuführende Methode (`M`). Die Methoden sind identisch mit den Methoden des HTTP/1.1-Protokolles, also CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE.
- Die Request-URI (Universal Resource Identifier) (`RU`) zur Benennung der angesprochenen Einheit. Auch diese ist unverändert dem HTTP-Protokoll entnommen.
- Optional: Die Request-Headers (`RQH`). Diese sind ebenfalls von ihrer Semantik her identisch mit den HTTP-Headers.
- Gestattet `M` einen Request-Body (`RQB`), so kann auch dieser als Parameter angefügt werden. Er ist semantisch äquivalent zum HTTP Entity-Body.

Der Methodenaufruf wird vom Server mit dem Dienstprimitiv `S-MethodResult` quittiert. Dieses liefert das gewünschte Ergebnis der Methode bzw. entsprechende Fehlermeldungen. Folgende Informationen begleiten das Primitiv in Form von Parametern:

- Die Identifikationsnummern (`CTID`) und (`STID`) müssen identisch mit den Identifikatoren aus dem Aufruf sein.
- Eine Ganzzahl zur Statusmeldung (`S`), deren Codierung den Statusnummern von HTTP entspricht. Beispiele: 200=OK, 404=Ressource nicht gefunden usw.
- Response Headers (`RH`) können als Liste von Attributen analog zu den HTTP Response Headers beigefügt werden.
- Einen Response Body (`RB`), der im Erfolgsfalle die angeforderten Informationen und ansonsten eine genauere Fehlerbeschreibung enthält.
- Durch Acknowledgement Headers (`AH`) können optional Informationen an den Server zurückgegeben werden.

Eine komplette Transaktion innerhalb einer Sitzung mit Methodenaufruf und Antwort zeigt Bild 19.21. Der Methodenaufruf und die darauffolgende Antwort stellen *asynchrone Transaktionen* dar, d.h. die Antwort zu einer bestimmen Anfrage kann auch erst nach Ergebnissendungen zu zeitlich später gestellten Anfragen erfolgen.

Abbildung 19.21 *Request/Reply-Transaktion im WSP*



Ein »in Arbeit« befindlicher Methodenaufruf kann vor Erhalt eines Ergebnisses durch das Dienstprimitiv `S-MethodAbort` vorzeitig abgebrochen werden. Bei Aufruf muß die Identifikations-

nummer (TID) der abzubrechenden Transaktion angegeben werden. Der Server erhält den Grund (R) des Abbruches in einem weiteren Parameter mitgeteilt.

Neben dem Methodenaufruf ist der *Push-Versand* die zweite Möglichkeit, dem Client Daten zukommen zu lassen. In diesem Fall wird die Übertragung vom Server initiiert und die Informationen unaufgefordert an den Client übermittelt. Damit der Empfänger vor willkürlich gesendeten Datenmengen geschützt bleibt, bestimmen letztendlich die Dienstanbieter, welche Inhalte per Push versendet werden dürfen. Die Anzeige eines eingehenden Anrufes ist beispielsweise eine (Push-)Information, die von jedem Dienstanbieter unterstützt wird. Push-Sendungen vom Client zum Server gibt es nicht. Ein ausschließlicher Datenversand vom Client zum Server würde quasi einen Rollentausch bedeuten, der mit der Definition von Client und Server im Widerspruch stünde. Das WSP unterscheidet zwischen *bestätigten* und *unbestätigten* Push-Sendungen.

Unbestätigte und somit unzuverlässige Push-Sendungen werden durch das Dienstprimitiv S-Push übermittelt. Dieses sendet als optionale Parameter die sogenannten *Push Headers* (PH) gemäß der HTTP-Spezifikation [11] und einen *Push Body* (PB) für die zu übertragenden Daten. Eine Nachricht kann ohne weitere Konsequenzen oder Meldungen verlorengehen.

Mehr Sicherheit erhält man durch den Push-Versand mittels S-ConfirmedPush. Hiermit wird dem Server über Erfolg oder Mißerfolg der Zustellung einer Nachricht berichtet. Zusätzlich zu den bei S-Push übertragenen Daten werden bei diesem Primitiv auch Identifikationsnummern mitgeliefert, die dem Client (CPID) beim Empfang und dem Server (SPID) beim Senden eine eindeutige Einordnung einzelner Push-Nachrichten erlauben. Ferner können über sogenannte *Acknowledgement Headers* (AH) optional Informationen an den Server zurückgegeben werden.

Nachrichten durch unbestätigten Push-Versand können vom Client durch bloßes Ignorieren abgelehnt werden. Zur Verweigerung der *bestätigten* Push-Sendungen kann oder muß der Client (oder ein Dienstanbieter) das Dienstprimitiv S-PushAbort einsetzen. Als Parameter wird die Nummer der verweigerten Nachricht (PID) und eine Begründung in Form eines Zahlenwertes (R) mitgeliefert. Dem Server bleibt daher zwar auch bei der Primitive S-ConfirmedPush in einigen Fällen die Auslieferung verwehrt, doch wird er zumindest durch die erforderliche explizite Ablehnung über die Nichtauslieferung informiert.

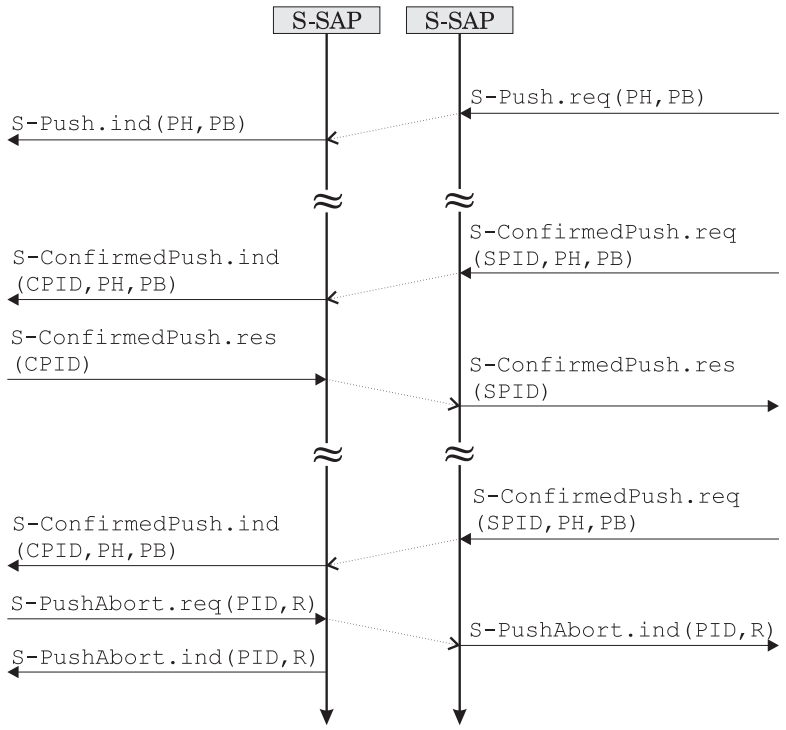


Abbildung 19.22 *Die drei Push-Varianten im WSP*

Abbildung 19.22 zeigt alle drei Varianten: Einen unbestätigten, einen bestätigten und einen explizit abgelehnten Push-Versand.

19.7 Die Anwendungsumgebung WAE

Die Spitze des WAP-Protokollstapels bildet eine plattform- und geräteunabhängige Anwendungsumgebung, das *Wireless Application Environment* (WAE) [39]. Es besteht aus einer Menge von sogenannten Benutzerinstanzen (*User Agents*), die zusammengefaßt auch als *Microbrowser* bezeichnet werden. Darüber hinaus umfaßt das WAE noch mehrere Inhaltsformate (Contents). Tabelle 19.4 auf der nächsten Seite zeigt einen Überblick über die Elemente des WAE.

Die Tabelle ist – durch die Punkte angedeutet – offen ausgelegt, da mit jeder neuen WAP-Version neue Elemente hinzugefügt und vorhandene Elemente ausgebaut werden können. Auf Elemente, die zwingend erforderlich sind bzw. als Option vom Geräte-

Inhaltsformate	Benutzerinstanzen
Wireless Markup Language (WML)	WML User Agent
WMLScript Language	WMLScript User Agent
Telefonie-Dienste (WTA)	WTA User Agent
URLs	...
...	

Tabelle 19.4
Elemente der WAE

hersteller implementiert werden können, hat sich das WAP-Forum bis zu WAP 1.3 noch nicht exakt festgelegt. Die in der Tabelle aufgeführten Komponenten sind heute jedoch in jedem vorhandenen WAP-Endgerät verfügbar, der Entwickler kann sich also auf das Vorhandensein dieser Elemente verlassen.

Die zur Umgebung definierten Inhaltsformate bedienen sich so oft wie möglich bewährter Konzepte und Standards. Sie können auf bereits heute millionenfach vorhandenen Hostrechnern im globalen Internet angesiedelt sein und somit eine gigantische Infrastruktur nutzen. So betrachtet stellen sie eine *Erweiterung des Internets* um zusätzliche Inhaltstypen unter Substitution des vorhandenen HTTP dar.

Die sich seit Mitte der 90er im klassischen Kommunikationsbereich unter dem Namen *Computer Telephony Integration* (CTI) stark verbreitete Auflösung der Grenze zwischen Computer- und Telefonieanwendungen hält nun auch Einzug in den mobilen Bereich: Die Schnittstelle *Wireless Telephony Application (WTA)* bzw. ihr User Agent im Microbrowser ermöglicht innerhalb einer typischen, datenorientierten Umgebung einen Zugriff auf Telefonfunktionen des Mobilfunknetzes und des Mobiltelefones.

Das WAE wurde speziell für Endgeräte mit geringer Speicherkapazität und schwacher Prozessorleistung entwickelt und setzt der geringen Bandbreite eine vollständig datenkomprimierte Übertragung entgegen. Das Prinzip der minimalen Bandbreite durchläuft alle Anwendungen wie ein roter Faden: Sämtliche Objekte werden vor ihrer Reise durch das Mobilfunknetz in ein schlankes, bytecodiertes Format gebracht. Die nahezu komplette Beseitigung von Redundanz verringert das Übertragungsvolumen, entlastet somit das Funknetz und führt zu einer effizienteren Kommunikation. In WML-Dokumenten kann durch Einsatz von Variablen und Schablonen (*Templates*) Platz eingespart wer-

den. Die Dokumente werden dann als *Binary XML* in kompakter binärer Form übertragen und erst im Microbrowser des Endgerätes wieder decodiert und interpretiert. Bei der Entwicklung der Skriptsprache WMLScript haben sich die Entwickler an der Architektur von Java orientiert: das WMLScript-Programm wird vor der Übertragung in eine Art Bytecode - man könnte es fast Applet nennen - kompiliert. Dieser wird beim Client in einer virtuellen Maschine, dem sogenannten *WAEScript User Agent*, ausgeführt. Selbst die Protokolldatenköpfe (Headers) von Requests und Replies werden vor der Übermittlung in ein gepacktes Format gewandelt. Auch das WAP-eigene Bildformat WBMP spart Platz – allerdings auf Kosten des Komforts: WBMP versprüht in punkto Qualität und Platzbedarf den Charme der Computergrafik in den frühen 80er Jahren. Ein erweitertes Caching-Modell sorgt überdies für weitere Optimierung bei der Funkübertragung. Insgesamt wurde also bei der Konstruktion aller Objekte stets auch die Bandbreite im Auge behalten.

WML und WMLScript werden im Microbrowser strikt getrennt behandelt. Die Kommunikation zwischen diesen beiden Elementen findet über *Umgebungsvariablen* und Verweise statt. Diese Kapselung ermöglicht es beispielsweise, bei einem Verzicht auf WMLScript die Script-Objekte überhaupt nicht anzufordern und somit das Netz zu entlasten. Bei HTML-Dokumenten werden die Skripts fast immer automatisch »mitgekauft«, unabhängig davon, ob sie ausgeführt werden (können) oder nicht. Auch eine Speicherung separater Skripts im Cache-Speicher des Proxy-Gateways kann zur Einsparung des Transfervolumens beitragen.

Nach allen in diesem Kapitel ergründeten Tiefen der Protokollschichten trifft man mit den WAE-Objekten wieder genau auf diejenigen Bereiche, mit denen auch der Entwickler für WAP-Anwendungen häufig konfrontiert wird. Dieser Bedeutung soll mit einem eigenen Kapitel für diese Elemente Rechnung getragen werden: im anschließenden Kapitel werden die einzelnen Inhaltsformate der WAE ausführlich beschrieben.

20 WAE-Anwendungen

Hier werden Ihnen alle Anwendungen der im vorangegangenen Kapitel beschriebenen Ablaufumgebung WAE im Detail vorgestellt. Neben den Sprachen WML und WMLScript, die Sie schon ausführlich im vorderen Teil des Buches kennen- und programmieren gelernt haben, tummeln sich noch weitere Objekte innerhalb der Anwendungsumgebung. Jeder der dort spezifizierten Anwendungen ist in diesem Kapitel ein Abschnitt gewidmet.

20.1 Die Wireless Markup Language WML

Alle im Microbrowser darzustellenden Informationen werden in Dokumenten gehalten, welche dem Standard der *Wireless Markup Language (WML)* [44] gerecht werden. Wie bereits im vorderen Teil des Buches beschrieben, basiert WML auf geklammerten Steuerbefehlen (Tags), die den Informationen spezielle Attribute (z.B. fett, Überschrift, Verweis etc.) zuweisen. WML selbst stellt die Information nicht dar, die optische Aufbereitung erfolgt geräteabhängig im Microbrowser.

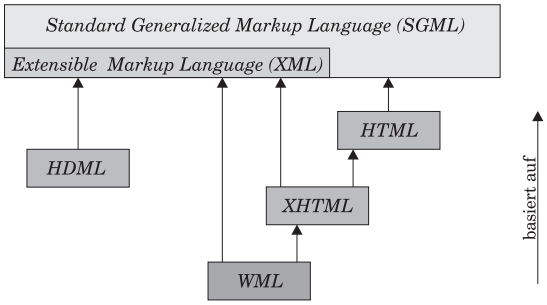
WML ist eine XML-konforme Auszeichnungssprache, die zahlreiche Elemente von der im WWW geläufigen *Hypertext Markup Language (HTML)* sowie von der *Handheld Device Markup Language (HDML)* [15] geerbt hat.

Alle diese Sprachen sind Anwendungen der *Standard Generalized Markup Language (SGML)*, einer ISO-standardisierten Beschreibungssprache. Diese definiert ein Verfahren zur Beschreibung von Auszeichnungssprachen. Da SGML eine sehr umfangreiche und überdies auch umständliche Norm darstellt, hat man daraus eine insbesondere auf elektronische Medien zugeschnittene, überschaubare Teilmenge extrahiert und als *Extensible Markup Language (XML)* [56] bezeichnet. Nicht konform zu XML, jedoch einer durch XML beschreibbaren Struktur sehr ähnlich, ist die populär gewordene Auszeichnungssprache HTML. Nahezu jede im WWW übertragene Information ist in einem HTML-Dokument

eingebettet. Die Weiterentwicklung von HTML, die sogenannte *Extensible Hypertext Markup Language (XHTML)*, wurde später als gültige XML-Anwendung definiert. An diese Sprachnorm musste sich WML ab der WAP-Version 1.1 anpassen und wurde daher inkompatibel zur Vorgängerversion 1.0.

Speziell für Endgeräte im Mobilfunkbereich wurde die Sprache HDML [15] von einer Firma names Unwired Planet, heute unter dem Namen Phone.com ein führender Hersteller von WAP-Microbrowsern für Mobiltelefone, entwickelt und im Mai 1997 dem W3C-Gremium vorgeschlagen. Die Sprache konnte sich jedoch seinerzeit nicht als Standard durchsetzen, obwohl sie doch einige Verbreitung auf dem amerikanischen Markt fand. Zahlreiche Elemente dieser ebenfalls XML-basierten Beschreibungssprache sind in WML eingearbeitet worden, so daß eine starke Ähnlichkeit zwischen diesen beiden »drahtlosen« Auszeichnungssprachen besteht. Tatsächlich können HDML-Dokumente problemlos nach WML, z.B. mit einem frei verfügbaren Toolkit von Ericsson (www.ericsson.com), konvertiert werden. Abbildung 20.1 zeigt den Zusammenhang zwischen allen genannten Sprachdefinitionen.

Abbildung 20.1
WML und Auszeichnungssprachen



WML dient *nicht* zur Darstellung von Dokumenten, sondern beschreibt lediglich deren Struktur, ebenso wie HTML oder andere Auszeichnungssprachen. Für die Anzeige im Display ist der *WML User Agent*, der WML-Browser, zuständig. Dieser decodiert die binären WML-Dokumente und interpretiert sie in das Anzeigefenster des WAP-Endgerätes. Man spricht in diesem Zusammenhang auch vom *Rendern* der Informationen. Hierbei können die sichtbaren Ergebnisse zwischen verschiedenen Geräten erheblich voneinander abweichen, da insbesondere die Auflösung von Display zu Display stark variiert. Es bleibt also festzuhalten, daß auch im WAP das leidige Problem inkompatibler Browser vorherrscht und man nur dann wirklich auf der sicheren Seite liegt,

wenn man sämtliche Programmierungen auf dem kleinsten Nenner durchführt.

MIME-Typen (MIME = Multipurpose Internet Mail Extensions) liefern Informationen über bestimmte, allgemein bekannte Inhaltstypen. Vor dem eigentlichen Objekt erhält der Internet-Client die Information, was für ein Objekttyp (z.B. JPG-Grafik) auf ihn zukommt. So kann dieser entsprechend reagieren und das ankommende Objekt benutzergerecht weiterverarbeiten. Für WML-Dokumente gibt es zwei MIME-Medientypen, die sich in den Headers der HTTP-Transaktionen wiederfinden:

```
text/vnd.wap.wml
```

für Dokumente in Textform und

```
text/vnd.wap.wmlc
```

für Dokumente im Binärformat.

Um zur Entlastung der Bandbreite mehrere Informationsseiten in eine einzige Übertragung zu packen, wurde eine ausgeklügelte dokumenteninternen Navigation entwickelt. Es entstand die Metapher des Kartenspieles (*Deck*): das in einer Transaktion übertragene Deck enthält mehrere Karten (*Cards*), von denen jeweils eine Card den Inhalt eines Gerätedisplays beschreibt. Die Navigation erlaubt Sprünge zwischen den einzelnen Karten, so daß erst bei einem Verweis auf ein neues Deck eine weitere Transaktion über das Netz erfolgen muß.

Eine weitere Optimierung mit dem Ziel der Netzentlastung stellt die sogenannte *String-Parametriesierung* dar: WML-Textbausteine lassen sich hiermit auf Variablen reduzieren, die zur Laufzeit wieder durch den Originaltext substituiert werden. So können Sie beispielsweise innerhalb einer Card mittels des Befehls `setvar` einen langen Namen in einer Variable `$name` ablegen. Diese Variable ist nun innerhalb des Browserkontextes überall verfügbar und kann in jedem darauffolgenden Deck und jeder Card stellvertretend für den langen Namen eingesetzt werden.

Eine ausführliche Beschreibung der WML-Syntax mit zahlreichen Beispielen finden Sie in den voranstehenden Kapiteln dieses Buches.

20.2 Die Skriptsprache WMLScript

Bei *WMLScript* [54] handelt es sich um eine prozedurale Skriptsprache, die analog zu den WWW-Skriptsprachen in HTML die

statischen WML-Dokumente um interaktive Funktionalität erweitert. Die Skriptsprache ist eng verwandt mit der in vielen WWW-Browsern verfügbaren Sprache JavaScript, da sie eine auf die Bedürfnisse einer mobilen Umgebung optimierte Variante von *ECMA-Script* [5] darstellt. Bei dieser handelt es sich um eine von der ECMA (European Computer Manufacturing Association) standardisierte Skriptsprache, die sich zum Ziel gesetzt hat, proprietäre Implementierungen wie JavaScript (Netscape) und JScript (Microsoft) in einen offenen Standard münden zu lassen und somit Entwicklern die browserspezifischen Mehrfachanfertigungen von skriptgestützten HTML-Dokumenten zu ersparen.

Die Sprache WMLScript verfolgt vier Hauptziele:

- *Prüfung und Formatierung von Benutzereingaben* wie beispielsweise erlaubte Zahlenbereiche oder korrekte Telefonnummern.
- *Zugriff auf geräte- oder herstellerspezifische APIs* (Application Programming Interfaces). Es können z.B. per WTA (s.u.) Telefongespräche initiiert, das interne Telefonbuch bearbeitet und Zugriffe auf die interne SIM-Karte durchgeführt werden.
- *Schnelle Interaktion durch lokale Ereignisbehandlung.* Ereignisse wie z.B. Fehleingaben werden dem Benutzer prompt zurückgegeben, da keine Inanspruchnahme des Mobilfunknetzes erfolgen muß.
- *Eine Benutzerinstanz soll durch Funktionsbibliotheken von WMLScript nachträglich erweiter- und veränderbar sein.* Somit wird dem Endgerät zusätzliche »Intelligenz« durch ein Software-Update verliehen.

Bevor ein WMLScript-Programm über das Funknetz den WAP-Client erreicht, wird es im Gateway von einem Compiler in einen *Bytecode* kompiliert und dann in kompakter Form übertragen (Abbildung 18.3 auf Seite 203). Der *WMLScript User Agent*, also der Interpreter im Endgerät, führt dann diesen Bytecode bei Funktionsaufrufen aus. Bei einer Abarbeitung kann der Interpreter stets auf vier Statusinformationen zugreifen:

- *Befehlszähler*: Dieser zeigt auf die Stelle des momentan ausgeführten Befehles im Bytecode.
- *Variablen*: globale Funktionsparameter und Variablen.

- *Stapel der Operanden*: Wird zur Berechnung von Ausdrücken und Übergabeparametern verwendet.
- *Stapel der Funktionsaufrufe*: Lagert Informationen über verschachtelte Funktionsaufrufe (Rücksprungadressen) innerhalb des eigenen oder auch zu einem anderem Bytecode.

Im WWW ist es üblich (wenn auch nicht zwingend), Skript-Programme mittels eines `<script>`-Tag direkt in das HTML-Dokument einzubetten. Nicht so im WAP: Programme in WMLScript werden ausschließlich in eigenen, getrennten Objekten abgelegt und über URLs aufgerufen. Eine direkte Einbindung von Programmcode in ein WML-Dokument ist nicht möglich. Ein Aufruf einer WMLScript-Funktion erscheint in der Gestalt

```
URL/Skript#Funktion(Parameter).
```

Beispielsweise steht

```
http://wap.selfwml.de/wuerfel.wmls#werfen()
```

für den (parameterlosen) Aufruf der Funktion `werfen` innerhalb des WMLScript-Dokuments `wuerfel.wmls`, anzutreffen im Internet unter der (beispielhaften) Adresse `http://wap.selfwml.de`. Die MIME-Medientypen für WMLScript lauten

```
text/vnd.wap.wmlscript
```

für WMLScript als Text und

```
text/vnd.wap.wmlscriptc
```

für WMLScript in Bytecode.

Betrachten wir zur Veranschaulichung ein kurzes Beispiel (Listing in Abbildung 20.2) für den Einsatz von WMLScript. Das Programm soll einen Würfel simulieren, d.h. eine Zufallszahl zwischen 1 und 6 generieren. Die Abbildung besteht aus einem WML- und einem WMLScript-Dokument. Das WML-Dokument zeigt dem Benutzer den Würfel an. Bestätigt dieser per Tastendruck bei `wurf`, so wird die Funktion `werfen` im WMLScript `wuerfel.wmls` im gleichen Verzeichnis ohne Parameter (leere Klammern) aufgerufen. Die aufgerufene Funktion generiert eine Zufallszahl per `Lang.random(6)` und legt diese in einer Umgebungsvariable `AUGEN` im WML User Agent ab. Dann wird die Funktion durch einen Sprung zurück in die WML-Datei zur zweiten Card (`karte2`) beendet. Diese Card kann nun durch Auslesen der Umgebungsvariablen `$(AUGEN)` die gewürfelte Zahl integrieren.

Abbildung 20.2 *Einsatz von WMLScript*

```
<!-- WML-Dokument (wuerfel.wml) -->

  <wml>
    <card id="karte1" title="Wuerfeln">
      <p>Wuerfel werfen</p>
      <do type="accept" label="wurf">
        <go href="wuerfel.wmls#werfen()"/>
      </do>
  </card>
  <card id="karte2" title="Ergebnis">
    <p>gewuerfelt: $(AUGEN)</p>
  </card>
</wml>


<!-- WMLScript-Datei (wuerfel.wmls) -->

extern function werfen() {
  var a = Lang.random(6);
  WMLBrowser.setVar("AUGEN",a);
  WMLBrowser.go("wuerfel.wml#karte2");
}
```

Die Berechnung der Zufallszahl durch `Lang.random(6)` läßt schon erahnen, daß WMLScript mit *Funktionsbibliotheken* arbeitet. Tatsächlich sind nur die elementarsten Funktionen wie einfache arithmetische Operatoren, Bedingungsabfragen und Schleifenkonstrukte im Sprachkern verankert. Um dennoch zu einem akzeptablen Funktionsumfang zu gelangen, wird in der WAP-Spezifikation gleich ein ganzes Paket von Standardbibliotheken mitgeliefert, die bei Bedarf eingebunden werden können [55]. Auch diese Maßnahme minimiert die benötigten Funknetz-Ressourcen, da nur wirklich benötigte Funktionen den knappen Speicher des WAP-Clients beanspruchen. In WAP 1.3 werden insgesamt sechs Standardbibliotheken plus eine Crypto-Bibliothek [53] definiert, die in Tabelle 20.1 knapp zusammengefasst sind.

Eine interessante Perspektive liegt in der Möglichkeit, WML und WMLScript in Form von Bytecode permanent in einem ROM-Speicher im Endgerät abzulegen. Beispielsweise kann so ein Taschenrechner, wie er heute in jedem modernen Mobiltelefon proprietär implementiert ist, als WMLScript abgelegt und so inner-

Tabelle 20.1
Standardbibliotheken von WMLScript

Bibliothek	Funktionen
Lang	Ergänzende Funktionen zur Typenumwandlung, arithmetische Funktionen, Umgebungsabfragen, Flußkontrolle und Zufallszahlen.
Float	Funktionen zur Handhabung von Fließkommazahlen.
String	Zahlreiche Funktionen zur Zeichenkettenbearbeitung und -umwandlung.
URL	Funktionen zum Prüfen, Abfragen, Manipulieren und Aufsplitten von URL-Zeichenketten.
WMLBrowser	Setzen und Abfragen von Umgebungsvariablen, Steuerung der WML-Benutzerinstanz und Wechsel zu Deck/Task nach Programmende.
Dialogs	häufig benötigte Standard-Dialogboxen (ok, cancel, wait usw.)
Crypto	Funktionen zur Unterstützung sicherer, verschlüsselter Übertragungen

halb der WAP-Umgebung genutzt werden. Auch der Einbettung von Spielen und sonstigen Zusatzfunktionen in WMLScript steht unter WAP nichts im Wege.

Von einer tieferen Betrachtung der Struktur und Syntax von WML und WMLScript soll an dieser Stelle abgesehen und auf den vorhergehenden Praxisteil verwiesen werden. Ein genauerer Blick auf die Telefoniedienste der WTA darf jedoch hier nicht fehlen und soll daher im nächsten Abschnitt einen weiteren Kontrast zu den klassischen WWW-Diensten liefern.

20.3 Telefoniedienste mittels WTA

Einen bedeutenden Teil der WAP-Spezifikation stellt das *WTA-Gerüst* (WTA Framework) [50] dar. Dieses integriert die gewöhnlichen Dienste der Telefonie und Funktionen des (mobilen) Telefonnetzes in die WAE und gestattet so eine Vermengung der Telefonfunktionen mit den datenorientierten Diensten und Inhaltsformaten des WAP. Das WTA-Gerüst bereichert die WAE um zahlreiche Komponenten:

- Eine WTA-Benutzerinstanz, der *WTA User Agent*. Dieser empfängt beispielsweise Ereignismeldungen des Mobilfunknetzes oder Push-Nachrichten.
- Eine Schnittstelle zwischen den im Client verfügbaren Telefonfunktionen und WML/WMLScript. Diese ist im WAP unter dem Namen *Wireless Telephony Application Interface (WTAI)* [46] bekannt.
- Eine *Bearbeitung von Ereignissen aus dem Mobilfunknetz* durch die WTA-Benutzerinstanz. Beispielsweise kann auf einen eingehenden Anruf in Abhängigkeit von der signalisierten Rufnummer differenziert reagiert werden.
- Einen persistent speichernden Container für Inhaltsobjekte, das sogenannte *Repository*, welches von einem WTA-Server gefüllt und im Client lokal aufgerufen werden.
- Ein Sicherheitsmodell, das nur authentifizierten Zugriff auf bestimmte (private oder kostenverursachende) Telefondienste erlaubt.

Der *WTA-Server* (Abbildung 18.3 auf Seite 203) bietet verschiedene Telefondienste unter Verwendung der im WTA-Gerüst definierten Funktionen an. Er unterliegt, im Gegensatz zu beliebigen Content-Servern im Internet, der Kontrolle der Betreibers des verwendeten Mobilfunknetzes. Hierdurch wird ein Sicherheitsmodell geschaffen, welches WTA-Funktionen aus dem Netz des eigenen Anbieters von WTA-Funktionen fremder Drittanbieter abgrenzt. Die Funktionen des WTA-Gerüstes wurden daher in drei Bibliotheken aufgeteilt:

- Die Funktionen der *öffentlichen Dienste* (public) sind allen Benutzern zugänglich und umfassen das Anrufen von Telefonnummern, das Senden von DTMF-Tönen (DTMF = *Dual-Tone Multifrequency*, hierzulande als *Tonwahlverfahren* be-

kannt) und das Hinzufügen eines Eintrages in das Telefonbuch des Gerätes. Ein beliebiger Content-Server kann diese Funktionen anbieten, da sie weder sicherheitsrelevante, private Daten behandeln noch unmittelbare Telefonkosten verursachen können.

- Die meisten WTA-Funktionen können vom WTA-Server eines Netzbetreibers innerhalb seines Netzes angeboten werden. Diese *einheitlichen Netzwerkdienste* (network common) gelten für *alle* Netzwerktypen (GSM, IS-136, ...) und umfassen neben dem Initiieren, Annehmen und Beenden von Gesprächen auch zahlreiche Funktionen zur Verwaltung von Textnachrichten und Telefonbucheinträgen sowie Zugriff auf Anrufprotokolle.

- Die *netzspezifischen Funktionen* (network specific) erweitern den Funktionsumfang um Dienste, die einem speziellen Netzwerktyp unterliegen. In WAP wurden die drei geläufigsten Mobilfunknetze GSM [47], IS-136 [48] und PDC [49] um weitere Funktionen bereichert. Ein Beispiel soll mit einer Funktion zum Versenden von USSD-Nachrichten bei GSM genannt werden (siehe auch USSD über GSM auf Seite 190).

20.3.1 WTA-Ereignisse

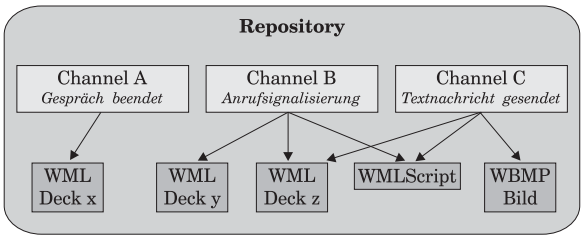
Die mobilen Teilnehmer und das Mobilfunknetz selbst lösen üblicherweise *Ereignisse* wie z.B. Textbotschaften, Anrufe oder Warnmeldungen in unregelmäßigen Zeitabständen aus. Es entstehen sogenannte *Ereignismeldungen* bei den betroffenen Instanzen. Tabelle 20.2 gibt einen Überblick über Ereignisse, die durch den WTA User Agent behandelt werden können. Der WTA User Agent in der WAE des Mobiltelefones kann nun auf eine individuelle Ereignisbehandlung angepasst werden, d.h. es können ausgesuchte WAE-Inhaltsformate als Reaktion auf bestimmte Ereignisse ausgegeben werden.

Eine bedeutende Komponente dieser Ereignisbehandlung ist das sogenannte *Repository*, ein für den WTA User Agent reservierter Speicher, mit seinem Konzept der *Channels*. Ein Channel bildet ein Netzereignis (z.B. »Textbotschaft erhalten«) auf eine Menge von Ressourcen (WML-Dokumente, WMLScripts, ...) ab (Abbildung 20.3). Hierbei können Ressourcen auch von mehreren Channels gleichzeitig belegt werden. Channels sind durch

Tabelle 20.2
WTA-Ereignisse

Ereignis	Identifikator
incoming call indication	cc/ic
call cleared	cc/cl
call connected	cc/co
outgoing call indication	cc/oc
connectiong call indication	cc/cc
DTMF sent	cc/dtmf
incoming network text indication	nt/it
network text send	nt/st
network status indication	ms/ns

Abbildung 20.3
Das Repository des WTA User Agent



ein XML-konformes Inhaltsformat beschrieben und werden durch die Medientypen

`text/vnd.wap.channel`

in lesbarer Textform und

`application/vnd.wap.channelc`

für Channels in Binary XML identifiziert. Das XML-Dokument in Abbildung 20.4 zeigt ein Beispiel (aus [WTA]) für eine solche Channel-Beschreibung. In diesem Fall verweist der Channel auf die Ressourcen innerhalb der `<resource>`-Tags, wenn das Ereignis »eingehender Anruf« (`EventID=wtaev-cc/ic`, siehe auch Tabelle 20.2) aufgezeigt wird.

Der WTA-Server des Netzbetreibers hat nun die Möglichkeit, Channels und die zugehörigen Ressourcen in einem reservierten

```
<?xml version="1.0" ?>
<!DOCTYPE channel PUBLIC
 "-//WAPFORUM//DTD CHANNEL 1.1//EN" "channel.dtd">
<channel
        maxspace="2048"
        base="http://wap.IhrNetzbetreiber.de"
        EventID="wtaev-cc/ic"
        success="success.wml"
        failure="failure.wml" >

  <title>Anrufselektierung</title>

  <abstract>
    Selektierung eingehender Anrufe
  </abstract>

  <resource href="willkommen.wml" />
  <resource href="start.wml" />
  <resource href="skript.wmls" />
</channel>
```

Abbildung 20.4
Definition eines Channels

Bereich, dem Repository, innerhalb der WAE im WAP-Client abzulegen. Bei einem auftretenden Ereignis kann nun der WTA User Agent des Clients prüfen, ob zu diesem Ereignis ein passender Channel mit Ressourcen im eigenen Repository vorhanden ist. Falls ja, werden die darin enthaltenen Ressourcen direkt ausgegeben, ansonsten ist eine Anforderung über das Netz erforderlich.

Eine Beispielanwendung kann sein, daß bei einem eingehenden Anruf ein WML-Dokument aus dem Repository geladen wird, welches die Optionen *annehmen, ablehnen, halten, weiterleiten* oder *auf Mailbox weiterleiten* anbietet und zudem noch das aktuelle Logo des Netzbetreibers einblendet. Der Vorteil liegt auf der Hand: Es müssen neben der Ereignismeldung keine weiteren Informationen über das Funknetz übertragen werden, da alle Ressourcen lokal abgerufen werden.

20.3.2 WTAI-Aufrufe

Wie können nun konkret WTAI-Funktionen in WML und WMLScript genutzt werden? Der allgemeine Aufruf einer WTAI-Funk-

tion aus einem WML-Dokument heraus geschieht über einen URI der Form

```
wtai://<Bibliothek>/<Funktion>;Parameter.
```

Das Codefragment in Abbildung 20.5 zeigt eine beispielhafte Verwendung der WTAI-Funktion *Nummer wählen* innerhalb eines WML-Dokumentes. Nachdem der Benutzer der Aufforderung gefolgt ist, eine Rufnummer einzugeben, wird mittels der lokalen Variable `N` eine URL mit Paramter (=Telefonnummer) erzeugt. Die URL wird nun durch einen `<go>`-Tag aufgerufen. Bei dieser URL ist nicht ein WML-Deck, sondern das WTAI das Ziel. Dort wird aus der Bibliothek öffentlicher Funktionen (`wp` steht für wtai public) die Funktion `mc` (make call) mit der zugehörigen Rufnummer aufgerufen.

Abbildung 20.5
WTAI-Aufruf in WML

```
<wml>
  <card>
    <do type="accept" label="start">
      <go URL="wtai://wp/mc;$(N)"/>
    </do>
    Bitte Telefonnummer eingeben:
    <input type="text" key="N"/>
  </card>
</wml>
```

Wie ein entsprechender Aufruf dieser WTA-Funktion in *WMLScript* aussieht, zeigt die Funktion in Abbildung 20.6. Nach Eingabe der Nummer wird die Funktion `rufeNummer` aufgerufen, um die gewünschte Telefonnummer anzuwählen. Zusätzlich kann hier nun durch `Lang.isInt` die eingegebene Nummer darauf getestet werden, ob es sich um ein gültiges Zahlenformat handelt. Wurden z.B. versehentlich Buchstaben in die Nummer eingefügt, so endet das Programm in einer Fehlermeldung. Ohne das Funknetz in Anspruch zu nehmen, wurde hier also schon eine echte Funktionalität realisiert. Es spricht aber noch mehr für die Verwendung von WTA durch WMLScript anstatt WML: zahlreiche WTA-Funktionen wie z.B. Telefonbuch-Funktionen und Verwaltung von Textbotschaften können *ausschließlich* über WMLScript aufgerufen werden. Genauer formuliert besitzt *jede* WTA-Funktion eine WMLScript-Schnittstelle; eine URI zum Aufruf per WML ist aber nur für einige dieser Funktionen definiert.

```
function rufeNummer(N) {
  if (Lang.isInt(N))
    WTAPublic.makeCall(N);
  else
    Dialog.alert("Nummernformat falsch !");
}
```

Abbildung 20.6 *WTAI-Aufruf in WMLScript*

20.4 Bilder im WBMP-Format

Auch bei Grafiken hat WAP sein Minimalprinzip durchgesetzt und ein eigenes Bildformat ins Leben gerufen: das Format *Wireless Bitmap (WBMP)* [40]. Dieses pixelorientierte Bildformat berücksichtigt Farbpaletten, Kompressionsverfahren und sogar Animationssequenzen. Bis zur WAP-Version 1.2 sind jedoch nur Bilder vom *Typ 0* genauer spezifiziert. Typ-0-Bilder werden im WSP/HTTP durch den MIME-Typ

```
image/vnd.wap.wbmp; level=0
```

identifiziert. Dieses sehr schlanke und einfache Bildformat beschreibt zweifarbige Bilder (genauer gesagt: schwarz/weiß), die in einem Format ohne Palette und Datenkompression vorliegen. In späteren WAP-Versionen können an dieser Stelle weitere Typen ergänzt werden, die z.B. im GIF- oder PNG-Format vorliegen. Auf diese Art kann sich das WAP-Protokoll an zukünftige Verbesserungen der Geräte anpassen, ohne eine vorhandene Spezifikation revidieren zu müssen. Für eine Abwärtskompatibilität wurde also gesorgt.
Das Typ-0-Format ist angenehm einfach gehalten und kann daher in kompakter Ausführung genauer betrachtet werden. Eine WBMP-Datei teilt sich in einen Dateikopf und in einen Datenteil auf. Der Dateikopf besteht aus

- einem Byte zur Typenkennung,
- einem Byte, das Informationen über erweiterte Dateikopf-Einträge gibt,
- optional aus weiteren Dateikopf-Einträge, sofern im vorhergehenden Byte »angekündigt«,
- der Breite das Bildes in Pixel
- und der Höhe des Bildes in Pixel.

Abbildung 20.7
Das WBMP-Format

Bild (13 x 8 Pixel):

Bit-Matrix:

0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0
0	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0
0	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0
0	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0
0	0	0	0	0	1	1	1	0	1	1	1	1	0	0	0

Datei (hexadezimal):

0, 0, D, 8, 70, 0, 77, 70, 77, 70, 77, 70, 50, 0, 57, 78, 57, 78, 07, 78

Kopf — Bilddaten

Die Bilddaten des Typ 0 werden durch eine Bit-Matrix repräsentiert. Bild 20.7 zeigt ein Beispiel eines WBMP-Bildes. Die Linien im oberen Bild sind in der Bit-Matrix als Spuren von Nullen wiederzufinden. Gesetzte Pixel werden durch eine Null, ungesetzte Pixel durch eine Eins repräsentiert. Da nur ganze Bytes verarbeitet werden können, ist das Bild durch Anfügen dreier zusätzlicher Spalten am rechten Rand, allesamt mit Nullen belegt, auf 16 x 8 Pixel erweitert worden. Nun werden Zeile um Zeile jeweils acht Bit zu einem Byte zusammengefaßt. Das erste, zweite und neunte Byte wurde exemplarisch in der Matrix und im Hexadezimal-Code durch eine Umrandung hervorgehoben, um die Beziehung zwischen Bit-Matrix und Code zu verdeutlichen.

Daß das Typ-0-Format sehr effizient ist, zeigt der Vergleich mit anderen Formaten: während das WBMP-Bild im Beispiel genau 20 Byte belegt, kostet das gleiche Motiv im (minimalen) GIF-Format schon 54 Byte und im Windows-BMP-Format sogar mit 94 Byte schon mehr als das Vierfache. Neben der Bereitstellung von WBMP-Dateien gibt es auch die Möglichkeit, mächtigere Formate wie z.B. GIF-Dateien zur Verfügung zu stellen und die Konver-

tierung in ein WAP-konformes Format durch das WAP-Gateway durchführen zu lassen. Hierdurch muß bei einer Adaption der Bildformate in der WAP-Spezifikation nicht jeder Server, sondern nur die Gateway-Software angepasst werden.

Wie Sie am einfachsten – auch ohne Bitmanipulation – an Ihr WBMP-Bild herankommen, können Sie übrigens in Abschnitt 6.2 nachlesen. Der darauffolgende Abschnitt beschreibt Ihnen dann auch ein Verfahren, um vorhandene Bilder möglichst brauchbar in das WBMP-Format zu überführen.

20.5 Visitenkarten und Kalender

Vom Versit-Konsortium wurden Mitte der 90er Jahre standardisierte Formate für elektronische Visitenkarten und Terminkalender festgelegt. Ein einheitliches Format für diese immer wiederkehrende Art der Datenhaltung sollte verschiedensten Endgeräten wie z.B. PDA, Telefon, Mobiltelefon oder Anwendungssoftware die Möglichkeit geben, diese Information untereinander auszutauschen. Die Formate *vCard* als elektronische Visitenkarte und *vCalendar* zur Terminverwaltung haben sich heute als Industriestandards durchgesetzt und werden von WAP ebenfalls unterstützt. Die assoziierten MIME-Medientypen lauten

```
text/x-vcard
```

für die elektronische Visitenkarte und

```
text/x-vCalendar
```

für das Kalenderformat.

Die heute noch meist proprietären Implementierungen der Telefonbücher und Terminkalender in Mobiltelefonen sollen im WAP-Zeitalter diesen Standards weichen.

Die elektronische Visitenkarte vCard [13] ist ein heute schon sehr beliebtes Anhängsel bei E-mail-Nachrichten und breitet sich insbesondere in mobilen Kleingeräten immer weiter aus. Eine vCard ist ein standardisierter Datensatz zur Haltung persönlicher Kontaktdaten wie Anschriften, Fax- und Telefonnummern, E-mail-Adressen und URLs. Auch multimediale Elemente wie Fotos oder Audiodaten können mit einer vCard übermittelt werden, hierbei sind jedoch die Einschränkungen der verwendeten Geräteplattformen zu berücksichtigen. Die vCard-Spezifikation stützt sich auf bzw. erweitert vorhandene Standards wie CCITT X.500 (Objekttyp »Person«) und ISO 8601 (Zeit-/Datumsformat). Ein oft

eingesetzter Weg zum drahtlosen Austausch einer vCard ist die Infrarotübertragung.

Die Spezifikation des vCalendar [12] beschreibt ein einheitliches Format für Termine und Terminkalender. Die *Verarbeitung* solcher Daten wurde nicht festgelegt und bleibt den Implementierungen vorbehalten. Standardisierte »Terminobjekte« sollen bequem aus elektronischen Quellen wie PDA-Schnittstellen, E-mail-Nachrichten und Webseiten in die eigene Terminverwaltung übernommen werden können. Mobiltelefone benutzen, wenn überhaupt vorhanden, heute meist noch herstellereigene Formate zur Speicherung von Termindaten. Mit steigender Verarbeitungsleistung dieser Kleingeräte wird in absehbarer Zeit eine Terminverwaltung zur »Serienausstattung« dazugehören und somit die Bedeutung eines standardisierten Formates wie vCalendar erhöhen. Auch vCalendar ist, ebenso wie vCard, eine öffentliche Spezifikation und stützt sich auf vorhandene Standards.

21 Ausblick

Dieses letzte Kapitel sei all den Kritikern gewidmet, die mit überzeugten Mienen das WAP unmittelbar nach seiner Geburt zu Grabe getragen haben. Jenen Stimmen, die dem WAP als unpraktikable und am Markt deplazierte Technologie keinerlei Chancen eingeräumt haben. Liebe Pessimisten, es gibt ihn doch: den WAP-Nutzer.

Zugegeben, bei den heutigen Tarifen bleiben die meisten WAP-Microbrowser in Deutschland noch ausgeschaltet. Doch wird bald nicht nur der natürliche Wettbewerb unter den Telefongesellschaften einen WAP-Zugang erheblich billiger gestalten. Durch neue, paketorientierte Mobilfunknetze entstehen neue Abrechnungsmodelle, die eine WAP-Nutzung quasi »pro Byte« in Rechnung stellen. Die WAP-Nutzer sind dann rund um die Uhr online, was wiederum neuen Diensten (z.B. Push-Services) die Türen öffnet.

Wie es um die Situation der Mobilfunknetze in Deutschland und deren Zukunft bestellt ist, haben Sie bereits in Kapitel 17 erfahren: momentan wird das heutige GSM auf GPRS aufgerüstet, so daß bis Ende 2001 mit einer breiten Markteinführung dieses Netzes zu rechnen ist. Bis etwa zum Jahr 2005 können sich dann die Teilnehmer langsam an einen paketorientierten Dienst und an eine höhere Bandbreite von etwa 100 kbit/s gewöhnen, um dann mit UMTS eine weitere, erhebliche Steigerung der Übertragungsgeschwindigkeit bis auf 2 Mbit/s zu erfahren.

Der aufmerksame Leser kombiniert nun die Prognose der stetig anschwellenden Bandbreite und die immer noch anhaltenden Leistungssteigerungen in der Mikroprozessortechnologie mit der auf Seite 197 dargestellten Entwicklungsmotivation für WAP und folgert haarscharf: wenn sich die genannten Restriktionen immer weiter auflösen, so verliert doch WAP immer mehr seine Existenzberechtigung. Tatsächlich ist WAP in seiner heutigen Form eine, wenn auch etwas länger andauernde *Übergangstechnologie*. Die Grenze zwischen Web und WAP liegt heute schon bei den PDAs, auf denen sowohl WAP-Browser als auch normale Web-Browser

(unter Windows CE) installiert sein können. WAP wird sich daher auch zukünftig nur auf die kleinsten aller mobilen Endgeräte beschränken: die Mobiltelefone und Pager mit ihren Mini-Displays. Das Protokoll wird sich laufend dem technischen Fortschritt anpassen müssen. Während die Benutzerschnittstelle noch längerfristig eine Herausforderung darstellen wird, können sich mit leistungsfähigeren Geräten und schnellen Funknetzen neue Inhaltsobjekte in der WAE ansiedeln. Im einfachsten Fall kann dies eine Erweiterung von WML um z.B. Farbattribute, um neue Typen von WBMP-Bildern oder um erweiterte WMLScript-Bibliotheken sein; in ferner Zukunft sind aber auch umfangreichere Typen wie Video-Objekte oder Java-Applets denkbar.

Das WAP-Forum macht sich zum heutigen Zeitpunkt jedoch noch nicht so viele Gedanken um neue Inhaltsformate, sondern stopft Löcher in der aktuellen Spezifikation und reorganisiert die Versionshaltung und Benennung der Einzeldokumente (es ist daher möglich, daß Sie eine im angehängten Literaturverzeichnis aufgeführte Spezifikation beim WAP-Forum unter anderem Namen wiederfinden). Dessen aktueller Schwerpunkt bei den Neuentwicklungen liegt im Bereich Sicherheit. Eine Spezifikation für eine durchgehende Verschlüsselung über das Gateway hinweg, eine Public-Key-Infrastruktur und ein Zertifikatsmanagement liegen zur Verabschiedung für WAP 1.4 bereit. Ebenfalls zur Debatte steht die sogenannte *Provisioning Architecture*, die eine automatische Konfiguration der Endgeräte *über das Netz* ermöglicht, indem es dieses mit relevanten Informationen über das zugrunde liegende Netzwerk versorgt. Es bleibt zu hoffen, daß die Dynamik des WAP-Forums nicht durch die steigende Mitgliederzahl (davon stehen viele in direkter Konkurrenz am Markt!) auf der Strecke bleibt und dessen Energie in internen Diskussionen verpufft. Was sprichwörtlich »zu viele Köche« anrichten können, ist Ihnen sicherlich geläufig.

Welche Alternativen gibt es zu WAP und WML? In Japan und Korea wächst der mobile Zugriff aufs Internet rapide schnell an. Nicht jedoch WAP bringt die Asiaten ins Netz, sondern ein Dienst des Internetproviders *NTT DoCoMo* namens *i-Mode*. Der Dienst startete Anfag 1999. In einem Blitzwachstum stiegen die Benutzerzahlen in den ersten neun Monaten des Jahres 2000 von 3,1 Millionen auf über 11 Millionen an – Tendenz weiter steigend. Das Netz arbeitet, vergleichbar mit GPRS, paketorientiert und die Abrechnung von i-Mode erfolgt volumenbasiert, also pro übertragenem Datenpaket. Die Benutzer sind daher permanent onli-

ne. Der Dienst benutzt mit *cHTML (Compact HTML)* ([21]) eine HTML-Variante, die mit speziellen Tags für die Darstellung auf tragbaren Endgeräten ausgestattet ist. Der gewaltige Erfolg von i-Mode in Asien wird wohl auch in Kürze eine Markteinführung in Europa und den USA sehr wahrscheinlich werden lassen.

Neben WML gibt es noch weitere Bestrebungen, eine schlanke Auszeichnungssprache für mobile Bedürfnisse zu entwicklen. Mit *XHTML Basic* [26] spezifiziert das W3C eine weitere Sprache, die eine Untermenge von XHTML darstellt. Die Auszeichungssprache verzichtet auf zahlreiche Features seines »großen Bruders« XHTML, trumpft dafür jedoch mit einer Erweiterbarkeit durch XHTML-Module auf. Am Markt spielt XHTML Basic derzeit jedoch keine Rolle.

Eine weitere Alternative zu WML ist seine eigene Vorgängerin, die *Handheld Device Markup Language (HDML)* ([15]). Die in der Version 2 vorliegende Auszeichnungssprache wird sich jedoch wohl kaum noch als marktbeherrschender Standard etablieren, da die fehlende Konformität zu XML einen gravierenden Nachteil darstellt.

Es bleibt die Frage, ob WAP sich durchsetzen wird. Momentan stehen alle Zeichen in Europa auf WAP, woran sich in den nächsten Monaten auch kaum etwas ändern dürfte. Derzeit wird etwa jedes zweite in den Handel kommende Mobiltelefon mit einem WAP-Microbrowser ausgeliefert und eine echte Ablösetechnologie liegt augenscheinlich nicht in greifbarer Nähe. Ob WAP sich durchsetzen wird, darf momentan nicht zur Debatte stehen, denn Tatsache ist: WAP ist das einzige, was wir *heute* haben. Dem mobilen, drahtlosen Internetzugriff wird die Zukunft gehören. Die Richtung im Sinne eines zukunftsorientierten Kommunikationsverhalten stimmt jedenfalls. Es ist zwar absehbar, daß WAP sich den besseren Netzen bis zur Unkenntlichkeit anpassen oder komplett einer Nachfolgetechnologie weichen muß, für mindestens die nächsten fünf Jahre wird das Wireless Application Protocol jedoch noch unser ständiger Begleiter bleiben.

A Übersicht der WML-Tags

A.1 Struktur

`<?xml>`	Markierung als XML-Dokument
`<!DOCTYPE>`	Dokumententyp
`<wml>`	Markierung als WML-Dokument Attribute: `class id xml:lang`
`<head>`	Dokumenteninformationen Attribute: `class id`
`<card>`	Karte Attribute: `class id newcontext onenterbackward onenterforward ontimer ordered title xml:lang`
`<meta>`	Metainformationen Attribute: `class content forua http-equiv id name schema`
`<access>`	Zugriffsregelung Attribute: `class domain id path`

A.2 Inhaltsauszeichnung

`<i>`	Kursivschrift `<i>kursiv</i>` ergibt *kursiv*
`<b>`	Fettschrift `<b>fett</b>` ergibt **fett**
`<u>`	Unterstreichung `<u>unterstrichen</u>` ergibt <u>unterstrichen</u>
`<em>`	hervorgehoben (von »emphasis«) `<em>wichtig</em>` ergibt *wichtig*
`<strong>`	hervorgehoben `<strong>wichtigst</strong>` ergibt ***wichtigst***
`<big>`	Größere Schrift `<big>größer</big>` ergibt größer

<small>	Kleinere Schrift `<small>kleiner</small>` ergibt kleiner
<p>	Absatz Attribute: `align mode`
** **	Zeilenumbruch
<a>	Quellanker eines Links Attribute: `accesskey class href id title xml:lang`
<anchor>	Quellanker eines Links mit Aktion Attribute: `accesskey class id title xml:lang`
<!-- -->	Kommentar
<![CDATA[ ]]>	Uninterpretierter Text
<pre>	Vorformatierter Text Attribute: `class id`
<img>	Bild Attribute: `align alt class height hspace id localsrc src vspace width xml:lang`

A.3 Tabellen

<table>	Tabelle Attribute: `align class columns id title xml:lang`
<tr>	Tabellenzeile Attribute: `class id`
<td>	Tabellenzelle Attribute: `class id xml:lang`

A.4 Formulare

<select>	Auswahlliste Attribute: `class id iname ivalue multiple name tabindex title value xml:lang`
<option>	Auswahlpunkt Attribute: `class id onpick title value xml:lang`
<optgroup>	Gruppe von Auswahlpunkten Attribute: `class id title xml:lang`

<**input**>	Eingabefeld Attribute: `accesskey class emptyok format id maxlength name size tabindex title type value xml:lang`
<**fieldset**>	Gruppe von Eingabefeldern Attribute: `class id title xml:lang`
<**postfield**>	Datenfeld zur Übermittlung an Server Attribute: `class id name value`

A.5 Tasks

<**go**>	Navigation zu einer Karte Attribute: `accept-charset class enctype href id method sendreferer`
<**refresh**>	Neudarstellen der aktuellen Karte Attribute: `class id`
<**prev**>	Navigation zur vorher dargestellten Karte Attribute: `class id`
<**noop**>	Leere Aktion Attribute: `class id`

A.6 Ereignisse und Variable

<**do**>	Bindung von Aktion an Steuerelement Attribute: `class id label name optional type xml:lang`
<**onevent**>	Aktion an Ereignis binden Attribute: `class id type`
<**timer**>	Stoppuhr Attribute: `class id name value`
<**template**>	Schablone für Aktionsbindungen Attribute: `class id onenterbackward onenterforward ontimer`
<**setvar**>	Variable setzen Attribute: `class id name value`

B Symbolische Werte von Attributen

align

`bottom`	Unten ausrichten
`C`	Tabellenzelle zentriert ausrichten
`center`	Zentriert ausrichten
`D`	Tabellenzelle nach Standardausrichtung
`L`	Tabellenzelle links ausrichten
`left`	Links ausrichten
`middle`	Mittig ausrichten
`R`	Tabellenzelle rechts ausrichten
`right`	Rechts ausrichten
`top`	Oben ausrichten

enctype

`application/x-www-form-urlencoded`	Formulardaten in URL codieren
`multipart/form-data`	Formulardaten in Mitteilung schicken

forua

`false`	Protokollzeilen nicht zum Endgerät senden
`true`	Protokollzeilen zum Endgerät senden

method

`get`	HTTP-Method GET zum Seitenzugriff
`post`	HTTP-Method POST zum Seitenzugriff

mode

`nowrap`	Automatischer Zeilenumbruch aus
`wrap`	Automatischer Zeilenumbruch an

multiple

`false`	Nur Einfachauswahl möglich

`true`	Mehrfachauswahl möglich

`newcontext`

`false`	Variableninhalte beibehalte
`true`	Variableninhalte löschen

`optional`

`false`	Steuerelement muß angeboten werden
`true`	Steuerelement kann unterdrückt werden

`ordered`

`false`	Feldreihenfolge veränderbar
`true`	Feldreihenfolge verbindlich

`sendreferer`

`false`	Adresse der Verweisquelle nicht mitsenden
`true`	Adresse der Verweisquelle mitsenden

`type`

`accept`	Beschriftung der Annahmetaste festlegen
`delete`	Taste Löschen gedrückt
`help`	Taste Hilfeanforderung gedrückt
`onenterbackward`	Ereignis Betreten einer Karte
`onenterforward`	Ereignis Verlassen einer Karte
`onpick`	Ereignis Auswahl eines Auswahlelements
`ontimer`	Ereignis Ablauf der Stoppuhr
`options`	Taste Auswahlerklärung gedrückt
`password`	Eingabefeld soll unlesbar bleiben
`prev`	Taste Zurück gedrückt
`reset`	Taste Zurücksetzen gedrückt
`text`	Eingabefeld ist lesbarer Text
`unknown`	Unbekanntes Ereignis
`vnd.*`	Herstellerspezifisches Ereignis

C Internationale Sprachcodes

WML verwendet an verschiedenen Stellen die im ISO Standard 639 definierten Kürzel für Sprachen. Diese sind:

Code	Sprache	Code	Sprache	Code	Sprache
aa	Afar	hy	Armenisch	rm	Rätoromanisch
ab	Abchasisch	ia	Interlingua	rn	Kirundisch
af	Afrikaans	id (in)	Indonesisch	ro	Rumänisch
am	Amharisch	ie	Interlingue	ru	Russisch
ar	Arabisch	ik	Inupiak	rw	Kijarwanda
as	Assamesisch	is	Isländisch	sa	Sanskrit
ay	Aymara	it	Italienisch	sd	Zinti
az	Aserbaidschanisch	ja	Japanisch	sg	Sango
ba	Baschkirisch	jw	Javanisch	sh	Serbokroatisch
be	Belorussisch	ka	Georgisch	si	Singhalesisch
bg	Bulgarisch	kk	Kasachisch	sk	Slowakisch
bh	Biharisch	kl	Grönländisch	sl	Slowenisch
bi	Bislamisch	km	Kambodschanisch	sm	Samoanisch
bn	Bengalisch	kn	Kannada	sn	Schonisch
bo	Tibetanisch	ko	Koreanisch	so	Somalisch
br	Bretonisch	ks	Kaschmirisch	sq	Albanisch
ca	Katalanisch	ku	Kurdisch	sr	Serbisch
co	Korsisch	ky	Kirgisisch	ss	Swasiländisch
cs	Tschechisch	la	Lateinisch	st	Sesothisch
cy	Walisisch	ln	Lingalisch	su	Sudanesisch
da	Dänisch	lo	Laotisch	sv	Schwedisch
de	Deutsch	lt	Litauisch	sw	Suaheli
dz	Bhutani	lv	Lettisch	ta	Tamilisch
el	Griechisch	mg	Malagasisch	te	Telugu
en	Englisch	mi	Maorisch	tg	Tadschikisch
eo	Esperanto	mk	Mazedonisch	th	Thai
es	Spanisch	ml	Malajalam	ti	Tigrinja
et	Estnisch	mn	Mongolisch	tk	Turkmenisch
eu	Baskisch	mo	Moldavisch	tl	Tagalog
fa	Persisch	mr	Marathi	tn	Sezuan
fi	Finnisch	ms	Malaysisch	to	Tongaisch
fj	Fiji	mt	Maltesisch	tr	Türkisch
fo	Faröisch	my	Burmesisch	ts	Tsongaisch
fr	Französisch	na	Nauruisch	tt	Tatarisch
fy	Friesisch	ne	Nepalisch	tw	Twi
ga	Irisch	nl	Holländisch	uk	Ukrainisch
gd	Schott. Gälisch	no	Norwegisch	ur	Urdu
gl	Galizisch	oc	Okzitanisch	uz	Usbekisch
gn	Guarani	om	Oromo	vi	Vietnamesisch
gu	Gujaratisch	or	Orija	vo	Volapük
ha	Haussa	pa	Pundjabisch	wo	Wolof
he (iw)	Hebräisch	pl	Polnisch	xh	Xhosa
hi	Hindi	ps	Paschtu	yi (ji)	Jiddish
hr	Kroatisch	pt	Portugiesisch	yo	Joruba
hu	Ungarisch	qu	Quechua	zh	Chinesisch
hy	Armenisch	rm	Rätoromanisch	zu	Zulu

Sprache	Code	Sprache	Code	Sprache	Code
Abchasisch	ab	Irisch	ga	Rumänisch	ro
Afar	aa	Isländisch	is	Russisch	ru
Afrikaans	af	Italienisch	it	Rätoromanisch	rm
Albanisch	sq	Japanisch	ja	Samoanisch	sm
Amharisch	am	Javanisch	jw	Sango	sg
Arabisch	ar	Jiddish	yi (ji)	Sanskrit	sa
Armenisch	hy	Joruba	yo	Schonisch	sn
Aserbaidschanisch	az	Kambodschanisch	km	Schott. Gälisch	gd
Assamesisch	as	Kannada	kn	Schwedisch	sv
Aymara	ay	Kasachisch	kk	Serbisch	sr
Baschkirisch	ba	Kaschmirisch	ks	Serbokroatisch	sh
Baskisch	eu	Katalanisch	ca	Sesothisch	st
Belorussisch	be	Kijarwanda	rw	Sezuan	tn
Bengalisch	bn	Kirgisisch	ky	Singhalesisch	si
Bhutani	dz	Kirundisch	rn	Slowakisch	sk
Biharisch	bh	Koreanisch	ko	Slowenisch	sl
Bislamisch	bi	Korsisch	co	Somalisch	so
Bretonisch	br	Kroatisch	hr	Spanisch	es
Bulgarisch	bg	Kurdisch	ku	Suaheli	sw
Burmesisch	my	Laotisch	lo	Sudanesisch	su
Chinesisch	zh	Lateinisch	la	Swasiländisch	ss
Deutsch	de	Lettisch	lv	Tadschikisch	tg
Dänisch	da	Lingalisch	ln	Tagalog	tl
Englisch	en	Litauisch	lt	Tamilisch	ta
Esperanto	eo	Malagasisch	mg	Tatarisch	tt
Estnisch	et	Malajalam	ml	Telugu	te
Faröisch	fo	Malaysisch	ms	Thai	th
Fiji	fj	Maltesisch	mt	Tibetanisch	bo
Finnisch	fi	Maorisch	mi	Tigrinja	ti
Französisch	fr	Marathi	mr	Tongaisch	to
Friesisch	fy	Mazedonisch	mk	Tschechisch	cs
Galizisch	gl	Moldavisch	mo	Tsongaisch	ts
Georgisch	ka	Mongolisch	mn	Turkmenisch	tk
Griechisch	el	Nauruisch	na	Twi	tw
Grönländisch	kl	Nepalisch	ne	Türkisch	tr
Guarani	gn	Norwegisch	no	Ukrainisch	uk
Gujaratisch	gu	Okzitanisch	oc	Ungarisch	hu
Haussa	ha	Orija	or	Urdu	ur
Hebräisch	he (iw)	Oromo	om	Usbekisch	uz
Hindi	hi	Paschtu	ps	Vietnamesisch	vi
Holländisch	nl	Persisch	fa	Volapük	vo
Indonesisch	id (in)	Polnisch	pl	Walisisch	cy
Interlingua	ia	Portugiesisch	pt	Wolof	wo
Interlingue	ie	Pundjabisch	pa	Xhosa	xh
Inupiak	ik	Quechua	qu	Zinti	sd
Irisch	ga	Rumänisch	ro	Zulu	zu

D Document Type Definitions

D.1 WML DTD

```
<!--
Wireless Markup Language (WML) Document Type Definition.
WML is an XML language. Typical usage:
   <?xml version="1.0"?>
   <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
   "http://www.wapforum.org/DTD/wml12.dtd">
   <wml>
   ...
   </wml>

   Terms and conditions of use are available from the
   Wireless Application Protocol Forum
   Ltd. web site at http://www.wapforum.org/docs/copyright.htm.

-->


<!ENTITY % length  "CDATA">       <!-- [0-9]+ for pixels or [0-9]+"%" for
                                       percentage length -->
<!ENTITY % vdata   "CDATA">       <!-- attribute value possibly
                                       containing variable references -->
<!ENTITY % HREF    "%vdata;">     <!-- URI, URL or URN designating a
                                       hypertext node. May contain
                                       variable references -->
<!ENTITY % boolean "(true|false)">
<!ENTITY % number  "NMTOKEN">       <!-- a number, with format [0-9]+ -->
<!ENTITY % coreattrs   "id     ID     #IMPLIED
                        class  CDATA  #IMPLIED">
<!ENTITY % ContentType "%vdata;">   <!-- media type. May contain variable
                                          references -->

<!ENTITY % emph     "em | strong |b |i |u |big |small">
<!ENTITY % layout   "br">
```

```
<!ENTITY % text     "#PCDATA | %emph;">

<!-- flow covers "card-level" elements, such as text and images -->
<!ENTITY % flow     "%text; | %layout; | img | anchor |a |table">

<!-- Task types -->
<!ENTITY % task "go | prev | noop | refresh">

<!-- Navigation and event elements -->
<!ENTITY % navelmts "do | onevent">

<!--================ Decks and Cards =================-->

<!ELEMENT wml ( head?, template?, card+ )>
<!ATTLIST wml
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!-- card intrinsic events -->
<!ENTITY % cardev
  "onenterforward  %HREF;      #IMPLIED
  onenterbackward  %HREF;      #IMPLIED
  ontimer          %HREF;      #IMPLIED"
>

<!-- card field types -->
<!ENTITY % fields "%flow; | input | select | fieldset">
<!ELEMENT card (onevent*, timer?, (do | p | pre)*)>
<!ATTLIST card
  title            %vdata;     #IMPLIED
  newcontext       %boolean;   "false"
  ordered          %boolean;   "true"
  xml:lang         NMTOKEN     #IMPLIED
  %cardev;
  %coreattrs;
>

<!--================ Event Bindings =================-->

<!ELEMENT do (%task;)>
<!ATTLIST do
  type             CDATA       #REQUIRED
  label            %vdata;     #IMPLIED
  name             NMTOKEN     #IMPLIED
  optional         %boolean;   "false"
```

```
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT onevent (%task;)>
<!ATTLIST onevent
  type             CDATA       #REQUIRED
  %coreattrs;
>

<!--================ Deck-level declarations =================-->

<!ELEMENT head ( access | meta )+>
<!ATTLIST head
  %coreattrs;
>

<!ELEMENT template (%navelmts;)*>
<!ATTLIST template
  %cardev;
  %coreattrs;
>

<!ELEMENT access EMPTY>
<!ATTLIST access
  domain           CDATA       #IMPLIED
  path             CDATA       #IMPLIED
  %coreattrs;
>

<!ELEMENT meta EMPTY>
<!ATTLIST meta
  http-equiv       CDATA       #IMPLIED
  name             CDATA       #IMPLIED
  forua            %boolean;   "false"
  content          CDATA       #REQUIRED
  scheme           CDATA       #IMPLIED
  %coreattrs;
>

<!--================ Tasks =================-->

<!ELEMENT go (postfield | setvar)*>
<!ATTLIST go
  href             %HREF;          #REQUIRED
  sendreferer      %boolean;       "false"
```

```
  method           (post|get)      "get"
  enctype          %ContentType;   "application/x-www-form-urlencoded"
  accept-charset   CDATA           #IMPLIED
  %coreattrs;
>

<!ELEMENT prev (setvar)*>
<!ATTLIST prev
  %coreattrs;
>

<!ELEMENT refresh (setvar)*>
<!ATTLIST refresh
  %coreattrs;
>

<!ELEMENT noop EMPTY>
<!ATTLIST noop
  %coreattrs;
>

<!--================ postfield ================-->

<!ELEMENT postfield EMPTY>
<!ATTLIST postfield
  name             %vdata;     #REQUIRED
  value            %vdata;     #REQUIRED
  %coreattrs;
>

<!--================ variables ================-->

<!ELEMENT setvar EMPTY>
<!ATTLIST setvar
  name             %vdata;     #REQUIRED
  value            %vdata;     #REQUIRED
  %coreattrs;
>

<!--================ Card Fields ================-->

<!ELEMENT select (optgroup|option)+>
<!ATTLIST select
  title            %vdata;     #IMPLIED
  name             NMTOKEN     #IMPLIED
  value            %vdata;     #IMPLIED
```

```
  iname            NMTOKEN     #IMPLIED
  ivalue           %vdata;     #IMPLIED
  multiple         %boolean;   "false"
  tabindex         %number;    #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT optgroup (optgroup|option)+ >
<!ATTLIST optgroup
  title            %vdata;     #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT option (#PCDATA | onevent)*>
<!ATTLIST option
  value            %vdata;     #IMPLIED
  title            %vdata;     #IMPLIED
  onpick           %HREF;      #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT input EMPTY>
<!ATTLIST input
  name             NMTOKEN         #REQUIRED
  type             (text|password) "text"
  value            %vdata;         #IMPLIED
  format           CDATA           #IMPLIED
  emptyok          %boolean;       "false"
  size             %number;        #IMPLIED
  maxlength        %number;        #IMPLIED
  tabindex         %number;        #IMPLIED
  title            %vdata;         #IMPLIED
  accesskey        %vdata;         #IMPLIED
  xml:lang         NMTOKEN         #IMPLIED
  %coreattrs;
>

<!ELEMENT fieldset (%fields; | do)* >
<!ATTLIST fieldset
  title           %vdata;      #IMPLIED
  xml:lang        NMTOKEN      #IMPLIED
  %coreattrs;
>
```

```
<!ELEMENT timer EMPTY>
<!ATTLIST timer
  name             NMTOKEN     #IMPLIED
  value            %vdata;     #REQUIRED
  %coreattrs;
>

<!-----------------= Images =----------------->

<!ENTITY % IAlign "(top|middle|bottom)" >
<!ELEMENT img EMPTY>
<!ATTLIST img
  alt              %vdata;     #REQUIRED
  src              %HREF;      #REQUIRED
  localsrc         %vdata;     #IMPLIED
  vspace           %length;    "0"
  hspace           %length;    "0"
  align            %IAlign;    "bottom"
  height           %length;    #IMPLIED
  width            %length;    #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!----------------= Anchor =----------------->

<!ELEMENT anchor ( #PCDATA | br | img | go | prev | refresh )*>
<!ATTLIST anchor
  title            %vdata;     #IMPLIED
  accesskey        %vdata;     #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>
<!ELEMENT a ( #PCDATA | br | img )*>
<!ATTLIST a
  href             %HREF;      #REQUIRED
  title            %vdata;     #IMPLIED
  accesskey        %vdata;     #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!-----------------= Tables =----------------->

<!ELEMENT table (tr)+>
```

```
<!ATTLIST table
  title            %vdata;     #IMPLIED
  align            CDATA       #IMPLIED
  columns          %number;    #REQUIRED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT tr (td)+>
<!ATTLIST tr
  %coreattrs;
>
<!ELEMENT td ( %text; | %layout; | img | anchor |a )*>
<!ATTLIST td
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!--================= Text layout and line breaks =================-->

<!ELEMENT em (%flow;)*>
<!ATTLIST em
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT strong (%flow;)*>
<!ATTLIST strong
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT b (%flow;)*>
<!ATTLIST b
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT i (%flow;)*>
<!ATTLIST i
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT u (%flow;)*>
<!ATTLIST u
```

```
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT big (%flow;)*>
<!ATTLIST big
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT small (%flow;)*>
<!ATTLIST small
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ENTITY % TAlign    "(left|right|center)">
<!ENTITY % WrapMode  "(wrap|nowrap)" >

<!ELEMENT p (%fields; | do)*>
<!ATTLIST p
  align            %TAlign;    "left"
  mode             %WrapMode;  #IMPLIED
  xml:lang         NMTOKEN     #IMPLIED
  %coreattrs;
>

<!ELEMENT br EMPTY>
<!ATTLIST br
  %coreattrs;
>

<!ELEMENT pre (#PCDATA | a | br | i | b | em | strong | input | select )*>
<!ATTLIST pre
  xml:space        CDATA       #FIXED    "preserve"
  %coreattrs;
>

<!ENTITY quot "&#34;">       <!-- quotation mark -->
<!ENTITY amp  "&#38;#38;">   <!-- ampersand -->
<!ENTITY apos "&#39;">       <!-- apostrophe -->
<!ENTITY lt   "&#38;#60;">   <!-- less than -->
<!ENTITY gt   "&#62;">       <!-- greater than -->
<!ENTITY nbsp " ">      <!-- non-breaking space -->
<!ENTITY shy  "&#173;">      <!-- soft hyphen (discretionary hyphen) -->
```

D.2 Channel DTD

```
<!--
  This DTD is identified by the PUBLIC identifier:
  "-//WAPFORUM//DTD CHANNEL 1.1//EN"
-->

<!-- a Uniform Resource Identifier -->
<!ENTITY % URI "CDATA" >

<!-- one or more digits (NUMBER) -->
<!ENTITY % Number "CDATA" >

<!-- Channel Events -->
<!ENTITY % ChannelEvent.attribs
 "success %URI; #REQUIRED
  failure %URI; #IMPLIED"
>

<!ELEMENT channel (title , abstract? , resource* ) >
<!ATTLIST channel
  maxspace %Number; #REQUIRED
  base %URI; #IMPLIED
  EventId CDATA #REQUIRED
 %ChannelEvent.attribs;
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>

<!ELEMENT resource EMPTY>
<!ATTLIST resource
  href %URI; #REQUIRED
  lastmod %Number; #IMPLIED
  etag NMTOKEN #IMPLIED
  md5 NMTOKEN #IMPLIED
>
```

Literaturverzeichnis

[1] ISO/IEC 10646-1. *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*, 1993.

[2] ISO/IEC 8859-1. *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*, 1998.

[3] T. Berners-Lee, R. Fielding, und H. Frystyk. *RFC 1945: Hypertext Transfer Protocol — HTTP/1.0*, May 1996.

[4] T. Berners-Lee, R. Fielding, und L. Masinter. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*, August 1998.

[5] ECMA. *ECMAScript Language Specification*. June 1997. http://www.ecma.ch.

[6] ETSI European Digital Cellular Telecommunication Systems, phase 2+. *Technical Realisation of the Short Message Service (SMS) Point-to-Point (P)*. GSM 03.40. http://www.etsi.ch.

[7] ETSI European Digital Cellular Telecommunication Systems, phase 2+. *Technical Realisation of the Short Message Service Cell Broadcast (SMSCB)*. GSM 03.41. http://www.etsi.ch.

[8] ETSI European Digital Cellular Telecommunication Systems, phase 2. *Unstructured Supplementary Service Data (USSD) - stage 1*. GSM 02.90. http://www.etsi.ch.

[9] ETSI European Digital Cellular Telecommunication Systems, phase 2. *Unstructured Supplementary Service Data (USSD) - stage 2*. GSM 03.90. http://www.etsi.ch.

[10] ETSI European Digital Cellular Telecommunication Systems, phase 2. *Unstructured Supplementary Service Data (USSD) - stage 3*. GSM 04.90. http://www.etsi.ch.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, und T. Berners-Lee. *RFC 2068: Hypertext Transfer Protocol — HTTP/1.1*, January 1997.

[12] Internet Mail Consortium (IMC). *vCalendar - The Electronic Calendaring and Scheduling Format, Version 1.0*. 18. Sep. 1996. http://www.imc.org/pdi/vcal-10.doc.

[13] Internet Mail Consortium (IMC). *vCard - The Electronic Business Card, Version 2.1*. 18. Sep. 1996. http://www.imc.org/pdi/vcard-21.doc.

[14] J. C. Mogul und J. Postel. *RFC 950: Internet Standard Subnetting Procedure*, August 1985. Updates RFC0792 [17].

[15] P. King, T. Hyland. *W3C Note: Handheld Device Markup Language Specification*. 09-May-1997. http://www.w3.org/pub/WWW/TR/NOTE-Submission-HDML-spec.html.

[16] J. Postel. *RFC 768: User Datagram Protocol*, August 1980.

[17] J. Postel. *RFC 792: Internet Control Message Protocol*, September 1981. Updated by RFC0950 [14].

[18] Rohit Khare. *Report: W* Effect Considered Harmful*. 1999. http://www.4k-associates.com/IEEE-L7-WAP-BIG.html.

[19] Jochen Schiller. *Mobilkommunikation - Techniken für das allgegenwärtige Internet*. Addison-Wesley, 1. Auflage, 2000. ISBN 3-8273-1578-6.

[20] Bruce Schneier. *Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C*. Addison Wesley, 3. Auflage, 1996. ISBN 3-8931-9854-7.

[21] T. Kamada. *W3C Note: Compact HTML for Small Information Appliances*. 09-Feb-1998. http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/.

[22] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 3. Auflage, 1996. ISBN 0-1334-9945-6.

[23] The Unicode Consortium. *The Unicode Standard, Version 2.0*. Addison-Wesley, Reading, MA, USA, 1996.

[24] Robert Tolksdorf. *Die Sprachen des Web: HTML und XHTML. Informationen aufbereiten und präsentieren im Internet*. dpunkt.verlag, Heidelberg, 4. Auflage, 1999. ISBN 3-932588-58-4.

[25] Robert Tolksdorf und Oli Kai Paulus. *Informationslokalisierung mit Suchmaschinen*. In Irene Heinen, editor, *Internet - von der Idee zum kommerziellen Einsatz. Deutscher Internet Kongreß 1998*, Seiten 27–34. dpunkt.verlag, 1998.

[26] W3C Working Draft. *XHTML Basic*. 10-Feb-2000. http://www.w3.org/TR/xhtml-basic.

[27] Bernd Walke. *Mobilfunknetze und ihre Protokolle, Bd.1, Grundlagen, GMS, UMTS und andere zellulare Mobilfunknetze*. Teubner, Stgt., 2. Auflage, 2000. ISBN 3-5191-6430-2.

[28] WAP Forum. *Wireless Application Protocol Architecture Specification*, 1998.

[29] WAP Forum. *Binary XML Content Format Specification*, 1999.

[30] WAP Forum. *Push Access Protocol Specification*, 1999.

[31] WAP Forum. *Push Architectural Overview*, 1999.

[32] WAP Forum. *Push Message Specification*, 1999.

[33] WAP Forum. *Push OTA Protocol Specification*, 1999.

[34] WAP Forum. *Push Proxy Gateway Service Specification*, 1999.

[35] WAP Forum. *WAP over GSM USSD Specification*, 1999.

[36] WAP Forum. *WAP Service Indication Specification*, 1999.

[37] WAP Forum. *WAP Service Loading Specification*, 1999.

[38] WAP Forum. *WDP/WCMP Wireless Data Gateway Adaption*, 1999.

[39] WAP Forum. *Wireless Application Environment Overview*, 1999.

[40] WAP Forum. *Wireless Application Environment Specification*, 1999.

[41] WAP Forum. *Wireless Control Message Protocol Specification*, 1999.

[42] WAP Forum. *Wireless Datagram Protocol Specification*, 1999.

[43] WAP Forum. *Wireless Identity Module*, 1999.

[44] WAP Forum. *Wireless Markup Language Specification*, 1999.

[45] WAP Forum. *Wireless Session Protocol Specification*, 1999.

[46] WAP Forum. *Wireless Telephony Application Interface Specification*, 1999.

[47] WAP Forum. *Wireless Telephony Application Interface Specification, GSM Specific Addendum*, 1999.

[48] WAP Forum. *Wireless Telephony Application Interface Specification, IS-136 Specific Addendum*, 1999.

[49] WAP Forum. *Wireless Telephony Application Interface Specification, PDC Specific Addendum*, 1999.

[50] WAP Forum. *Wireless Telephony Application Specification*, 1999.

[51] WAP Forum. *Wireless Transaction Protocol Specification*, 1999.

[52] WAP Forum. *Wireless Transport Layer Security Specification*, 1999.

[53] WAP Forum. *WMLScript Crypto API Library*, 1999.

[54] WAP Forum. *WMLScript Language Specification*, 1999.

[55] WAP Forum. *WMLScript Standard Libraries Specification*, 1999.

[56] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. W3C Recommendation, 1998. http://www.w3.org/TR/REC-xml.

[57] F. Yergeau. *RFC 2279: UTF-8, a transformation format of ISO 10646*, January 1998.

Index

In diesem Register verweist eine Seitenzahl in **halbfetter** Schrift auf die Einführung und Erklärung eines Begriffs, Tags oder Attributs. Mit normaler Schrift sind Seiten gekennzeichnet, auf denen ein Begriff genannt wird, zusätzliche Informationen stehen oder ein Tag oder Attribut in einem Beispiel Anwendung findet.

Symbole

' , **103**
(, **107**, **110**
) , **107**, **110**
* Operator, **108**, 111
*/ , **117**
*= Operator, 111, **111**
+ Operator, **107**, **108**, 111
++ Operator, **108**, 111
+= Operator, 111, **111**
, Operator, 111, 115, **115**, 122
- Operator, 102, **108**, 111
-- Operator, **108**, 111
-= Operator, 111, **111**
. , **103**
/ Operator, **108**, 111, 134
/* , **117**
// , **117**
/= Operator, 111, **111**
: , 107
; , 114, 115
< Operator, **110**, 111
<< Operator, **109**, 111, **111**
<<= Operator, 111
<= Operator, **110**, 111
= Operator, 111
== Operator, **110**, 111
> Operator, **110**, 111
>= Operator, **110**, 111
>> Operator, **109**, 111
>>= Operator, 111, **111**
>>> Operator, **109**, 111
>>>= Operator, 111, **111**
? , **107**
? : Operator, 111
<?xml>-Tag, **16**, 269
Übersetzungseinheit, 127
Überwachungsprotokoll, **220**
$, 25, 89
$$, **26**
% Operator, **108**, 111, 134
%= Operator, 111, **111**
& Operator, **109**, 111
&= Operator, 111, **111**
&& Operator, **109**, 111
{ , **117**
| Operator, **109**, 111
|= Operator, 111, **111**
|| Operator, **109**, 111
} , **117**
~ Operator, **109**, 111
! Operator, **109**, 111
<!-- -->-Tag, **12**, 270
!= Operator, **110**, 111
<!DOCTYPE>-Tag, **16**, 269
<![CDATA[ ]]>-Tag, **35**, 270
" , **103**
\' Operator, **104**
\/ Operator, **104**
\" Operator, **104**
\\ Operator, **104**
\b Operator, **104**
\f Operator, **104**
\n Operator, **104**

\o Operator, **104**
\r Operator, **104**
\t Operator, **104**
\u Operator, **104**
\x Operator, **104**
^ Operator, **109**, 111
^= Operator, 111, **111**
0X, **102**
0x, **102**

0-9
0, **102**

A
<a>-Tag, **31**, 40, 65, 70, 270
A-Netz, **186**
abort Funktion, **143**
abs Funktion, **139**
accept
  Wert von
    type-Attribut, 67, **69**, 274
accept-charset-Attribut
  bei <go>, **73**, 271
<access>-Tag, **19**, 269
access Schlüsselwort, 119
Access violation Fehler, **136**
accesskey-Attribut
  bei <a>, **32**, 270
  bei <anchor>, **35**, 270
  bei <input>, **65**, 271
Acknowledgement Headers, 245
Addendums, 202
Adobe Photoshop, 48
Advanced Mobile Phone System, **187**
agent Schlüsselwort, 119
alert Funktion, **165**
align-Attribut
  bei <img>, **45**, 270
  bei <p>, 12, **28**, 270
  bei <table>, **39**, 270
alt-Attribut
  bei <img>, **44**, 270
amp Kürzel, **25**
Amplitudenumtastung, 184
AMPS, **187**
<anchor>-Tag, **33**, 40, 65, 70, 270
Anforderung, 209
Antwort, 209
Anwendungsschicht, 210
Anzeige, 209
apos Kürzel, **25**
application/vnd.wap.wmlc, **5**
application/x-www-form-urlencoded
  Wert von
    enctype-Attribut, **73**, 273
ASCII, 23, 24
asynchrone Transaktionen, 244
Attribute, 4, 12
Auskunftsdienste, 173, **174**
Auszeichnungssprache, 1
Authentifizierung, 166, 217, 221

B
<b>-Tag, **26**, 40, 95, 269
B-Netz, **186**
Bündelfunk, 188
Base Transceiver System, 189
Basisstation, **182**
Bestätigung, 209
<big>-Tag, **28**, 40, 95, 269
Binary WML, **200**
Blöcke, 117
Bluetooth, 183
BMP, 48, 51
bottom
  Wert von
    align-Attribut, **45**, 273

-Tag, 10, **30**, 31, 40, 270
break Anweisung, 119, **124**
Broadcast, 183
BTS, 189
Bytecode, 252

C
C
  Wert von
    align-Attribut, **39**, 273
C-Netz, **186**
call cleared, 258
call connected, 258
Capability Negotiation, **236**, 240
Capability negotiation, 232
Card, 3, **15**
<card>-Tag, **20**, 76, 77, 85, 89, 269
case Schlüsselwort, 119
catch Schlüsselwort, 119
CDMA, **185**
ceil Funktion, **146**

center
Wert von
align-Attribut, 12, **29**, 273
change cipher, **221**
Channel, 257
characterSet Funktion, **144**
charAt Funktion, **148**
cHTML, **267**
Circuit Switched Data, **189**
class Schlüsselwort, 119
class-Attribut
bei <a>, **31**, 270
bei <access>, **20**, 269
bei <anchor>, **35**, 270
bei <card>, **20**, 269
bei <do>, **70**, 271
bei <fieldset>, **21**, 271
bei <go>, **73**, 271
bei <head>, **17**, 269
bei <img>, **46**, 270
bei <input>, **65**, 271
bei <meta>, **18**, 269
bei <noop>, **75**, 271
bei <onevent>, **77**, 271
bei <optgroup>, **59**, 270
bei <option>, **58**, 270
bei <postfield>, **71**, 271
bei <pre>, **36**, 270
bei <prev>, **74**, 271
bei <refresh>, **74**, 271
bei <select>, **57**, 270
bei <setvar>, **87**, 271
bei <table>, **40**, 270
bei <td>, **40**, 270
bei <template>, **83**, 271
bei <timer>, **79**, 271
bei <tr>, **40**, 270
bei <wml>, **16**, 269
Client-Zertifikat, 218, 221
Cluster, **185**
Code Division Multiple Access, **185**
Codemultiplex, **185**
columns-Attribut
bei <table>, **38**, 270
Compact HTML, **267**
compare Funktion, **154**
Computer Telephony Integration, 247
confirm Funktion, **165**
connectiong call indication, 258
const Schlüsselwort, 119
content-Attribut
bei <meta>, 18, **18**, 269
continue Anweisung, 119, **125**
Cookies, **231**
Crypto Bibliothek, 165–166
CSD, **189**
CTI, 247

D

D
Wert von
align-Attribut, **39**, 273
D1-Netz, 188
Datagramm-Paket, 226
DataTAC, **188**
Datentransfer-Protokoll, **220**
DCS 1800, **188**
Debug Library, 167
debugger Schlüsselwort, 119
Deck, 3, **15**
DECT, 188, **192**
default Schlüsselwort, 119
delete
Wert von
type-Attribut, **69**, 274
delete Schlüsselwort, 119
Dialogs Bibliothek, 164–165
Dialogs.alert Funktion, 120, 125
Dienstelemente, 207
Dienstelementtypen, **209**
Dienstgüte, 195
Dienstprimitive, **207**
Dienstzugangspunkt, 207
Digital Enhanced Cordless Telecommunications, 188, **192**
div Operator, **108**, 111, 119, 134
div= Operator, 111, **111**, 119
Divide by zero Fehler, **134**
<do>-Tag, 67, **67**, 71, 82, 83, 271
do Schlüsselwort, 119
domain Schlüsselwort, 119
domain-Attribut
bei <access>, **19**, 269
dpunkt.verlag, vi
DTMF, **256**
DTMF sent, 258
Dual-Tone Multifrequency, 256
Dublin Core Metadata Initiative, 17

E
E, **103**
e, **103**
E-Netz, 188
ECMA, 252
ECMA-Script, **252**
EDGE, **192**, 195
elementAt Funktion, **151**
elements Funktion, **150**
else Anweisung, **119**
else Schlüsselwort, 119
-Tag, 3, **27**, 40, 95, 269
emptyok-Attribut
bei <input>, **65**, 271
enctype-Attribut
bei <go>, **73**, 74, 271
Enhanced Data Rates for GSM Evolution, **192**
Entities, **24**
enum Schlüsselwort, 119
equiv Schlüsselwort, 119
Ereignismeldungen, 257
escape, **90**
escapeString Funktion, **160**
ETSI, 194
European Computer Manufacturing Association, 252
European Telecommunications Standardisation Institute, 194
Exception Data, 242
exit Funktion, **143**
export Schlüsselwort, 119
extends Schlüsselwort, 119
Extensible Markup Language, *siehe* XML
extern Schlüsselwort, 119, **127**, 131
External function not found Fehler, **136**

F
false
Wert von
forua-Attribut, **19**, 273
multiple-Attribut, **55**, 273
newcontext-Attribut, **21**, 274
optional-Attribut, **70**, 274
ordered-Attribut, **21**, 274
sendreferer-Attribut, **70**, 274
false Schlüsselwort, **101**, 109
Fatal library function error Fehler, **137**
FDMA, **185**
<fieldset>-Tag, **21**, 95, 271
finally Schlüsselwort, 119
Finanzanwendungen, 173
Finanzdienste, **175**
find Funktion, **149**
FLEX, **188**
float Funktion, **142**
Float Bibliothek, 145–147
Float.maxFloat() Funktion, 103
Float.minFloat() Funktion, 103
Floating-point overflow Fehler, **134**
Floating-point too large Fehler, **135**
Floating-point too small Fehler, **135**
Floating-point underflow Fehler, **134**
floor Funktion, **145**
for Anweisung, 119, **122**, 125
format Funktion, **155**
format-Attribut
bei <input>, **62**, 271
forua-Attribut
bei <meta>, **19**, 269
Frequency Division Multiple Access, **185**
Frequenzmultiplex, **185**
Frequenzumtastung, 184
function Anweisung, 119
function Schlüsselwort, **127**

G
Gaussian Minimum Shift Keying, 188
General Packet Radio Service, **191**
get
Wert von
method-Attribut, **72**, 273
getBase Funktion, **159**
getCurrentCard Funktion, **164**
getFragment Funktion, **159**
getHost Funktion, **157**
getParameters Funktion, **158**
getPath Funktion, **158**
getPort Funktion, **157**
getQuery Funktion, **159**
getReferer Funktion, **160**
getScheme Funktion, **156**
getVar Funktion, **162**
GIF, 48, 49, 51, 79
Global System for Mobile Communications, 187, **188**

GMSK, 188, 193
<go>-Tag, 34, 68, **70**, 77, 83, 86, 89, 162, 271
go Funktion, **162**
GPRS, **191**, 195, 265
GSM, 181, 187, **188**, 195, 212, 257, 265
gt Kürzel, **25**

H
Handheld Device Markup Language, **249**, 267
Handover, **182**
Handshake, **218**
Handshake-Protokoll, **220**
Handshake-Prozedur, 218
HDML, **249**, 267
<head>-Tag, **17**, 269
Header, 233
header Schlüsselwort, 119
height-Attribut
  bei <img>, **44**, 47, 270
help
  Wert von
    type-Attribut, **69**, 274
High Speed Circuit Switched Data, **189**
HLR, **191**
Home Location Register, **191**
href-Attribut
  bei <a>, **31**, 270
  bei <go>, 68, **70**, 89, 271
HSCSD, **189**
hspace-Attribut
  bei <img>, **45**, 47, 270
HTML, **249**
HTTP, 70, 72
http Schlüsselwort, 119
http-equiv-Attribut
  bei <meta>, **18**, 269
Hypertext Markup Language, **249**

I
<i>-Tag, **26**, 40, 95, 269
i-Mode, **266**
ICMP, 214
id-Attribut
  bei <a>, **31**, 270
  bei <access>, **20**, 269
  bei <anchor>, **35**, 270
  bei <card>, **20**, 269
  bei <do>, **70**, 271
  bei <fieldset>, **21**, 271
  bei <go>, **73**, 271
  bei <head>, **17**, 269
  bei <img>, **46**, 270
  bei <input>, **65**, 271
  bei <meta>, **18**, 269
  bei <noop>, **75**, 271
  bei <onevent>, **77**, 271
  bei <optgroup>, **59**, 270
  bei <option>, **58**, 270
  bei <postfield>, **71**, 271
  bei <pre>, **36**, 270
  bei <prev>, **74**, 271
  bei <refresh>, **74**, 271
  bei <select>, **57**, 270
  bei <setvar>, **87**, 271
  bei <table>, **40**, 270
  bei <td>, **40**, 270
  bei <template>, **83**, 271
  bei <timer>, **79**, 271
  bei <tr>, **40**, 270
  bei <wml>, **16**, 269
iDEN, **188**
if Anweisung, 119, **119**
Illegal floating-point reference Fehler, **134**
image/jpeg, 5
image/vnd.wap.wbmp, **5**
IMEI, **190**
<img>-Tag, 13, 31, 40, **43**, 270
import Schlüsselwort, 119
IMT, **194**
in Schlüsselwort, 119
iname-Attribut
  bei <select>, **55**, 270
incoming call indication, 258
incoming network text indication, 258
Infinite floating-point constant Fehler, **135**
Informationsdienste, 172, **174**
Initiated by the system Fehler, **137**
Initiated by the user Fehler, **137**
Initiator, 237
<input>-Tag, **60**, 85, 86, 271
insertAt Funktion, **152**
int Funktion, **145**
Integer overflow Fehler, **134**
Integer too large Fehler, **135**
Integrated Digital Enhanced Network, 188

International Mobile Telecommunications, **194**
Internetprotokolle, 210
`invalid` Schlüsselwort, 103, **104**, 105, 106, 110, 120, 121, 133–135
Invalid function arguments Fehler, **137**
IP, 211
IP-Tabelle, 231
Iridium, 183
IS-136, **188**, 257
IS-95, **188**
`isEmpty` Funktion, **148**
`isFloat` Funktion, **141**
`isInt` Funktion, **141**
ISO 10646, 23
ISO 639, 275
ISO 8859-1, 23, 26, 73, 104
ISO/OSI-Referenzmodell, **210**
`isValid` Funktion, **156**
`isvalid` Operator, **105**, 111, 119
`ivalue`-Attribut
bei `<select>`, **55**, 270

J
Java, 117, 121, 128, 136
Java Virtual Machine, 136
JavaScript, 113, 121, 252
JPG, 48, 49, 51
JScript, 252
JVM, 136

K
Kämmerer, Christine ♡, vi
Kanalzugriffsverfahren, 184, **184**
KBrowser, 95
Kommentare, *siehe auch* `<!-- -->`, `/*`, `*/`, `//`
, 12
Konditionalausdrücke, 107
Kurzmitteilung, 189
Kurzmitteilungszentrale, 189

L
`L`
Wert von `align`-Attribut, **39**, 273
`label`-Attribut
bei `<do>`, **69**, 71, 271
`Lang` Bibliothek, 129, 139–144
`Lang.abort` Funktion, 137
`Lang.float()` Funktion, 134
`Lang.maxInt()` Funktion, 102
`Lang.minInt()` Funktion, 102
leere Anweisung, 114
Leerraum, 24
Leerzeichen, 24
`left`
Wert von `align`-Attribut, **29**, 273
`length` Funktion, **148**
`lib` Schlüsselwort, 119
Link, **31**
`loadString` Funktion, **161**
`localsrc`-Attribut
bei `<img>`, **43**, 270
`lt` Kürzel, **25**

M
m-Commerce, 173, **177**
Macintosh, 95
`max` Funktion, **140**
`maxFloat` Funktion, **147**
`maxInt` Funktion, **142**
`maxlength`-Attribut
bei `<input>`, **62**, 271
`<meta>`-Tag, **18**, 95, 269
`meta` Schlüsselwort, 119
`method`-Attribut
bei `<go>`, **72**, 74, 271
MIBenum-Wert, 144
Microbrowser, **246**
`middle`
Wert von `align`-Attribut, **46**, 273
MIME, **251**, 253, 261
`min` Funktion, **139**
`minFloat` Funktion, **147**
`minInt` Funktion, **142**
Mobilfunknetze, **181**
Mobilitätsdienste, 173, **178**
Mobitex, 188
`mode`-Attribut
bei `<p>`, **29**, 270
Modulationsart, 184
Modulationstyp, 184
MSISDN, 203
`multipart/form-data`
Wert von `enctype`-Attribut, **73**, 273

multiple-Attribut
bei <select>, **55**, 57, 270
Multiplex, **184**
Multiplexverfahren, **184**

N
name Schlüsselwort, 119
name-Attribut
bei <do>, **70**, 83, 271
bei <input>, **60**, 86, 271
bei <meta>, 18, **18**, 269
bei <postfield>, **71**, 271
bei <select>, 54, **54**, 57, 270
bei <setvar>, **87**, 271
bei <timer>, **80**, 271
nbsp Kürzel, 25, 28, **30**
Negotiated Capabilities, 237
network status indication, 258
network text send, 258
new Schlüsselwort, 119
newContext Funktion, **163**
newcontext-Attribut
bei <card>, **21**, 85, 269
NMT, 187
noesc, **90**
Nokia WAP Toolkit, 48
<noop>-Tag, **75**, 77, 84, 271
Nordic Mobile Telephone, **187**
Not a number floating-point constant Fehler, **135**
Notepad, 91
nowrap
Wert von
mode-Attribut, **29**, 273
null Schlüsselwort, 119
Numerische Darstellung von Kürzeln, 25

O
onenterbackward
Wert von
type-Attribut, **77**, 274
onenterbackward-Attribut
bei <card>, 21, **76**, 269
bei <template>, **83**, 271
onenterforward
Wert von
type-Attribut, **77**, 274
onenterforward-Attribut
bei <card>, 21, **76**, 269
bei <template>, **83**, 271
<onevent>-Tag, 57, 77, **77**, 82, 83, 271
onpick
Wert von
type-Attribut, **77**, 274
onpick-Attribut
bei <option>, **57**, 76, 270
ontimer
Wert von
type-Attribut, **77**, 274
ontimer-Attribut
bei <card>, 21, **76**, 269
bei <template>, **83**, 271
Opera, 96
<optgroup>-Tag, **58**, 95, 270
optimiertes Handshake, **218**
<option>-Tag, 54, **57**, 76, 77, 270
optional-Attribut
bei <do>, **70**, 271
options
Wert von
type-Attribut, **69**, 274
ordered-Attribut
bei <card>, **21**, 269
Out of memory Fehler, **138**
outgoing call indication, 258

P
<p>-Tag, 11, **28**, 270
PAP, **204**
parseFloat Funktion, **141**
parseInt Funktion, **140**
password
Wert von
type-Attribut, **61**, 274
path Schlüsselwort, 119
path-Attribut
bei <access>, **19**, 269
PCN, 194
PDC, **188**, 257
PDU, **208**, 228
Personal Communications Network, 194
Personal Digital Cellular, 188
Personal Handyphone System, 188
Phasenumtastung, 184
PHS, **188**
Portnummer, 157
post
Wert von
method-Attribut, **72**, 273

<postfield>-Tag, **71**, 271
pow Funktion, **146**
Pragma, 131
<pre>-Tag, **36**, 95, 270
<prev>-Tag, 34, **74**, 77, 86, 271
prev
Wert von
type-Attribut, 69, **69**, 274
prev Funktion, **163**
private Schlüsselwort, 119
Private Anwendungen, 173
Programmed abort Fehler, **137**
Projektgruppe KIT/FLP, vi
prompt Funktion, **164**
Protocol Data Unit, 208, 228
Protokollschichten, **207**
Protokollstapel, **208**, 212
Provisioning Architecture, 266
public Schlüsselwort, 119
Pull, 233
Pull-Dienst, **204**
Push, 233
Push Access Protocol, **204**
Push Body, 245
Push Headers, 245
Push OTA, **205**
Push Proxy Gateway, **204**
Push-Übertragung, 223
Push-Architektur, 204
Push-Dienste, **204**
Push-Pull-Dienste, **205**
Push-Versand, 245
Push-Zustände, **239**

Q
Quadratwurzel, 147
quot Kürzel, **25**

R
R
Wert von
align-Attribut, **39**, 273
random Funktion, **143**
Raummultiplex, **184**
Reassemblierung, 227
Record-Protokoll, **219**
Referer, 70
REFLEX, **188**
<refresh>-Tag, 34, **74**, 77, 81, 86, 271
refresh Funktion, **164**
Regulierungsbehörde für Telekommunikation und Post, 194
removeAt Funktion, **151**
replace Funktion, **150**
replaceAt Funktion, **152**
Repository, 256, **257**
Request-Body, 235
Request-Headers, 235
Requested Capabilities, 237
reset
Wert von
type-Attribut, **69**, 274
resolve Funktion, **160**
Responder, 237
Response Body, 235
Response Header, 235
return Anweisung, 119, **129**
right
Wert von
align-Attribut, **29**, 273
Roaming, 242
Röhricht, Martin, vi
round Funktion, **146**

S
S-ConfirmedPush Primitiv, 245
S-Connect Primitiv, 240
S-Disconnect Primitiv, 241
S-Exception Primitiv, 242
S-MethodAbort Primitiv, 244
S-MethodInvoke Primitiv, 242
S-MethodResult Primitiv, 243
S-Push Primitiv, 245
S-PushAbort Primitiv, 245
S-Resume Primitiv, 242
S-SAP, 234, 240
S-Suspend Primitiv, 241
S-Unit-MethodInvoke Primitiv, 234
S-Unit-MethodResult Primitiv, 235
S-Unit-Push Primitiv, 235
SAP, **207**
schema-Attribut
bei <meta>, **18**, 269
Schlüsselaustausch, 217
SDMA, **185**
SDU, 226
SEC-Commit Primitiv, 217
SEC-Create Primitiv, 216

SEC-Create-Request Primitiv, 218
SEC-Exception Primitiv, 217
SEC-Exchange Primitiv, 217
SEC-SAP, 216
SEC-Terminate Primitiv, 217
SEC-Unidata Primitiv, 216
Secure Sockets Layer, 215
seed Funktion, **144**
Segmentierung, 227
<select>-Tag, **53**, **54**, 57, 85, 270
selektive Neuübertragung, 227
sendreferer-Attribut
    bei <go>, **70**, 271
service indication, **205**
service loading, **205**
session management, 232
setVar Funktion, 85, **162**
<setvar>-Tag, **86**, 271
SGML, **249**
Short Message Service, **189**
Shortcut Evaluation, 108
shy Kürzel, 25, **30**
Sicherungsschicht, 211
signText Funktion, **165**
SIM, 222
Sitzungsaufbau, 238
Sitzungsschicht, 211, 230
Sitzungsschlüssel, 217
Sitzungszustände, 230, **238**
size-Attribut
    bei <input>, **62**, 271
sizeof Schlüsselwort, 119
Slots, 185
<small>-Tag, **28**, 40, 95, 270
Smart Cards, 221
SMPP, 215
SMS, **189**
SMS Cell Broadcast, **190**
SMSCD, **190**
Space Division Multiple Access, **185**
sqrt Funktion, **147**
squeeze Funktion, **153**
src-Attribut
    bei <img>, 13, **43**, 270
SSL, 211, 215
Stack overflow Fehler, **138**
Stack underflow Fehler, **137**
Standard Generalized Markup
        Language, **249**
Standardattribute, *siehe auch*
        id-Attribut,
        class-Attribut,
        xml:lang-Attribut
, 16
Standardbibliotheken, 254
Stepan, Vera ♡, vi
String Bibliothek, 148–156
-Tag, **27**, 40, 95, 269
struct Schlüsselwort, 119
Subscriber Identity Module, 222
subString Funktion, **149**
super Schlüsselwort, 119
Suspendierung, 241
switch Schlüsselwort, 119

T

T-DError Primitiv, 213
T-DUnidata Primitiv, 213
T-SAP, 213
tabindex-Attribut
    bei <input>, **65**, 271
    bei <select>, **56**, 65, 270
<table>-Tag, 10, **37**, 270
TACS, 187
Tag, 1, 249
<td>-Tag, 10, **40**, 270
TDMA, **185**, 192
<template>-Tag, 20, 70, **82**, 271
Templates, 247
Terrestrial Trunked Radio, 188
TETRA, **188**
text
    Wert von
        type-Attribut, **62**, 274
text/html, 5
text/vnd.wap.wml, **5**
this Schlüsselwort, 119
throw Schlüsselwort, 119
TID, 226
Time Division Multiple Access, **185**
Time Slots, **185**
<timer>-Tag, 76, **78**, 85, 271
title-Attribut
    bei <a>, **31**, 270
    bei <anchor>, **35**, 270
    bei <card>, **20**, 89, 269
    bei <fieldset>, **21**, 271
    bei <input>, **62**, 271
    bei <optgroup>, **59**, 270
    bei <option>, **57**, 270
    bei <select>, **54**, 270
    bei <table>, **39**, 270

TLS, 211, 215
Token, 201
Tolksdorf, Leander, vi
Tonwahlverfahren, 256
top
  Wert von
    align-Attribut, **46**, 273
toString Funktion, **154**
TPI, 227
<tr>-Tag, **40**, 270
TR-Abort Primitiv, 230
TR-Invoke Primitiv, 228
TR-Result Primitiv, 230
TR-SAP, 228
Trägermedien, 211
Transaktionsidentifikator, **226**
Transaktion, 209, **223**
Transaktions-Handle, 228
Transaktionsabbruch, 230
Transaktionsklassen, **223**
Transaktionsschicht, 211
Transaktionszustände, **239**
Transport Information Items, 227
Transport Layer Security, 215
Transportinformationen, **227**
Transportschicht, 211
trim Funktion, **154**
true
  Wert von
    forua-Attribut, **19**, 273
    multiple-Attribut, **55**, 57, 274
    newcontext-Attribut, **21**, 85, 274
    optional-Attribut, **70**, 274
    ordered-Attribut, **21**, 274
    sendreferer-Attribut, **70**, 274
true Schlüsselwort, **101**, 109
try Schlüsselwort, 119
TU-Berlin, vi
type-Attribut
  bei <do>, 67, **69**, 271
  bei <input>, **61**, 271
  bei <onevent>, **77**, 83, 271
typeof Operator, **105**, 111, 119

U

<u>-Tag, 10, **26**, 40, 95, 269
UDCP, **191**
UDP, 211, 213
Umgebungsvariablen, 248
UMTS, 192, **193**, 265
Unable to load compilation unit Fehler, **136**
unesc, **90**
unescapeString Funktion, **161**
UNICODE, 23–26, 103, 104
Universal Mobile Telecommunications System, **194**
Universal Resource Identifier, 235
Unix, 95
unknown
  Wert von
    type-Attribut, **69**, 274
Unstructured Supplementary Services Data, **190**
UP.Browser, 95
url Schlüsselwort, 119
URL Bibliothek, 156–161
US-ASCII, 26, 73
use Schlüsselwort, 119
use url Anweisung, **131**
user Schlüsselwort, 119
user acknowledgement, 223
User Agents, **246**
User Datagram Protocol, 211, **213**
USSD, **190**, 212, 257
USSD Dialogue Control Protocol, 190
USSD-Stufe, 190

V

value-Attribut
  bei <input>, **61**, 271
  bei <option>, **57**, 270
  bei <postfield>, **71**, 271
  bei <select>, **54**, 55, 270
  bei <setvar>, **87**, 271
  bei <timer>, **79**, 81, 271
var Schlüsselwort, **117**, 119
vCalendar, **263**
vCard, **263**
Verification failed Fehler, **136**
Vermeidung von Versionskonflikten, **227**
Verschlüsselung, 221
Verschlüsselungsmethode, 217, 221
vnd.*
  Wert von
    type-Attribut, **69**, 274
void Schlüsselwort, 119

vspace-Attribut
bei <img>, **45**, 48, 270

W
WAE, 210, **246**, 249
WAP, 171, **197**, 207
WAP Identity Module, **221**
WAP Proxy, 200
WAP Specification Suite, 199
WAP-Angebote, **172**
WAP-Anwendungen, **172**
WAP-Dienst, 171
WAP-Forum, **199**, 266
WAP-Gateway, **200**, 203
WAP-Protokollstapel, **210**
wapalizer, 96
WAPDraw, 48
WapIDE, 91
WBMP, 43, **261**
WCMP, **214**
WDP, 211
while Anweisung, 119, **121**, 123, 125
White Space, 153
width-Attribut
bei <img>, **44**, 47, 270
WIM, 166, **221**
Windows, 95
winwap, 96
Wireless Application Environment, **203**, 210, **246**
Wireless Application Protocol, **197**
Wireless Bitmap Format, *siehe* WBMP
Wireless Control Message Protocol, 212, **214**
Wireless Datagram Protocol, 211, **212**
Wireless Markup Language, **1**, **249**
Wireless Session Protocol, 211, **232**
Wireless Telephony Application, 247
Wireless Telephony Application Interface, **202**, **256**
Wireless Transaction Protocol, 211, **223**
Wireless Transport Layer Security, 211, 215
with Schlüsselwort, 119
WML, **1**, **9**, **249**
<wml>-Tag, 11, **16**, 20, 269
WML 1.2
Tags und Attribute, vi, 32, 35, 36, 39, 64, 65, 73
WML User Agent, **250**
WML-Browser, 250
WMLBrowser Bibliothek, 162–164
WMLScript, **251**
WMLScript User Agent, **252**
Wohlgeformtheit, 10
Word, 91
Working Groups, 199
wrap
Wert von mode-Attribut, **29**, 273
WSP, 211, **232**
WSP-Einheiten, **236**
WSP/B, 233
WTA, 247
WTA Framework, 256
WTA User Agent, 256
WTA-Ereignisse, 257
WTA-Funktionen, 202
WTA-Gerüst, **256**
WTAI, **256**
WTAI-Funktionen, 259
WTLS, 211, **215**
WTP, 211, **223**

X
XHTML, **250**
XHTML Basic, **267**
XML, 9, 103, 201, **249**, 258
xml:lang-Attribut
bei <a>, **31**, 270
bei <anchor>, **35**, 270
bei <card>, **20**, 269
bei <do>, **70**, 271
bei <fieldset>, **21**, 271
bei <img>, **46**, 270
bei <input>, **65**, 271
bei <optgroup>, **59**, 270
bei <option>, **58**, 270
bei <select>, **57**, 270
bei <table>, **40**, 270
bei <td>, **40**, 270
bei <wml>, **16**, 269

Z
Zeichenkette, 148
Zeitablauf-Diagramm, **209**
Zeitmultiplex, **185**

Zeitrahmen, **185**
Zeitschlitze, **185**
Zellenmitteilung, 190
Zustandsdiagramm, 218