

Cracking the Code

WAP, Bluetooth, and 3G Programming



Take a look inside 15 professional wireless applications

Complete with design specs, flow charts and line by line code analysis

**Dreamtech
Software Team**

WAP, Bluetooth, and 3G Programming

WAP, Bluetooth, and 3G Programming

Cracking the Code

Dreamtech Software Team



Hungry Minds™

Best-Selling Books • Digital Downloads • e-Books • Answer Networks •
e-Newsletters • Branded Web Sites • e-Learning

New York, NY ♦ Cleveland, OH ♦ Indianapolis, IN

WAP, Bluetooth, and 3G Programming: Cracking the Code

Published by

Hungry Minds, Inc.

909 Third Avenue

New York, NY 10022

www.hungryminds.com

Copyright © 2002 Hungry Minds, Inc. All rights reserved. No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

Library of Congress Control Number: 2001095398

ISBN: 0-7645-4905-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/RR/RQR/QR/IN

Distributed in the United States by Hungry Minds, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria, and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For general information on Hungry Minds' products and services please contact our Customer Care department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and reseller information, including discounts, premium and bulk quantity sales, and foreign-language translations, please contact our Customer Care department at 800-434-3422, fax 317-572-4002 or write to Hungry Minds, Inc., Attn: Customer Care Department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. THE PUBLISHER AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

Trademarks: Hungry Minds and the Hungry Minds logo are trademarks or registered trademarks of Hungry Minds, Inc. Macromedia, Homesite, and ColdFusion are trademarks or registered trademarks of Macromedia, Inc., in the United States and/or other countries. All other trademarks are the property of their respective owners. Hungry Minds, Inc., is not associated with any product or vendor mentioned in this book.



Hungry Minds® is a trademark of Hungry Minds, Inc.

Credits

Acquisitions Editor

Chris Webb

Project Editor

Neil Romanosky

Technical Editor

N. R. Parsa

Copy Editors

Jerelind Charles

Jeremy Zucker

Media Development Specialist

Angela Denny

Permissions Editor

Carmen Krikorian

Media Development Manager

Laura Carpenter VanWinkle

Project Coordinator

Nancee Reeves

Cover Design

Anthony Bunyan

Proofreader

Mary Lagu

Indexer

Johnna VanHoose Dinse

Cover

Vault door image used courtesy of

Brown Safe Manufacturing

www.BrownSafe.com

Dreamtech Software India, Inc., Team

dreamtech@mantraonline.com

www.dreamtechsoftware.com



Dreamtech Software India, Inc., is a leading provider of corporate software solutions. Based in New Delhi, India, the company is a successful pioneer of innovative solutions in e-learning technologies. Dreamtech's developers have over 50 years of combined software engineering experience in areas including Java, wireless applications, XML, voice-based solutions, .NET, COM/COM+ technologies, distributed computing, DirectX, Windows Media technologies, and security solutions.

About the Authors

Dr. K. V. K. K. Prasad is a renowned software engineer and professor with extensive experience in software engineering, wireless Internet, computer telephony integration, artificial intelligence, data communication, and telecommunications. He is a software consultant.

Vikas Gupta is co-founder and president of Dreamtech Software. He is a software engineer and publisher actively engaged in developing and designing new technologies in wireless, e-learning, and other cutting-edge areas. He is also the managing director of IDG Books India (P) Ltd.

Avnish Dass, co-founder and CEO of Dreamtech Software, is a talented and seasoned programmer with 15 years of experience in systems and application/database programming. He has developed security systems, anti-virus programs, wireless and communication technologies, and ERP systems.

Deepesh Jain is a certified software developer and Microsoft Certified Professional with over three years of experience in VB, .NET, database programming, COM/COM+, Windows programming, and wireless technologies. He is a senior software developer at Dreamtech Software.

*To our parents and family and our beloved country India,
for providing an excellent environment
for nurturing and creating world-class IT talent.*

Preface

The last three decades of the twentieth century yielded two revolutionary developments in communications technology: the Internet and mobile communication networks. The Internet — the global network that integrates all computer networks — enables us to access data services from our desktops. Mobile communications pave the way for voice communication services for people who are (literally) on the move. The tremendous impact of both these developments is evident from the growth rates of both Internet and the mobile network subscribers throughout the world. Standing at the beginning of the first decade of the twenty-first century, we will now witness the next revolutionary communications development: the integration of the Internet and mobile communications. This will pave the way for wireless Internet access and high-speed services on wireless devices.

The ability to access Internet services through mobile devices will lead to anywhere–anytime communication. If wireless networks can support high speeds, users will be able to access data, voice, and video services. As is generally the case, the end user will probably not be concerned about the underlying technologies; he/she will be interested only in the applications that these networks support. Thus, strong application development is essential in order for futuristic wireless networks to achieve wide popularity. This book addresses the subject of developing applications for wireless Internet access.

An Overview of the Technology

A variety of tools and languages have been developed to create content for wireless systems that support applications with voice and video streaming services. This book describes in detail the tools and languages required to develop outstanding applications over wireless networks using the following three technologies:

- ◆ WAP
- ◆ Bluetooth
- ◆ 3G

WAP

Wireless Application Protocol (WAP) allows users to access Web content on low-speed wireless networks such as GSM, IS-136, and PDC. WAP was developed as an open standard protocol to bridge the wired Internet and the wireless networks. The WAP Forum was launched in December 1997 by Ericsson, Motorola, Nokia, and Phone.com. WAP specifications define the protocol conversion between the IP and cellular networks, as well as the markup language to create content for wireless Internet access.

The wired Internet uses the TCP/IP protocol stack and HTTP to access Web services. The desktop PC is a powerful system with a high-resolution monitor, high processing capability, and an ability to present rich multimedia content to the user through a browser. All this requires huge system resources. To provide Web services to mobile clients is a challenge because mobile networks support low data rates (300 to 14.4 Kbps), and delays are frequent. In addition, mobile devices have small displays (2 to 4 lines with 8 to 12 characters per line), low resolution, no support for color, a limited-function keypad, low battery power, and low processing power. WAP has been developed as a lightweight protocol based on TCP/IP and HTTP. A WAP gateway bridges the WAP protocols and the Internet protocols by carrying out the necessary protocol conversion. To develop content that mobile devices can access, the Wireless Application Environment (WAE) is specified as a part of WAP. The WAE consists of

- ◆ Wireless Markup Language (WML), a page description language that describes the content presentation. WML is similar to HTML and is based on XML.
- ◆ WML Script, a scripting language similar to JavaScript that can be used to facilitate calculations, validate user input, generate error messages locally, and pass parameters to the server.
- ◆ Content formats to describe the data, images, and other content.
- ◆ A micro-browser that runs on the mobile devices. The micro-browser occupies few system resources and provides only limited functionality, as compared with desktop browsers such as Internet Explorer and Netscape.

WAP is an open standard that has the support of major equipment manufacturers, service providers, and software developers. WAP 1.1 was released in June 1999, and Version 1.2 was released in November 1999. During the past few years, a number of content providers have developed WAP content for applications — such as obtaining stock quotes, weather information, astrological information, sports news, and so on. Other applications that are now commercially available include mobile commerce, mobile advertising, and mobile banking.

WAP has shown us the possibilities of using Internet access to obtain focused information on mobile phones in text format. However, as the capabilities of mobile devices improve and the data rates of the wireless networks increase, we now need to consider using other markup languages for wireless applications. Case in point: WAP has been revised to support XHTML for content creation.

The first part of this book addresses content creation for providing wireless Internet access using WAP. We discuss content development using WML, WML Script, Cold Fusion, and Java technologies for creating applications using server-side programming and database access.

Bluetooth

Today's business executive uses a large number of devices — desktop PC, laptop, PDA, mobile phone, and the like — in addition to peripherals such as a fax machine, LCD projector, cordless phone, and so on. These devices need to share information and resources, but interconnecting them through cables is cumbersome. Ideally, when two or more devices that need to share data are in close proximity, they should be able to form a network and exchange the data. That is, the devices should be capable of forming an ad-hoc network on their own and sharing data through simple commands given by the user. Bluetooth achieves this through a low-cost, low-power, short-range radio technology. Ad hoc networks can be formed among Bluetooth-enabled devices in the office, home, or car. Almost every electronic device can be Bluetooth enabled, be it a PC, laptop, printer, fax machine, modem, mobile phone, LCD projector, digital camera, cordless phone, music system, television, microwave oven, or Web TV.

Bluetooth is a nascent technology that harbors enormous potential. A large number of vendors have developed the hardware and software to make devices Bluetooth enabled. Now the technology is also maturing as a cost-effective solution to replace cable. Developing exciting applications on the Bluetooth protocol stack is the “need of the hour.” This book presents a comprehensive coverage of Bluetooth programming. We also examine the many interesting applications that can be developed through a combination of WAP and Bluetooth.

3G

End-users' desire for increased bandwidth is paving the way for wireless networks that support higher data rates. The 2.5 Generation wireless networks that are evolving from the present 2G networks will support data rates in the range 64–144 Kbps. These 2.5G systems will, in turn, evolve into 3G systems that will support data rates in the range 384–2048 Kbps. Such data rates can support services such as high-resolution graphics and animation, downloading music from the Internet, teleshopping, multiparty audio and video conferencing, audio/video broadcasting over mobile networks, and so forth

The 2G and 2.5G wireless networks are based on protocols that conform to regional standards. Wireless networks in Europe, North America, and Japan are based on different standards. 3G systems aim to achieve global roaming by providing appropriate gateways for protocol conversion depending on the user's geographic location. Wireless networks based on 3G networks are yet to be deployed on a large scale.

In order for 3G networks to be profitable, they must support quality content and applications. Developing applications that provide low-cost data, voice, and video services is the biggest challenge; software developers and content providers need to concentrate on this aspect in the years to come. A number of alternatives, such as XHTML, XML, Java, and C++, are available for content development. In addition, mobile devices that access the Internet will have different capabilities in terms of memory, processing power, display resolution, size, and so on. To develop content that can cater to all types of devices is a great challenge to content creators. Content creators have to work with a wide variety of tools to create killer applications that the end user can use to carry out his/her business, education, and entertainment activities through mobile devices, all at a very low cost.

We study aspects of 3G programming in detail in this book. We use the various tool kits available to test the content in the laboratory environment before deploying it on the network. We focus on creating applications for animation, voice, and video services using XHTML, XML, and Java. We use Qualcomm's BREW toolkit to do the 3G programming.

What This Book Covers

This book is based on the unique concept of *Cracking the Code* and, consequently, mastering the technologies of WAP, Bluetooth, and 3G programming. This book is *not* meant for beginners: It will teach you only the basics of specific technologies. The *Cracking the Code* series is meant for software developers/programmers who wish to upgrade their skills and understand the secrets behind professional-quality applications. This book starts where other tutorial books end. It will enhance your skills and take them to the next level as you learn a particular technology.

This is the first book to cover both Bluetooth and 3G programming. It contains a unique coverage of using WAP with Bluetooth and 3G content development for multimedia applications. The book is code intensive, with a lesser emphasis on theory. All the applications (and related source code) have been fully tested at Dreamtech Software Research Lab. The source code in this book is based on commercial applications developed by Dreamtech. Each program is explained in a very detailed manner so as to provide insight into the implementation of the technology in a real-world situation. The appendices provide reference links so that the earnest reader can further explore the new developments that are taking place.

Please note that this book does not provide a comprehensive tutorial of specific technologies — it provides only command summaries, as there are plenty of books available to teach you WML, WML Script, Cold Fusion, XHTML, Java Servlets, JSP, and the theoretical aspects of the Bluetooth and 3G protocols.

This book's objective is to put you on the evolutionary pathway of wireless communication and to help you develop exciting software that provides rich content and applications on wireless networks. The book begins with WAP content development using WML, WML Script, JSP, Servlets, Cold Fusion, and other technologies. It then moves on to a discussion of Bluetooth technology and finally to a detailed discussion of 3G. Emphasis is placed on developing applications for Bluetooth and 3G networks. The discussion of 3G content development is targeted to programmers and communication engineers and enables them to use available toolkits in their work. This is the first book that addresses 3G wireless application development and conversion of the WAP applications to 3G applications. This book is also unique in that it provides the programmer with a holistic approach to content development using various markup and programming languages to create high-end multimedia applications.

In the future, every mobile device that is capable of accessing the Internet services through wireless networks will need to be Bluetooth enabled, so that the user can have completely wireless Internet access and data synchronization on various devices. This book gives you the programming ammunition to achieve this objective. Anywhere–anytime communication is the objective of the Global Village, and this book is designed to make you the architect of that village.

Who Should Read This Book

As it was stated earlier, this book is not for beginners. It is intended for experienced wireless application developers who want to learn the third-generation technologies, 3G and Bluetooth, that serve to integrate hardware peripherals such as refrigerators, televisions, ovens, and mobile phones with a PC. The book mainly targets innovative developers who envision developing their own applications along these lines. This book will also benefit those who aspire to explore the relatively new concept of WAP, as it outlines all of the vital aspects of this technology.

Because this book does not provide a comprehensive tutorial of relevant technologies needed for WAP, Bluetooth, and 3G programming, the reader should also have a working knowledge of Java, XML, WML, Visual C++, and JMF.

How the Book Is Organized

This book contains 14 chapters and five appendixes, which are described as follows:

- ◆ Chapter 1 provides a brief explanation of the evolution of WAP, Bluetooth, and 3G, as well as the devices used for these applications.
- ◆ Chapter 2 demonstrates content development using WML and WML Script with the help of two case studies. These case studies illustrate front-end application development for WAP.
- ◆ Chapter 3 explains the integration of WAP with Cold Fusion. Only the relevant details of the technology are reviewed, but the project helps explain every aspect of a Cold Fusion application. The emphasis here is on using Cold Fusion for server-side programming with MS Access as the database and WML for content presentation.
- ◆ Chapter 4 contains an introduction to the WTA architecture and programming. It describes the applications of WTA to integrate data and voice applications on mobile devices.
- ◆ Chapter 5 focuses on the integration of Java with WAP. The two main Java technologies used for Internet applications — JSP and Java Servlets — are also discussed.
- ◆ Chapter 6 discusses push technology in the WAP framework. We access Internet services using the pull model, whereby the user sends a request to the server and the server responds with content. Another model, the push model, is now being used to provide services such as stock quotes, advertisements, and so on, when the user has not specifically requested that information.
- ◆ Chapter 7 provides an introduction to Bluetooth technology and protocols. This chapter includes discussions on Bluetooth hardware, software, architecture, and protocols, as well as on Bluetooth applications for creating Personal Area Networks (PANs).
- ◆ Chapter 8 presents applications that implement WAP with Bluetooth. Using Bluetooth as the bearer, one can develop useful applications: for example, information kiosks that transmit information to mobile devices in public places such as airports and shopping malls. The implementation of this application is also discussed in this chapter.
- ◆ Chapter 9 focuses on programming aspects of Bluetooth. Using Ericsson's PC Reference stack, you can see how each layer of the Bluetooth protocol stack can be accessed and also how applications can be developed.
- ◆ Chapter 10 is an introduction to 3G. The fundamental principles of cellular mobile communication and the Global System for Mobile Communications (GSM) are covered, followed by a discussion

of the evolution of wireless networks into 2.5G and 3G networks. The various applications of 3G networks, which can support data rates in the range 384–2048 Kbps, are also discussed. This chapter also presents developments in mobile devices and languages for content development.

- ◆ Chapter 11 covers advanced 3G programming. It illustrates the limitations of WML and focuses on content development using XHTML, XML/XSL, and Java.
- ◆ Chapter 12 focuses on 3G content development using Qualcomm's Binary Runtime Environment for Wireless (BREW) toolkit, a powerful tool for creating content for CDMA-based networks.
- ◆ Chapter 13 deals with using 3G programming to develop multimedia content over IP networks. We discuss the implementation of audio and video streaming applications using Java Media Framework (JMF). These futuristic applications will enable us to access audio and video services from mobile devices that support Mobile IP protocol.
- ◆ Chapter 14 reviews the exciting developments taking place that will lead to the convergence of networks and services. In this chapter, we peep into the futuristic developments in mobile communications. The integration of mobile networks with broadcasting and fixed networks will lead to low-cost high-speed data, voice, and video services. This chapter discusses the various technologies and standards needed to achieve this convergence.
- ◆ Appendix A contains a discussion of the contents of this book's CD-ROM.
- ◆ Appendix B walks you through Tomcat installation and configuration so that you can install the software and run the code in the book. Tomcat is the toolkit required to work with Java servlets, and JSP.
- ◆ Appendix C covers the installation of SQL Server 2000 and XML Support Configuration.
- ◆ Appendixes D and E contain lists of URLs for sites that provide information on Bluetooth and 3G, respectively.

All the code provided in this book has been 100% tested and verified. The Nokia toolkit is used for WML and WML Script. Cold Fusion studio is used for working with Cold Fusion, and Jakarta Tomcat is used for working with JSP and Java Servlets. The procedures for installing and configuring all these software packages are also explained in the book. Qualcomm's BREW is the other tool kit used for 3G programming. All the applications have been developed on a Microsoft platform with Windows NT/ME/98/2000 as the base.

Let's now begin our exciting journey into the realm of content development for wireless networks.

Acknowledgments

We would like to acknowledge the contributions of the following people for their support in making this book possible: John Kilcullen, for sharing the dream and providing the vision in making this project a reality; Mike Violano and Joe Wikert, for believing in us; and M. V. Shastri, Asim Chowdury, V. K. Rajan, Sanjeev Chatterjee, and Priti for their immense help in coordinating various activities throughout this project. We also thank technical writers Mridula Sharma and Sunil Gupta, who contributed in developing this book's content.

Contents

Preface	vii
Acknowledgments	xi
Chapter 1: WAP, Bluetooth, and 3G: A Brief Introduction	1
Evolution of Wireless Networks	1
Evolution of Wireless Protocols and Applications	1
Languages and Tools for Content Development	2
Wireless Access Devices/Bluetooth.....	3
Summary	4
Chapter 2: WML and WML Script Programming: A Case Study	5
WML Commands and Syntax	5
WML Script — Commands and Syntaxes	7
The Information Master Application	9
The Restaurant Application.....	15
Summary	25
Chapter 3: WAP Using Cold Fusion: A Project	26
Cold Fusion: An Overview.....	26
Application: Question Quiz.....	29
Summary	48
Chapter 4: WTA: An Advanced Interaction Technique for Mobile Phones.....	49
Applications of WTA	49
Introduction to WTA Architecture	50
Using the Interface Components	54
Event and State Management in WTA.....	59
WTAI Function Call Example	60
Summary	62
Chapter 5: Integrating Java with WAP	63
Introduction to Java Technologies	63
Create Dynamic Content with Servlets and JSPs for WAP Browsers.....	66
A JSP and Servlets-Based Application for WAP	68
Summary	83
Chapter 6: Push Technology in WAP	84
Pull Technology for Accessing Internet Content.....	84
What Is Push Technology?.....	84
Push Technology Applications.....	85
Push Technology Implementation.....	86
Push Framework in WAP	89

Push Proxy Gateway	94
Develop the Database and Servlet Applications	95
Application: Pushing the Stock Quotes	96
Application: Shopping Cart with Advertisement Push	107
Pros and Cons of Push Framework	125
Summary	126
Chapter 7: Bluetooth: A Basic Introduction	127
Introduction to Personal Area Networks (PANs)	127
Overview of Bluetooth	127
Bluetooth System Specifications	130
Bluetooth versus Other Technologies	131
Commercial Bluetooth Solutions	132
Network of Bluetooth Devices: Piconet and Scatternet	134
Data and Voice Support	134
Security Issues in Bluetooth	135
Architecture of a Bluetooth System	135
Bluetooth APIs for Developing Applications	147
Summary	147
Chapter 8: Using WAP with Bluetooth	148
Bluetooth as a WAP Bearer	148
Application of WAP with Bluetooth	148
Implementation of WAP for Bluetooth	153
Addressing in WAP with Bluetooth	153
Application: Airport Kiosk	154
Application: Shopping Mall Kiosk	158
Summary	162
Chapter 9: Bluetooth Programming	163
Overview of the Bluetooth Development Kit	163
Installing the Bluetooth Module and PC Reference Stack	163
HCI Programming	163
Registering and Discovering Services: SDP Programming	194
File Transfer Application	212
Application: Chat	271
Summary	323
Chapter 10: An Overview of 3G	325
Principles of Cellular Mobile Communications	325
Multi-Cell Wireless Networks	326
Cellular System Design Issues	327
First Generation Wireless Networks	328
Second Generation Wireless Networks	328
2.5G Wireless Networks	339
Third Generation Wireless Networks	341
Summary	346

Chapter 11: Advanced 3G Programming.....	349
3G Application Development Issues.....	349
Implementation of Real-World 3G Applications	352
Development of a Mobile Advertising Application Using the Wireless Tool Kit	370
Summary	375
Chapter 12: 3G Programming Using BREW	376
BREW Overview	376
Using BREW to Develop a New Application	377
Application: Developing Animation	384
Application: Downloading Music onto a Mobile Device	393
Application: Mobile Advertisements	399
Application: Database	409
Summary	419
Chapter 13: Voice and Video Communication over IP and Mobile IP Networks.....	420
Application of Voice and Video over IP.....	420
Protocols Overview.....	421
Low Bit Rate Coding of Voice and Video.....	421
H.323 Standards	422
Java Media Framework.....	423
Application Setup.....	424
Application: Voice Messaging	424
Application: Audio Broadcasting.....	434
Application: Audio–Video Broadcasting.....	446
Summary	458
Chapter 14: The Future of Wireless Networks	460
Convergence Technologies.....	460
Emerging Technologies.....	464
Instant Messaging	465
Unified Messaging	465
Precise Location-Based Services.....	467
Mobile Devices	467
Tools for Content Development	468
VoiceXML	468
SyncML	470
Protocols.....	470
Mobile IP.....	472
4G Systems.....	472
Summary	473
Appendix A: What's on the CD-ROM	474
System Requirements	474
CD Contents	474
Troubleshooting	476
Appendix B: Tomcat Installation and Configuration	477

Introduction to a Web Server	477
How a Web Server Works: An Overview	477
Introduction to the Tomcat Web Server	478
Install the Tomcat Web Server	478
Deploy Web Applications to Tomcat	486
Deploy a Web Application to Tomcat	488
Appendix C: SQL Server 2000 Installation and XML Support	
Configuration	490
About MS SQL Server 2000	490
Complete Installation of SQL Server 2000 (Setup)	493
XML Support in SQL Server 2000	501
Using IIS (Internet Information Server) for Accessing SQL Server 2000	502
Appendix D: Bluetooth Reference and Resources	508
Appendix E: 3G Reference and Resources	510
Index	512
End User License Agreement	528
Sun Microsystems, Inc. Binary Code License Agreement	530
License Agreement: Forte for Java Release 2.0 Community Edition for All Platforms	533

Chapter 1

WAP, Bluetooth, and 3G: A Brief Introduction

For people on the move, voice communication has been the killer application for many years. The demand for data services by mobile users has increased in recent years, and as a result, new protocols have emerged for providing wireless Internet access. The demand for multimedia services is now paving the way for high-speed, wireless networks that can support innovative applications combining data, graphics, voice, and video. In this chapter, we review the evolution of wireless networks and the applications supported by these networks. We also look at the new languages and tools used to develop content for various applications. Because of industry efforts to support multimedia services, mobile devices are evolving into powerful gadgets. This chapter also contains a brief overview of these developments.

Evolution of Wireless Networks

The cellular networks developed in the 1960s and 1970s were mostly analog systems that supported voice communication. Subsequently, digital mobile communication networks, which are known as the second generation (2G) wireless networks, came into vogue. The 2G networks aren't based on international standards, but on regional standards developed in North America and Europe. North American standards include IS 136 and IS 95A (IS stands for Interim Standard), and the European systems are based on GSM (Global System for Mobile Communications). Asian and African countries adapted the North American and European standards. These 2G networks support data rates up to a maximum of 14.4 Kbps. Hence, applications supported on these networks are capable of handling only text and low-resolution graphics.

The 2G networks are now evolving into 2.5G networks, which can support data rates in the range of 64 to 144 Kbps. Examples of 2.5G networks are the IS 95B standard-based networks that evolved from IS95A networks and the GPRS (General Packet Radio Service) networks built over the GSM networks. These networks can support high-speed data services such as high-resolution graphics and animation, audio, and low bit rate video services.

The 2.5G networks will, in turn, evolve into third generation (3G) networks, which will support data rates in the range of 384 to 2048 Kbps. The standardization efforts of many international bodies resulted in a few proposals for 3G networks; however, a single standard has not evolved, mainly because the 3G networks have to evolve from the existing networks. Two standards that are likely to find wide acceptance are W-CDMA (Wideband Code Division Multiple Access) systems, which evolve from GSM systems; and cdma2000 systems, which evolve from IS 95B systems. As these networks support higher data rates, they will be able to support full-fledged multimedia applications with streaming audio and video.

Evolution of Wireless Protocols and Applications

The 2G systems support data rates up to 14.4 Kbps only. Moreover, these networks are characterized by high delay. The mobile devices have limited capability for accessing the Internet — in other words, low processing power, small memory capacity, and small display. The browser that can be run on these

devices is also of limited capability. So, to provide access to Internet services, the Wireless Application Protocol (WAP) was developed. WAP enables Internet browsing through a set of protocols, which are based on TCP/IP but with low protocol overhead so that the protocols can run on small devices such as mobile phones and pagers. To WAP-enable a mobile phone, the WAP protocol stack and a micro-browser need to run on the mobile phone. The WAP gateway interfaces between the mobile network and the Internet to provide the content to the mobile phone. WAP-enabled mobile phones can obtain very focused information such as stock quotes, weather information, and news headlines. Applications such as mobile banking to access the bank account information, mobile advertising to display product information on the mobile devices, and so on are also finding wide acceptance. Entertainment content, such as astrological information, sports news, and betting odds has also gained wide acceptance among WAP users.

The WAP Forum was launched in December 1997 by Ericsson, Motorola, Nokia, and Phone.com. Most of the wireless equipment manufacturers and operators are committed to the WAP standards. WAP 1.1 was released in June 1999 and WAP 1.2 in November 1999. The latest version of WAP — WAP 2.0 — was released in July 2001. Another service, called I-Mode, has gained wide popularity in Japan. I-Mode offers the same services as WAP but in packet-switching mode and at a higher speed. I-Mode is now making in-roads in other countries as well.

WAP protocol has been developed to provide wireless Internet access on low-speed networks. When high-speed networks are available, and if the mobile devices have higher processing capability, they can support the TCP/IP protocol stack. The content that you access through your desktops, such as high-resolution graphics, animation, and audio and video clips, can be accessed through mobile devices as well. But the present IP (Internet Protocol) has been designed for fixed terminals. The Mobile IP (MIP), which is now standardized, can run on the mobile devices to provide access to all the Internet services for mobile devices.

Because the TCP (Transmission Control Protocol) is not well suited for real-time audio and video communication, the UDP (User Datagram Protocol) is used to carry the voice and video data over the IP networks. Above the UDP, the RTP (Real Time Transport Protocol) is used to provide real-time capability. The mobile devices and servers that support the RTP and related protocols will provide the users with real-time voice and video transmission over the IP networks.

Languages and Tools for Content Development

The Internet content that you access from desktops is written mostly by using HTML (HyperText Markup Language). HTML content is transferred from the Web server to the client machine. The content is then interpreted by a browser such as Internet Explorer (IE) or Netscape Navigator (NN). The computing power requirements for these browsers are enormous, in regard to both primary and secondary memories. Because mobile devices didn't have the capability to run such powerful browsers, new markup languages were needed to create content that could be presented to the mobile devices. WAP 1.2 uses the Wireless Markup Language (WML), which is derived from XML (eXtensible Markup Language). WMLScript, which is similar to JavaScript, was developed to provide interactive capability to the content. The support for graphics is limited in WAP, which supports the WBMP (Wireless Bitmap) format, and provides very low-resolution graphics.

To facilitate the development of content using WML and WMLScript, a number of tool kits are available for testing the complete application in the laboratory environment before deploying the application on the site. In this book, we will discuss content development for advanced WAP applications. Content can be developed and tested using any of the popular WAP tool kits.

The drawback with WML is that the content presently available on the Internet needs to be rewritten for mobile access. This is a gigantic task. It's possible to have tools that convert the HTML content to WML content, but the conversion won't be very effective because WML supports only a subset of the HTML tags.

To develop content that can be accessed through high-speed, wireless networks by more powerful wireless devices, new markup languages are required. A number of these languages are now being used for content development. The content can be written in XML, which is a *meta-language*, or a language used to develop other languages. Another standardized language is XHTML (eXtensible HyperText Markup Language), which is based on XML with almost the same tags as HTML. XHTML obviates the need for rewriting the content to be made available to mobile devices if the mobile devices can interpret XHTML. Another advantage is that in XHTML, the syntax must be followed strictly. As a result, you can develop browsers that don't need high processing capability. This book covers the content development by using XML and XHTML, the markup language standardized for applications in WAP 2.0 specifications.

You can exploit the Java programming language's capability of platform independence and network-centric programming to develop Internet content using Java. The content, in the form of applets, can be downloaded to the mobile device and executed, provided the mobile device has a Java Virtual Machine (JVM). Because the JVM has a high memory requirement, Sun Microsystems has released the KVM (where K stands for kilobyte), a virtual machine that requires a few kilobytes (as low as 160K) of memory. Using the KVM and a subset of the libraries, you can develop content for wireless devices and the code can move from the server to the mobile device. Sun Microsystems's Wireless tool kit facilitates development of Java applications using J2ME (Java 2 Micro Edition). In this book, we demonstrate application development using J2ME.

The CDMA-based systems, which have a large installation base in North America, have also found wide acceptance in many Asian countries. Qualcomm Corporation — which pioneered the development of CDMA technology — released BREW (Binary Runtime Environment for Wireless), which provides the environment to develop applications for wireless networks. BREW can be effectively used for development of content/applications irrespective of the air interface and the speed of the networks. This book demonstrates application development using BREW.

Wireless Access Devices/Bluetooth

The wireless devices of 2G support voice and data applications by using Short Messaging Service (SMS). WAP-enabled phones are used to obtain the WAP content. All these devices have limited processing capability (8 or 16 bit micro-controllers), small memory (generally less than 64 Kbytes), a small black and white display that can support 2 to 4 lines of text with 8 to 12 characters per line, and a keypad with limited functionality. The WAP protocol and WML have been developed to take care of these limited capabilities of mobile devices.

In recent years, exciting developments have taken place in the world of mobile devices, which made them very powerful. They now have high processing power, high battery life, larger color display, and a full-fledged keyboard. In addition, peripherals such as video cameras are being integrated into mobile devices. For effectively managing the input/output operations, memory, and the various processes that run on the mobile device, mobile operating systems such as Win CE, EPOC, and Palm OS are available. The browsers on the mobile devices have higher capability — the mobile device can download content written in XHTML and present it to the user.

Java-enabled mobile devices will run a KVM to download the Java code and present it to the users. The KVM can be ported on devices with limited capabilities such as mobile phones, Personal Digital Assistants (PDAs), and two-way pagers, giving them the capability to get connected to a wireless network and download the application.

Even after wide deployment of 3G networks, mobile devices with varying capabilities will be used for wireless Internet access. At the lower end of the spectrum are WAP-enabled phones; at the higher end are laptop computers that run a full-fledged operating system and a powerful browser. Developing the content that can cater to all types of mobile devices will be the greatest challenge for the content developers. We discuss these issues in detail.

Mobile devices don't make the fixed devices obsolete. People continue, for example, to use their desktops and fixed peripherals on a daily basis. If a person wants to transfer a file from the laptop to the desktop or to take a printout of a document that's on the laptop, he or she has to connect the devices through wires, and then make them communicate with each other. Another problem is to ensure that the data in various machines are *synchronized*: The mail in the mailboxes on the desktop and the laptop need to be the same, the address books on the desktop and laptop need to be the same, and so on.

This synchronization will achieve greater importance when a person downloads mail from different devices at different times but wants to ensure that both devices have the same copies. Bluetooth solves the problems associated with wires and lack of synchronization of data. Using low-cost, low-power radio technology, Bluetooth enables ad hoc networks to be formed among various devices (desktop, laptop, PDA, mobile phone, headset, LCD projector, printer, scanner, digital camera, and so on). The Personal Area Network (PAN) formed by these devices can exchange data without wires and also synchronize the data among themselves. Because it's likely that every 3G mobile device will be Bluetooth enabled in the future, we devote two chapters to Bluetooth protocols and programming aspects.

Summary

In this chapter, we introduced the various aspects of WAP, Bluetooth, and 3G programming that are explored in this book. For 3G to find wide acceptance by the user community, the content developers have to create exciting applications that have high visibility; this book aims to equip you to achieve this objective. The focus here is on developing commercial applications in mobile advertising, mobile commerce, mobile audio, and video streaming, which can be deployed on present and future wireless networks.

Chapter 2

WML and WML Script Programming: A Case Study

The topmost layer in the WAP (Wireless Application Protocol) architecture is made up of WAE (Wireless Application Environment), which consists of WML and WML scripting language. WML scripting language is used to design applications that are sent over wireless devices such as mobile phones. This language takes care of the small screen and the low bandwidth of transmission. WML is an application of XML, which is defined in a document-type definition. WML is based on HDML and is modified so that it can be compared with HTML.

This chapter focuses on explaining the application of WML and WML Script. We use case studies to explain how to program with WML and WML Script. In order to understand the syntax and commands used in the case studies, you must first understand the commands and their syntax in WML and WML Script.

WML Commands and Syntax

WML commands and syntaxes are used to show content and to navigate between the cards. Developers can use these commands to declare variables, format text, and show images on the mobile phone.

Program Structure

A WML program is typically divided into two parts: the document prolog and the body. Consider the following code:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
```

The first line of this text says that this is an XML document and the version is 1.0. The second line selects the document type and gives the URL of the document type definition (DTD). This DTD gives the full XML definition of WML. The DTD referenced is defined in WAP 1.1, but this header changes with the versions of the WML. The header must be copied exactly so that the tool kits automatically generate this prolog.

The body is enclosed within a `<wml>` `</wml>` tag pair. The body of a WML document can consist of one or more of the following:

- ◆ Deck
- ◆ Card
- ◆ Content to be shown
- ◆ Navigation instructions

This is declared as follows:

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

Commands

The commands used in WML are summarized as follows:

Formatting

<p>	Paragraph
	Bold
<big>	Large
	Emphasized
<i>	Italicized
<small>	Small
	Strongly Emphasized
<u>	Underlined
 	Line Break

Navigation controls

Do	<do>	Anchor link - <a>
Go	<go>	
Prev	<prev>	

Inserting images

```

```

Tables

<table>	Definition of a table
<tr>	Defining a row
<td>	Defining a column
<Thead>	Table header

Variable

Declared as:

```
<setvar name="x" value="123"/>
```

Used as:

```
$ identifier or
$ (identifier) or
$ (Identifier; conversion)
```

Forms

<select>	Define single or multiple list
<input>	Input from user

Events

The various events are as follows:

◆ Do

<do>	To start a specified action

◆ Tasks

<go>	To jump to other possible position
<prev>	To jump to the prev page
<refresh>	To reload the page
<noop>	No operation

WML Script — Commands and Syntaxes

WML Script is used to check the part of the program on the client machine. The functions used are stored in a separate file with the extension `.wmls`. The functions are called as the filename followed by a hash, followed by the function name:

```
currency.wmls#convert()
```

WML Script is function-based. The six main libraries to provide the functionality are:

- ◆ Lang — for core WML Scripts
- ◆ Float — for floating-point math functions
- ◆ String — for String manipulation functions
- ◆ URL — for specialized String manipulation functions for working with URL
- ◆ WMLBrowser — for controlling the browser from the program
- ◆ Dialogs — for controlling dialogs

Program Components

WML Script program components are summarized as follows:

Operators

- ◆ Assignment operator: equal to (=)
- ◆ Arithmetic operators:
 - + Addition
 - - Subtraction
 - * Multiplication
 - / Division
 - Div Integer division only
 - % Modulo
 - ++ Increment
 - -- Decrement
 - ?: Ternary operator
- ◆ Relational and logical operators:
 - == Equality
 - < Less than
 - > Greater than
 - <= Less than or equal to

- `>=` Greater than or equal to
- `!=` Not equal to
- ◆ Logical Operators:
 - `&&` And
 - `||` Or
 - `!` Not
 - `IsValid`: checks whether an expression evaluates to invalid
- ◆ Bitwise operators:
 - `&` Bitwise and
 - `|` Bitwise or
 - `^` Bitwise exclusive or
 - `<<` Left shift
 - `>>` Right shift
 - `>>>` Right shift with zero fill
 - `~` Bitwise not

Control structures

Control structures are used for controlling the sequence and iterations in a program.

- ◆ `if-else` Conditional branching
- ◆ `for` Making self-incremented fixed iteration loop
- ◆ `while` Making variable iteration loop

Functions

The user-defined functions are declared in a separate file having the extension `.wmls`. Functions are declared as follows:

```
function name (parameters)
{ control statements;
return var;
}
```

The function is called as follows:

```
filename#function name
```

Standard Libraries used in WML Scripts

- ◆ `Lang` For WML Script core programming
 - Examples: `abs()`, `abort()`, `characterSet()`, `float()`, `isFloat()`, `isInt()`, `max()`, `isMax()`, `min()`, `minInt()`, `maxInt()`, `parseFloat()`, `parseInt()`, `random()`, `seed()`
- ◆ `Float` For clients having floating-point capabilities
 - Examples: `sqrt()`, `round()`, `pow()`, `ceil()`, `floor()`, `int()`, `maxFloat()`, `minFloat()`
- ◆ `String` For performing string operations

- Examples: `length()`, `charAt()`, `find()`, `replace()`, `trim()`, `compare()`, `format()`, `isEmpty()`, `squeeze()`, `toString()`, `elementAt()`, `elements()`, `insertAt()`, `removeAt()`, `replaceAt()`
- ◆ URL For handling absolute and relative URLs
 - Examples: `getPath()`, `getReferer()`, `getHost()`, `getBase()`, `escapeString()`, `isValid()`, `loadString()`, `resolve()`, `unescapeString()`, `getFragment()`
- ◆ WMLBrowser Used to access WML context by WML Script
 - Examples: `go()`, `prev()`, `next()`, `getCurrentCard()`, `refresh()`, `getVar()`, `setVar()`
- ◆ Dialogs Contains the user interface functions
 - Examples: `prompt()`, `confirm()`, `alert()`

In the previous sections, you saw the structure and syntax of WML and WML Script. Now we will develop two applications and explain their functioning. The goal of the applications is to make the user understand the implementation of the WML and WML Script.

The applications are:

- ◆ The Information Master
- ◆ The Restaurant

These applications are written in WML and the functions are applied to them through WML Scripts. The functioning and output are shown on the screen of a wireless device/cell phone using a Nokia Toolkit as the emulator.

The Information Master Application

The Information Master application deals with providing information about movies and the weather to the client. It's made up of three WML files, one WMLS file, and one graphic file. The script file has just one function for generating the random numbers for the display of maximum and minimum temperatures on the screen.

Application Structure

The case study discussed here contains the following files:

- ◆ `Information.wml`
- ◆ `Movie.wml`
- ◆ `Weather.wml`
- ◆ `Weather.wmls`
- ◆ `sun.ico`

The first three files are the WML application files; the fourth is the script file that will be used by `Weather.wml` file. `sun.ico` is an image file used to display the image of sun on the browser screen by the `Weather.wml` code file.

Application Work Flow

The application opens a menu with two options: Movie and Weather. If the user clicks Movie, the `Movie.wml` file, which displays the name of movies and show timings, appears. If the user clicks Weather, the `Weather.wml` file, which displays the maximum and minimum temperatures of various cities on the screen, appears. These temperatures are generated by using `Lang.rand()` in the script file and by `Weather.wml` to display them on-screen.

To understand how the files are related, see Figure 2-1.

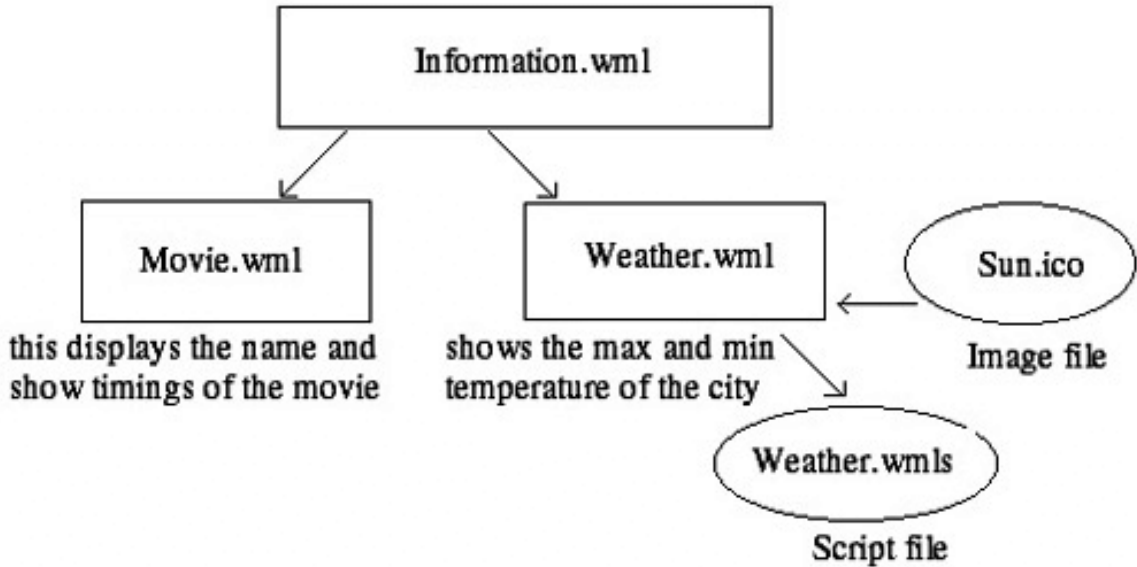


Figure 2-1: Flow diagram of the Information Master application

The main file is `Information.wml`, which shows two options:

- ◆ Movies Info (`Movie.wml`)
- ◆ Weather Info (`Weather.wml`)

On clicking the Movies Info option, the system opens the `Movie.wml` file and shows the listing of the all movies in the selected theatres. On clicking the Weather Info option, the system calls the `Weather.wml` file, which internally calls the `Weather.wmls` file (Scripting file) that initializes the minimum and the maximum temperatures and returns them to the `Weather.wml` file.

Application Description

Listing 2-1 contains the code for `Information.wml`.

Listing 2-1: `Information.wml`

// © 2001 Dreamtech Software India Inc.
// All Rights Reserved

```

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

   <!-- WML prolog-declaration of file type and version>

3. <wml>
   <!-- Declaration of the WML deck>
4. <head>
5. <meta http-equiv="Cache-Control" content="max-age=time" forua="true"/>

   <!-- meta tag to define the content and cache settings>
6. </head>
  
```

```

7. <card id="MyFirst" newcontext="true">
   <!-- declaration of a card in deck>

8. <p align="center"><b>Information Center</b></p>
   <!--paragraph declaration to display heading>
9. <p>
   <!--paragraph declaration to display links>
10.   <a href="Movie.wml">1. Movies info.</a>
11.   <a href="Weather.wml">2. Weather Info.</a>
   <!--declaration of links for weather and movies>
12. </p>
13. </card>
   <!-- card end>
14. </wml>
   <!-- program end>

```

Code description

The first two lines are the prolog of a typical WML file. This WML file contains only one card titled `MyFirst`, which is defined in Line 7. This card contains one text line (Line 8) and two links (which are defined in an `<a>` tag on Lines 10 and 11). By clicking the links, you can navigate to the file `Movie.wml` or `Weather.wml`. Lines 12, 13, and 14 are closing tags for paragraph, card, and WML, respectively.

Code output

Figure 2-2 shows the output of `Information.wml`.

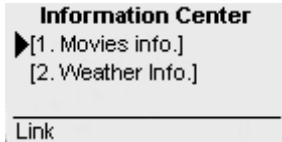


Figure 2-2: Output screen of the `Information.wml`

Listing 2-2 contains the code for `Movie.wml`.

Listing 2-2: `Movie.wml`

// © 2001 Dreamtech Software India Inc.
// All Rights Reserved

```

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

   <!-- WML prolog-declaration of the file type and version>

3.   <wml>
   <!-- Declaration of the WML deck>

4.       <head>
5.       <meta http-equiv="Cache-Control" content="max-age=time"
forua="true"/>
   <!--meta tag to define the content and cache settings>

6.       </head>

```

```

7.      <card id="First" newcontext="true">
      <!-- declaration of a card in deck>

8.      <p align="center"><b>Movies Information</b></p>
      <!-- declaration of a paragraph to display heading>
9.      <p align="center">Cinema 8</p>
10.     <p align="center">1040 South</p>
11.     <p align="center">Sage Way</p>
12.     <p align="center">.....</p>
13.     <p align="center">The Blair Witch Project</p>
14.     <p align="center">4:25, 7:30, 9:15</p>
15.     <p align="center">.....</p>
16.     <p align="center">Inspector Gadget</p>
17.     <p align="center">4:40, 7:35, 9:25</p>
18.     <p align="center">.....</p>
19.     <p align="center">Story of Us</p>
20.     <p align="center">7:10, 9:30</p>
21.     <p align="center">.....</p>
22.     <p align="center">Three to Tango</p>
23.     <p align="center">7:25, 9:30</p>
24.     <p align="center">.....</p>
25.     <p align="center">Three Kings</p>
26.     <p align="center">7:05, 9:20</p>
27.     <p align="center">.....</p>
28.     <p align="center">Super Star</p>
29.     <p align="center">7:25, 9:15</p>
30.     <p align="center">.....</p>
31.     <p align="center">Mystery Alaska</p>
32.     <p align="center">7:05, 9:20</p>
33.     <p align="center">.....</p>
34.     <p align="center">Runaway Bride</p>
35.     <p align="center">7:00, 9:10</p>
      <!-- declaration of paragraphs to display movie information>

36.     </card>
      <!--end of card>

37.     </wml>
      <!--end of wml code>

```

Code description

The first two lines are the prolog of a typical WML file. This WML file contains only one card, named `First`, which is defined in Line 7. Lines 8 to 35 are used to store the information about different movies in multiple paragraphs. Because the information takes up more than one screen, you must use the up and down arrows of the wireless device to read the information. Line 36 is the close tag of the card, and Line 37 is the close tag for WML.

Code output

Figure 2-3 shows the output of `Movie.wml`.

Movies Information	
Cinema 8	The Blair Witch Project
1040 South	4:25, 7:30, 9:15
Sage Way	
Back	Back

Figure 2-3: Output screen of `Movie.wml`

Listing 2-3 contains the code for Weather.wml.

Listing 2-3: Weather.wml

// © 2001 Dreamtech Software India Inc.

// All Rights Reserved

```

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

        <!--WML prolog-declaration of file type and version>

3.      <wml>
        <!--declaration of the wml deck >

4.      <head>
5.      <meta http-equiv="Cache-Control" content="max-age=time"
        forua="true"/>
        <!--meta tag to define the content and cache settings>
6.      </head>
7.      <card id="MyFirst">
        <!-- declaration of the first card>

8.      <onevent type="onenterforward">
        <!-- declaration of the event for navigation>
9.      <go href="Weather.wmls#Init()" />
        <!-- declaration of event and >
<!-- calling of function from the script file >
10.     </onevent>
11.     <p align="center"><b>Weather Information</b></p>
        <!-- declaration of paragraph to display heading>

12.     <p align="center">
        <!-- declaration of paragraph to display sun image>

13.     </p>
14.     <p>
15.     <table align="left" columns="2">
        <!-- declaration of the table element>
16.     <tr>
        <!-- declaration of the first row>
17.     <td><b>High</b></td>
18.     <td><b>Low</b></td>
        <!--declaration of two columns in a row to display High & Low>
19.     </tr>
20.     <tr>
        <!--declaration of the new row to display content>
21.     <td>$var1</td>
22.     <td>$var2</td>
        <!--declaration of the two columns to display the >
<!--value of variables var1 and var2>
23.     </tr>
24.     </table>
        <!--end of table>
25.     </p>
26.     </card>

```

```

        <!-- end of card>
27.    </wml>
        <!--end of WML code>

```

Code description

The table and the scripting language used in this program are a bit complicated. Let's look at the execution line by line:

- ◆ Lines 1–2: WML prolog
- ◆ Lines 3–6: WML declaration with meta tag
- ◆ Line 7: Card declaration
- ◆ Line 8: Event declared (trapping the click event)
- ◆ Line 9: Specifying the navigation link to the init function of the `Weather.wmls` script file
- ◆ Line 10: Event closed
- ◆ Lines 11–14: Paragraph declaration to show the text and image on-screen
- ◆ Line 15: Table declaration having two columns
- ◆ Line 16: First row declaration
- ◆ Lines 17–18: Two columns of first row having text as high and low
- ◆ Lines 21–22: Two columns of second row showing the values of variables defined in init function as of Line 9
- ◆ Lines 23–27: Closing tags

Code output

Figure 2-4 shows the output of `Weather.wml`.

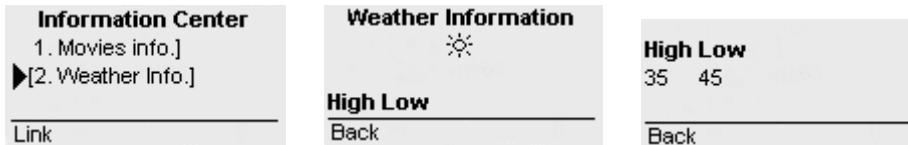


Figure2-4: Output screen of `Weather.wml`

Listing 2-4 shows the code for `Weather.wmls`.

Listing 2-4: Weather.wmls

```
// © 2001 Dreamtech Software India Inc.
// All Rights Reserved
```

```

1      extern function Init()
2      {
3          var var1="10";
4          var var2="15";
5          var Dummy = Lang.seed(-1);
6          var1 = Lang.random(100);
7          var2 = Lang.random(100);
8          WMLBrowser.setVar("var1", var1);
9          WMLBrowser.setVar("var2", var2);
10         WMLBrowser.refresh();
11     }

```

Code description

This is a scripting file and has only one function named `Init`. Take a look at the execution line by line:

- ◆ Line 1: Name of the function
- ◆ Lines 3–4: Initialization of two variables named `var1` and `var2`
- ◆ Lines 6–7: Generation of two random numbers below 100 using the standard WML libraries, and values are stored in the variables `var1` and `var2`. This is called by reference function, so the value of the variable will be stored in these variables and transferred to the calling file, which is `Weather.wml`.
- ◆ Lines 8–9: Stores the values in the variables `var1` and `var2` defined in `Weather.wml`
- ◆ Line 10: Refreshes the screen

Code output

You cannot execute the script file directly. You can only call it from other WML files, so there's no output for this code.

sun.ico

`sun.ico` is the image file, which is called from the weather file to display the information on-screen.

Complete Output

Figure 2-5 shows the project flow of the Information Master application.



Figure 2-5: Output of the complete project

The Restaurant Application

This application starts with a menu from which the user can select different items to order from a restaurant. After the user selects the items, the bill is generated accordingly.

Application Structure

The application is made up of five files:

- ◆ `ResScript.wmls` - The scripting file
- ◆ `Restaurant.wml` - The main menu file to select the category
- ◆ `South.wml` - Link file to select items of the South Indian dishes category
- ◆ `Soft.wml` - Link file to select items of Soft Drinks category
- ◆ `Snacks.wml` - Link file to select items of Snacks category

Application Work Flow

The main file is `Restaurant.wml`, which shows three items:

- ◆ South Indian (`South.wml`)
- ◆ Soft Drink (`Soft.wml`)
- ◆ Snacks (`Snacks.wml`)

After you click a particular item, the system calls the corresponding WML file and shows the item related to that category.

Figure 2-6 shows the flow of data in the files.

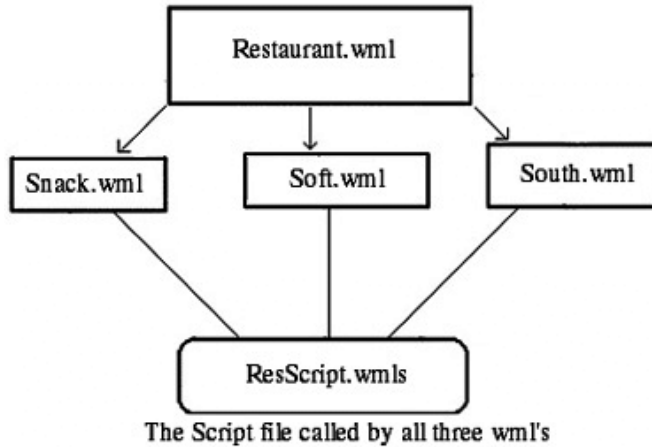


Figure 2-6: Flow diagram of the Restaurant application

Application Description

Listing 2-5 shows the code for `Restaurant.wml`. The main menu file shows a menu containing three links for three categories.

Listing 2-5: Restaurant.wml

// © 2001 Dreamtech Software India Inc.
// All Rights Reserved

```

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

   <!-- WML prolog-declaration of file type and version>
3. <wml>
   <!-- declaration of a new deck>
4. <head>
5. <meta http-equiv="Cache-Control" content="max-age=time" forua="true"/>
   <!-- meta tag to define the content and cache settings>
6. </head>
7. <card id="card1">
   <!-- declaration of a new card>
8. <onevent type="onenterbackward">
   <!-- declaration of an event for navigation>

```

```

9. <go href="ResScript.wmls#Initialize()" />
    <!-- declaration of an event for navigation and calling of>
    <!-- function from the script file>
10. </onevent>
    <!-- end of event declaration>

11. <p align="center"><b> Taj Palace </b>
    <!-- declaration of a new paragraph to display heading>

12. </p>
    <!-- end of paragraph>

13. <p align="left">Select a Category:
    <!-- declaration of a new paragraph to display selection>
    <!-- category>
14. </p>
15. <p align="left" mode="nowrap">
    <!-- declaration of a new paragraph to define table>

16. <table align="left" columns="1">
    <!-- declaration of a table>

17. <tr>
    <!-- declaration of a new row>

18. <td><a href="South.wml">South Indian</a></td>
    <!-- declaration of a new column having link to south.wml>

19. </tr>
    <!-- end of a row>
20. <tr>
    <!-- declaration of a new row >
21. <td><a href="Soft.wml">Soft Drink</a></td>
    <!-- declaration of a new column having link to soft.wml>

22. </tr>
    <!-- end of row>
23. <tr>
    <!-- declaration of a new row>

24. <td><a href="Snacks.wml">Snacks</a></td>
25. </tr>
    <!-- end of row>

26. </table>
    <!-- end of table>
27. </p>
28. </card>
    <!-- end of card>
29. </wml>
    <!-- end of wml>

```

Code description

- ◆ Lines 1–2: WML Prolog
- ◆ Lines 3–5: Meta tag on head

- ◆ Line 7: Defining the first card named `card1`
- ◆ Line 8: Event declared (trapping the click event)
- ◆ Line 9: Specifying the navigation link to the initialize function of the `ResScript.wmls` script file
- ◆ Line 10: Event closed
- ◆ Lines 11–12: Paragraph declaration to display Taj Palace in center
- ◆ Lines 13–15: Paragraph to display selection of category
- ◆ Lines 16–26: Creation of table with one column and three rows, each row containing the link to a new category. After you select the link, navigation takes you to a new file
- ◆ Lines 27–29: Closing tags

Code output

Figure 2-7 shows the output of `Restaurant.wml`.

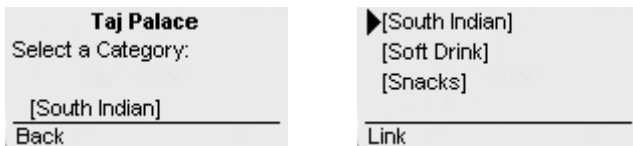


Figure 2-7: Output screen of the file `Restaurant.wml`

Listing 2-6 shows the code for `South.wml`. This file shows the items belonging to the South Indian category. The user can select an item from the list. The price of that item is passed as a parameter to the function `Func1` of the script file `ResScript.wmls`.

Listing 2-6: South.wml

// © 2001 Dreamtech Software India Inc.
// All Rights Reserved

```

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

   <!-- WML prolog-declaration of file type and version>
3. <wml>
   <!-- declaration of a new deck>
4. <head>
5. <meta http-equiv="Cache-Control" content="max-age=time" forua="true"/>
   <!-- meta tag to define the content and cache settings>

6. </head>
7. <card id="card1">
   <!-- declaration of a new card>

8. <onevent type="onenterforward">
   <!-- declaration of a new event>
9. <refresh>
   <!-- refreshing the variable>

10. <setvar name="var1" value="1.00" />
   <!-- declaration of a variable and initialising it>
11. </refresh>
12. </onevent>
   <!-- end of event>

```

```

13. <do type="accept">
    <!-- declaration of an event>

14. <go href="ResScript.wmls#Func1($var1)" />
    <!-- declaration of navigation to script file to >
    <!-- call the function func1 with parameter>

15. </do>
    <!-- end of event>
16. <p>
    <!-- declaration a new paragraph>
17. <select name="var1">
    <!-- declaration of a select list>
18.         <option value="23.00">Onion pancake 23.00</option>
19.         <option value="15.00">Rice pancake 15.00</option>
20.         <option value="15.00">Rice ball 15.00</option>
21.         <option value="10.00">Cheese pancake 10.00</option>
22.         <option value="30.00">Mixed stew 30.00</option>
23.         <option value="20.00">Rice doughnut 20.00</option>
        <!--options of select list>

24.     </select>         <!-- end of select list>
25. </p>
    <!-- end of paragraph>
26. </card>
27. <!-- end of card>
28. </wml>
29. <!-- end of deck>

```

Code description

- ◆ Lines 1–2: WML Prolog
- ◆ Lines 3–5: Meta tag on head
- ◆ Line 6: End of the head tag
- ◆ Line 7: Defining the first card named card1
- ◆ Line 8: Event declared (trapping the click event)
- ◆ Line 9: Specifying the navigation link to initialize the function of the ResScript.wmls script file
- ◆ Line 10: Defining a variable var1 and initializing it to 1
- ◆ Line 11: End of the refresh tag
- ◆ Line 12: Event closed
- ◆ Lines 13–15: Calling the function Func1 from ResScript.wmls with the value of variable var1 on select option
- ◆ Line 16: New paragraph
- ◆ Lines 17–23: Creating the selection list having items belonging to the South Indian category
- ◆ Lines 24–29: Closing tags

Code output

Figure 2-8 shows the output of South.wml.

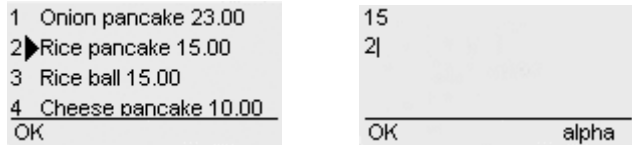


Figure 2-8: Output screen of South.wml

Listing 2-7 shows the code for Soft.wml. This file shows the items belonging to the Soft Drinks category and selects an item from the list. The price of that item is passed as a parameter to the function func1 of the script file ResScript.wmls.

Listing 2-7: Soft.wml

// © 2001 Dreamtech Software India Inc.

// All Rights Reserved

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1/EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
   <!-- WML prolog-declaration of file type and version>
3. <wml>
   <!-- declaration of a new deck>
4. <head>
5. <meta http-equiv="Cache-Control" content="max-age=time" forua="true"/>
   <!-- meta tag to define the content and cache settings >
6. </head>
7. <card id="card1">
   <!-- declaration of a new card>
8. <onevent type="onenterforward">
   <!-- declaration of a new event>
9. <refresh>
   <!-- refreshing the variables>
10.   <setvar name="var1" value="1.00" />
   <!-- declaring a variable and setting the initial value>
11. </refresh>
   <!-- end of refresh>
12. </onevent>
   <!-- end of event declaration>
13. <do type="accept">
   <!-- declaration of action event>
14.   <go href="ResScript.wmls#Func1($var1)" />
   <!-- declaration of action to call the function from>
   <!-- script file with a parameter>
15. </do>
   <!-- end of action tag>
16. <p>
   <!-- declaration of a new paragraph>
17. <select name="var1">
   <!-- declaration of a select list>
18.   <option value="40.00">Cold Coffee 40.00</option>
19.   <option value="20.00">Coffee 20.00</option>
20.   <option value="20.00">Cold Drink 20.00</option>
21.   <option value="10.00">Tea 10.00</option>
22.   <option value="50.00">Cold Coffee with ice-cream 50.00</option>
   <!-- options of select list>
23. </select>

```

```

24.      <!-- end of select list>
      </p>
      <!-- end of paragraph>
25.    </card>
      <!-- end of card>
26.    </wml>
      <!-- end of deck>

```

Code description

- ◆ Lines 1–2: WML Prolog
- ◆ Lines 3–5: Meta tag on head
- ◆ Line 7: Defining the first card named `card1`
- ◆ Line 8: Event declared (trapping the click event)
- ◆ Line 9: Specifying the navigation link to initialize function of the `ResScript.wmls` script file
- ◆ Line 10: Defining a variable `var1` and initializing it to 1
- ◆ Line 12: Event closed
- ◆ Lines 13–15: Calling the function `func1` from `ResScript.wmls` with the value of variable `var1` on select option
- ◆ Line 16: New paragraph
- ◆ Lines 17–23: Creating the selection list containing items belonging to the Soft Drinks category
- ◆ Lines 24–26: Closing tags

Figure 2-9 shows the output of `Soft.wml`.

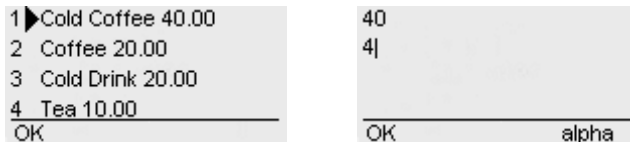


Figure 2-9: Output screen of `Soft.wml`

Listing 2-8 shows the output for `Snacks.wml`. This file shows the items belonging to the Snacks category. The user can select an item from the list. The price of that item is passed on as a parameter to the function `Func1` of the script file `ResScript.wmls`.

Listing 2-8: `Snacks.wml`

// © 2001 Dreamtech Software India Inc.
// All Rights Reserved

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
   <!-- WML prolog-declaration of file type and version>
3. <wml>
   <!-- declaration of a deck>
4. <head>
5. <meta http-equiv="Cache-Control" content="max-age=time" forua="true"/>
   <!-- meta tag to define the content and cache settings >
6. </head>
7. <card id="card1">
   <!-- declaration of a card>
8. <onevent type="onenterforward">

```

```

    <!-- declaration of an event>
9. <refresh>
    <!-- refereshing the variable>
10.   <setvar name="var1" value="1.00" />
        <!-- declaration of a variable and storing initial value>
11.   </refresh>
12.   </onevent>
        <!-- end of event declaration >
13.   <do type="accept">
        <!-- declaration of an action>
14.     <go href="ResScript.wmls#Func1($var1)" />
        <!-- declaration of an action and calling the function >
<!-- from script file>
15.     </do>
16.     <p>
        <!-- declaration of a new paragraph to display >
<!-- select list>
17.     <select name="var1">
        <!-- declaration of a select list>

18.       <option value="40.00">Cutlet 40.00</option>
19.       <option value="30.00">Cheese Fry 30.00</option>
20.       <option value="20.00">Bread Pakora 20.00</option>
21.       <option value="25.00">Mix. Veg. Pakora 25.00</option>
22.       <option value="30.00">Onion Pakora 30.00</option>
        <!-- list items>
23.     </select>
        <!-- end of select list>
24.     </p>
        <!-- end of paragraph>
25.   </card>
        <!-- end of card>
26. </wml>
    <!-- end of deck>

```

Code description

- ◆ Lines 1–2: WML Prolog
- ◆ Lines 3–5: Meta tag on head
- ◆ Line 7: Defining the first card named `card1`
- ◆ Line 8: Event declared (trapping the click event)
- ◆ Line 9: Specifying the navigation link to initialize the function of the `ResScript.wmls` script file
- ◆ Line 10: Defining a variable `var1` and initializing it to 1
- ◆ Line 12: Event closed
- ◆ Lines 13–15: Calling the function `func1` from `ResScript.wmls` with the value of variable `var1` on select option
- ◆ Line 16: New paragraph
- ◆ Lines 17–23: Creating the selection list containing items belonging to the Snacks category
- ◆ Lines 24–26: Closing tags

Code output

Figure 2-10 shows the output of `Snacks.wml`

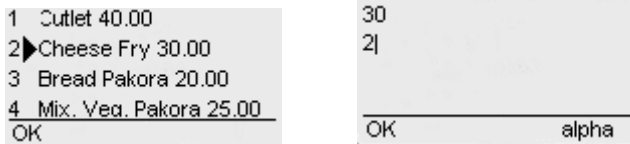


Figure 2-10: Output screen of snacks.wml

Listing 2-9 shows the code for ResScripts.wmls. The script file has two functions. Func1 is called from three link files to calculate the amount of the bill, and the function Initialize is called from the restaurant file to set the initial value of the browser variable TotalAmount to 0.

Listing 2-9: ResScripts.wmls

// © 2001 Dreamtech Software India Inc.

// All Rights Reserved

```

1. extern function Func1(Price){
    // declaration of function func1
2. var ITEM="0";
    // declaring a new ITEM variable and setting to 0

3. var TAmount=WMLBrowser.getVar("TotalAmount");
    // declaring a new TAmount variable and setting it to
    // the value of variable TotalAmount

4. ITEM = Dialogs.prompt("Enter Quantity", ITEM);
    // declaring a dialog to take input from user

5. var Amount = ITEM * Lang.parseFloat(Price);
    // declaring a new Amount variable and setting it to Price

6. TAmount = TAmount + Amount;
    // adding amount to TAmount

7. WMLBrowser.refresh();
    // refresh the browser

8. ITEM = Dialogs.prompt("Your Total Bill", TAmount);
    // declaring a dialog to display amount on screen

9. WMLBrowser.setVar("TotalAmount",TAmount);
    // setting the variable TotalAmount to TAmount

10. WMLBrowser.go("Restaurant.wml#card1 ");
    // setting the navigation to restaurant's card1

11. }

    // declaration of function initialize

12. extern function Initialize()
13. {
14. WMLBrowser.setVar("TotalAmount","0.00");
    // setting the variable TotalAmount to 0
    }
    
```


Code description

This is a script file, the function of which is to calculate the amount and generate the bill accordingly. The main calculations are done through this file. The following breakdown shows the functioning of this file in detail:

- ◆ Line 1: Defining the first function named `Func1`, which takes price as parameter
- ◆ Line 2: Defining a variable `ITEM` and initializing it to 0
- ◆ Line 3: Reading the browser variable `TotalAmount` and storing the result into local variable `TAmount`
- ◆ Line 4: Showing a prompt on-screen and accepting the value for quantity and storing it in the variable `ITEM`
- ◆ Line 5: Calculating the amount by multiplying the `ITEM` quantity with price received as parameter
- ◆ Line 6: Adding the amount calculated to `TAmount` variable
- ◆ Line 7: Refreshing the browser
- ◆ Line 8: Displaying the bill amount on browser
- ◆ Line 9: Setting the value of browser variable `TotalAmount` to local variable `TAmount`
- ◆ Line 10: Taking navigation control back to main `Restaurent.wml` files's `card1`
- ◆ Line 11: Function closed
- ◆ Line 12: Defining of second function named `Initialize`, which takes no parameter
- ◆ Line 13: Sets the browser variable `TotalAmount` to 0. This function is called when the application starts functioning

Complete output

Figure 2-11 shows the complete output of the Restaurant application.

Taj Palace Select a Category: [South Indian] Back	▶[South Indian] [Soft Drink] [Snacks] Link	1 Onion pancake 23.00 2▶Rice pancake 15.00 3 Rice ball 15.00 4 Cheese pancake 10.00 OK
15 2 OK alpha	Your Total Bill 30.00 OK alpha	[South Indian] ▶[Soft Drink] [Snacks] Link
1▶Cold Coffee 40.00 2 Coffee 20.00 3 Cold Drink 20.00 4 Tea 10.00 OK	40 4 OK alpha	Your Total Bill 190.00 OK alpha



Figure 2-11: Execution of The Restaurant application

Summary

In this chapter, you found out how WML offers software developers a new, exciting platform on which they can deploy their applications. The WML and WML Scripts syntax and commands are discussed briefly. Our objective was to give the user some insight into WML and WML Script by using real applications.

The main objective of the study of the two applications (Information Master and Restaurant) was to provide users with an understanding of WML and WML Script implementation in an application. We studied client-side programming by using WML and WML Script. For more information on WML and WML Script programming refer to the following resources:

Books

- ◆ Arehart, C. *et al*, *Professional WAP*, Wrox Press, 2000.
- ◆ Cook, J. L. *WAP Servlets: Developing Dynamic Web Content with Java and WML*, John Wiley & Sons, Inc., 2000.
- ◆ Foo, S. (Ed.), *Beginning WAP: Wireless Markup Language and Wireless Markup Language Script*, Wrox Press, 2000.

Links

- ◆ <http://cellphones.about.com>
- ◆ <http://www.wapforum.org/>
- ◆ http://www.zvon.org/ZvonHTML/Zvon/zvonTutorials_en_etc.
- ◆ <http://www.palowireless.com/wap/wml.aspWMLScript.com>
- ◆ <http://www.wirelessdevnet.com/channels/wap/training/wmlscript.html>

Chapter 3

WAP Using Cold Fusion: A Project

The last chapter dealt with WML and WML Script. Both of these are client-side application development languages used in wireless applications. This chapter emphasizes wireless application using Cold Fusion. Cold Fusion, developed by Allaire, runs concurrently with most Windows and Solaris Web servers. It's used to create forms and dynamic pages because all the benefits of languages, such as CGI and ASP, are provided in it in a very simple way. In this chapter, you will learn the fundamentals of Cold Fusion and see its use in wireless applications through a simple project.

Cold Fusion: An Overview

Cold Fusion is used to develop Web-based applications. If a client needs a static Web page, HTML is needed. But if the user wants a dynamic and interactive Web page (which can be accessed from any device such as a Palm, Mobile, or Desktop) Cold Fusion is the solution because Cold Fusion changes the output according to the user's needs. Cold Fusion contains the following technologies:

- ♦ **Language:** CFML (Cold Fusion Markup Language) — a dynamic language for the Web
- ♦ **Server:** Cold Fusion Server — residing on the same server as the Web server
- ♦ **Application:** Cold Fusion Studio — a rapid development environment (RAD)

A page written in CFML must pass through a Cold Fusion Server in order to be viewed properly.

Structure of Cold Fusion

Like HTML, Cold Fusion is also a tag-based language. But Cold Fusion tags start with CF and are processed on the server machine. You can later embed the Cold Fusion tags between the HTML tags when necessary. Also similar to HTML, Cold Fusion tags have attributes and a list of the available values for those attributes.

The complete list of Cold Fusion tags, along with a brief explanation, is provided in the following table.

Category	Tag	Function
Variable manipulation tags	CFSET	Sets the variable
	CFPARAM	Sets the parameter
	CFCOOKIE	Sets the cookie
	CFSCCHEDULE	Sets a schedule
Flow control tags	CFABORT	Stops the processing of the page
	CFBREAK	Breaks out of the CF loop
	CFEXECUTE	Facilitates execution of the process on the server machine

	CFLOCATION	Opens a specified Cold Fusion or HTML page
	CFTHROW	Throws the exception specified by the users
	CFLOOP	Makes loops (for example, for loop)
	CFIF CFELSE CFELSEIF	Checks a specified condition
	CFSWITCH	Makes a switch construct; checks cases of an expression
	CFTRY CFCATCH	For exception handling in a Cold Fusion page
Cold Fusion form tags	CFFORM	Defines a form in CFML
	CFSELECT	Defines a drop-down list
	CFTREE	Adds tree control in form
	CFGRID	Adds grid control in form
	CFGRIDROW	Specifies individual row data in CFGRID of form
	CFGRIDCOLUMN	Specifies individual column data in CFGRID of form
	CFTEXTINPUT	Enters a single-line text entry box in form
Database manipulation tags	CFINSERT	Inserts a new record in data source
	CFQUERY	Passes an SQL statement to the data source
	CFUPDATE	Updates the existing record in a database
	CFPROCPARAM	Specifies the information on the parameter
	CFPROCRESULT	Specifies a name to the result set obtained from a query result
	CFTRANSACTION	Groups multiple queries into one
Data output tags	CFCOL	Defines the table column header, width, alignment, and the text used inside CFTABLE
	CFCONTENT	Defines the mime type of the page
	CFHEADER	Generates custom http response header
	CFTABLE	Builds a table in Cold Fusion
	CFOUTPUT	Displays the result on-screen
File management tags	CFDIRECTORY	Sets interactions with directories
	CFFILE	Sets interactions with files

Other tags	CFCACHE	Speeds up static pages access
	CFLOCK	Sets the locks to database
	CFSILENT	Suppresses the outputs
	CFREPORT	Runs predefined reports
	CFINCLUDE	Includes a CFML page reference in current document
	CFASSOCIATE	Allows the sub tags in the base tag
	CFSETTING	Helps control the page setting aspects
	CFHTMLHEAD	Embeds other HTML tags in CFML

Cold Fusion for WAP

Cold Fusion pages are configured for the normal Web server with the extension `.cfm`. You can also define Cold Fusion pages for use by the WAP client. The main principle lies in the specification of the content type in a file being worked on in the client browser. The specific tag used for this purpose is `<CFCONTENT>`.

Specifying content type for WAP

`<CFCONTENT>` is the main tag is used for developing a WAP application with Cold Fusion. This tag should be at the beginning of the Cold Fusion template that contains the normal WML code. The tag is written as follows:

```
<CFCONTENT TYPE= "text/vnd.wap.wml">
```

For generating WML Script decks, the main tag is used as follows:

```
<CFCONTENT TYPE= "text/vnd.wap.wmlscript">
```

If the page is without any dynamic content, the WAP client uses `.wml` and `.wmls` files to access the content from the Web server.

Displaying text in the WAP browser/Commands summary

For displaying the text in the WAP browser, you use the Cold Fusion `<CFOUTPUT>` tag. The output can appear as it's completed in HTML (such as in paragraphs or a table). The commands related to the output are summarized as follows:

Formatting Tags Used in Cold Fusion

<code><p></code>	Paragraph
<code></code>	Bold
<code><big></code>	Large
<code></code>	Emphasized
<code><i></code>	Italicized
<code><small></code>	Small
<code></code>	Strongly Emphasized
<code><u></code>	Underlined
<code>
</code>	Line Break

Navigation Controls in Cold Fusion

Do	<code><do></code>	Anchor link - <code><a></code>
Go	<code><go></code>	
Prev	<code><prev></code>	

Putting Images

```

```

Tables

<table>	Definition of a table
<tr>	Defining a row
<td>	Defining a column
<Thead>	Table header

Variable

Declared as

```
<setvar name="x" value="123"/>
```

Used as

```
$ identifier or
$ (identifier) or
$ (identifier; conversion)
```

Forms

<select>	Define single or multiple list
<input>	Input from user

Events

<do>	To start a specified action

Tasks

<go>	To jump to other possible position
<prev>	To jump to the prev page
<refresh>	To reload the page
<noop>	No operation

Application: Question Quiz

The main objective of this chapter is to explain the function of Cold Fusion in a real-life application. In the preceding section, we discussed the commands and syntax that you use in the application. This chapter, though not meant to teach you Cold Fusion, provides an overview so that you understand its use and also an application for creating a WAP application.

The following sections discuss the application in detail. First, we cover the application structure and its function. Then we look at the code design and description. Finally, you see the output as it appears on the WAP browser.

Consider a CFML designed for playing a quiz game with the mobile user. In this application, the users register to play the quiz by their names, street addresses, and e-mail addresses. Having registered, the user logs in to play the quiz. Now as the username and password are accepted, a random question is picked up from the database depending on the course ID; then the mobile user picks one from several multiple-choice answers. After entering the correct answer, the user is greeted; otherwise, the retry option begins to work. For this application, ten Cold Fusion files and one database file are created in Microsoft Access.

Structure of the application

The application contains the following files:

- ◆ index.cfm
- ◆ action.cfm
- ◆ login.cfm
- ◆ checkvalues.cfm
- ◆ readallvalues.cfm
- ◆ questiondisplay.cfm
- ◆ submit.cfm
- ◆ answer.cfm
- ◆ bingo.cfm
- ◆ tryagain.cfm

The Database File Name is `prototype.mdb`, and the tables defined in the Database are as follows:

- ◆ answertable
- ◆ coursetable
- ◆ questiontable
- ◆ userlogininfo
- ◆ userpersonalinfo

Tables 3-1 to 3-5 describe the structure of each of these tables.

Table 3-1: answertable Structure

<i>Field Name</i>	<i>Datatype</i>	<i>Description</i>
Qid	Number	Contains the question ID
choice1	Text	Contains the Choice Number 1 for the question
choice2	Text	Contains the Choice Number 2 for the question
choice3	Text	Contains the Choice Number 3 for the question
Answerid	Text	Contains the Right Option Number for the question

Table 3-2: coursetable Structure

<i>Field Name</i>	<i>Datatype</i>	<i>Description</i>
Qno	Number	Contains the total number of questions

Table 3-3: questiontable Structure

<i>Field Name</i>	<i>Datatype</i>	<i>Description</i>
Qid	Number	Contains the question ID
Question	Text	Contains the text of the question

Table 3-4: userlogininfo Structure

<i>Field Name</i>	<i>Datatype</i>	<i>Description</i>

Userid	Text	Contains the user login name
Password	Text	Contains the user password

Table 3-5: userpersonalinfo Structure

Field Name	Datatype	Description
Indexno	AutoNumber	Contains the Unique ID for every row in the database, which generat automatically
Userid	Text	Contains the user login name
Username	Text	Contains the user name (actual name)
Cellno	Text	Contains the user cell phone number
Country	Text	Contains the name of the country
City	Text	Contains the name of the city
Provider	Text	Contains the name of the service provider of mobile service
Stdcode	Number	Contains the std code of the city in which the user resides
Emailid	Text	Contains the email address of the user

Application Function

The application begins with the file `index.cfm`, which displays a menu on the mobile screen with two options — Login and Register. The new user must click the Register option before Login to execute the quiz application. The file `index.cfm` is utilized to take this Register information for the user. The information received is `userid`, `password`, `username`, `cellno`, `city`, `country`, and e-mail address. All this information is sent to the `action.cfm` file from which the data is inserted into the table. If the user ID is already present, the user is asked to enter the user ID again. After entering the information in the database, the control is sent to the file `login.cfm`. For facilitating the login for the quiz, the file takes the `username` and `password`. The `username` and `password` are passed onto the file `checkvalue.cfm`, which checks the user password. If the user ID and password are correct, the file `readallvalues.cfm` is called for the question and answer reading. If it's incorrect, the execution is sent back to `login.cfm`. `readallvalues.cfm` reads the total number of questions in the database for the specified course and generates a random number to pick a question from the table. From here, the file `questiondisplay.cfm` is called, which reads the random question, picks an answer from the choices of the question table, and displays it on the mobile screen. The correct answer option is also picked up from the user through the next file, named `submit.cfm`, and this particular answer choice is transferred to `answer.cfm`. Here, it is validated with the correct answer accessed from the database. If the answer is correct, the file called is `bingo.cfm`; otherwise, the file called is `tryagain.cfm`. From `tryagain.cfm`, the file `questiondisplay.cfm` is called. From the file `bingo.cfm` again, the execution goes back to `index.cfm`.

Figure 3-1 shows a diagram of this application.

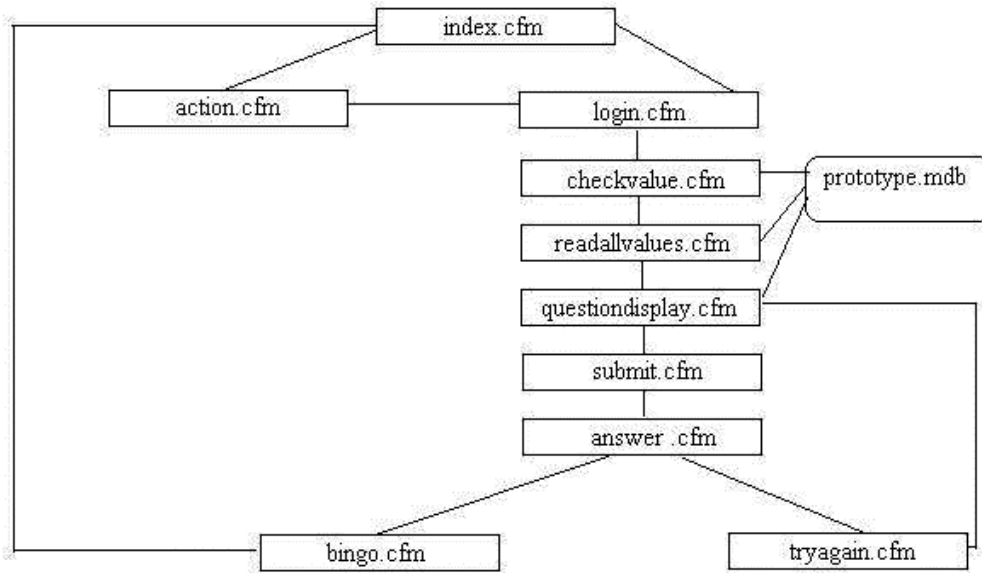


Figure 3-1: Question Quiz application work flow diagram

The following section is a description of each file in the application.

The `index.cfm` file's (Listing 3-1) main function is to create the session variables and display the menu on the mobile screen, so that the user can choose between Register and Login. If Register is selected, the user information is accepted and stored in the session variables. The line-by-line description of the code follows.

Listing 3-1: Index.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <CFCONTENT TYPE="text/vnd.wap.wml">

   <!-- this line is placed for indicating the type of page being sent to the
   browser -->.<!-- this line must be added to the top of the Cold Fusion
   template. -->
2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN" "
   http://www.phone.com/dtd/wml11.dtd" >
   <!-- WML Prolog -->
4. <wml>
   <!-- begin a new deck -->
5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
   <!-- it defines the meta tag to be used by the browser -->
7. </head>
8. <card id="card1" title="Main Menu" newcontext="true">
   <!--defining the first card named as card1, to display the menu -->
   <!--to define the session variables for the entire application. -->
9. <CFSET Session.courseid="0">
   <!--declaration of session variable courseid -->
10. <CFSET Session.render="0">
   <!-- declaration of session variable render -->

```

```

11. <CFSET Session.userid="abc">
    <!-- declaration of session variable userid -->
12. <CFSET Session.userpassword="abc">
    <!-- session variables declaration -->

13. <p align="center"> Main Menu</p>
    <!-- declaration of a new paragraph to display heading -->
14. <p mode="nowrap">
15. <select name="url">
    <!-- declaration of a new list for register and login -->
16. <option onpick="#entry">Register</option>
17. <option onpick="login.cfm">Login</option>
    <!-- list options -->
18. </select>
    <!-- end of list -->
19. </p>
    <!-- end of paragraph -->
20. </card>
    <!-- end of card -->
21. <card id="entry" title = "Record Entry" newcontext="true">
    <!-- declaration of a new card -->
22. <do type="accept" label="ok">
    <!-- declaration of new event -->
23. <go href="action.cfm" method="post" >
    <!-- declaration of new action -->
24. <postfield name="userid" value="$(userid)" />
    <!-- declaration of variables to be posted to new link -->
25. <postfield name="password" value="$(password)" />
26. <postfield name="username" value="$(username)" />
27. <postfield name="cellno" value="$(cellno)" />
28. <postfield name="country" value="$(country)" />
29. <postfield name="city" value="$(city)" />
30. <postfield name="emailid" value="$(emailid)" />
    <!-- declaration of variables to be posted to new link -->
31. </go>
32. </do>
33. <p>
    <!-- declaration of a new paragraph -->
34. Enter User ID <input name="userid" />
35. Enter Password <input name="password" type="password" />
36. Enter User Name <input name="username" />
37. Enter cell No. <input name="cellno" />
38. Enter country <input name="country" />
39. Enter city <input name="city" />
40. Email- ID of user <input name="emailid"/>
41. </p>
42. <CFSET Session.answerid1="0">
43. <do type="prev" label="Back">
    <!-- declaration of action -->
44. <prev/>
45. </do>
46. </card>
    <!-- end of card -->
47. </wml>
    <!-- end of deck -->

```

Code description

This preceding code shows the menu as a selection list named as a URL on-screen. If the Register option is selected, navigation is transferred to the second card in the same deck named as entry. In that card, userid, password, user name, cellno, country, city, and mail id are accepted and stored in the table. From here, the user is again taken to the file login to work on the application. After selecting the Login option, control is carried over to another file named `login.cfm`.

Code output

Figure 3-2 shows the output of `index.cfm`.



Figure 3-2: Output screens of `index.cfm`

The file in Listing 3-2 is used to insert the records in the database. In this program, we employed the `sql` query to search for the already existing data and also to add records in the database.

Listing 3-2: `action.cfm`

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <CFCCONTENT TYPE="text/vnd.wap.wml">
   <!-- declaration of page type to send to the browser -->

2. <cfquery name="abcd" datasource="prototype" dbtype="ODBC">
   <!-- declaration of query -->

3. SELECT      userid AS abc
4. FROM        userlogininfo

```

```

5. WHERE      (userid = '#userid#')
               <!--query -->
6. </cfquery>
               <!-- end of query -->
7. <?xml version="1.0"?>
8. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
               <!-- wml prolog -->

9. <wml>
               <!-- beginning of a new deck -->
10. <head>
11. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
               <!-- meta tag declaration -->

12. </head>
13. <card id="checking">
<!-- beginning of a new card to check for the presence of user -->
<!--
               id -->

14. <p>
15. <CFIF abcd.recordcount NEQ 0 >
               <!-- checking the condition -->

16. User ID Already Found Please Choose Again
17. <do type="accept" label="Back">
               <!-- declaration of action event -->

18. <go href="index.cfm" />
19. </do>
               <!-- checking else part of condition -->
20. <CFELSEIF abcd.recordcount IS 0>
21. <cfinsert datasource="prototype" tablename="userpersonalinfo"
   dbtype="ODBC" formfields="userid, username, cellno, country, city,
   emailid">
               <!-- adding record to the userpersonalinfo for new user -->
22. <cfinsert datasource="prototype" tablename="userlogininfo" dbtype="ODBC"
   formfields="userid, password">
               <!-- adding record to the userlogininfo for new user login --
>

23. <cflocation url="login.cfm" addtoken="No">
               <!-- redirecting to login.cfm file -->

24. </CFIF>
               <!-- end of if condition -->

25. </p>
               <!-- end of paragraph -->

26. </card>
               <!-- end of card -->

27. </wml>
               <!-- end of wml -->

```

Code description

- ◆ Line 1 is used for indicating the content type being sent to the browser.
- ◆ Lines 2 – 6 are employed when you need to define and execute a query to search for the record in the database containing the DSN name as prototype for the given userid.
- ◆ Line 7 defines the WML prolog.
- ◆ Line 11 defines the meta tag used by the browser.
- ◆ Line 15 – In the first card, if condition is applied to check the number of records passed through the query specified in Line 2. If the value is greater than 0, it indicates that the inputted user ID is already present in the table, and consequently the message is displayed to go back to the main menu (as shown in the Figure 3.3).

If the query returns zero, it means that this particular record does not exist in the database and is now ready to be added to the database. In our example, because the user Deepesh was not registered, the record is added to the table.

Code output

Figure 3-3 shows the output of `action.cfm`.

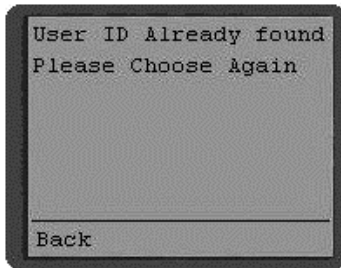


Figure 3-3: Output screen of `action.cfm`

The `Login.cfm` file (Listing 3-3) enables the user to log in after the user is registered. From here, control is passed to `checkvalue.cfm` (Listing 3-4) with the values of variables, `userid`, and `password`.

Listing 3-3: login.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <CFCONTENT TYPE="text/vnd.wap.wml">
   <!-- declaration of page type to send to the browser -->

2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
   <!-- WML Prolog -->

4. <wml>
                                   <!-- new deck -->

5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
                                   <!-- meta tag declaration -->

7. </head>

```

```

8. <card id="entervalue" newcontext="true" title="Login">
    <!-- declaration of new card -->

9. <do type="accept" label="ok">
    <!-- declaration of action -->

10. <go href="checkvalue.cfm" method="post" >
    <!-- declaration of the navigation on event -->

11. <postfield name="userid" value="$(userid)" />
12. <postfield name="password" value="$(password)" />
    <!-- declaration of the fields to be transferred to the new file -->
13. </go>
14. </do>
15. <p align="center">
    <!-- declaration of new paragraph for login -->

16. <b>Login Page: </b>
17. </p>
18. <p>
    <!-- declaration of new paragraph for userid -->

19. User Name <input name="userid"/>
20. </p>
21. <p align="center">
22. <b>Login Page: </b>
23. </p>
    <!-- end of paragraph -->

24. <p>
25. Enter password <input name="password" type="password"/>
    <!-- declaration of new paragraph for password entry -->

26. </p>
    <!-- end of paragraph -->

27. </card>
    <!-- end of card -->

28. </wml>
    <!--end of deck -->

```

Code output

Figure 3-4 shows the output of Login.cfm.

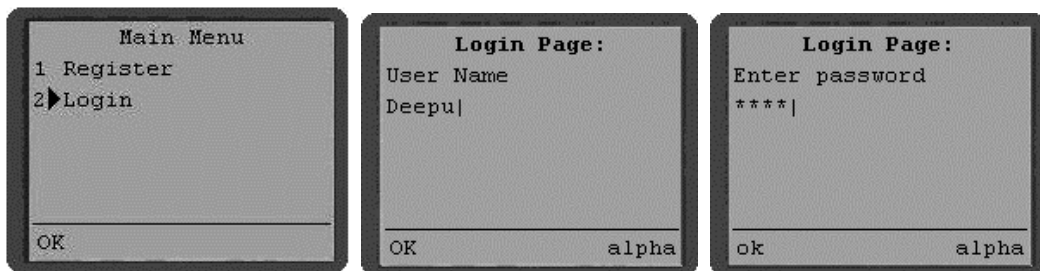


Figure3-4: Output Screen of Login.cfm

The file in Listing 3-4 validates the user ID and the password in the database file. The query is defined to search for the record in the database containing the DSN name as prototype for the password entered along with the user ID. The <CFIF> condition is used to check the number of records passing through the query specified in Line 2. If it's 0, then the inputted user ID is not present in the table, and the message appears to enter the user ID again.

Listing 3-4: checkvalue.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. <CFCONTENT TYPE="text/vnd.wap.wml">
    <!-- declaration of page type to send to the browser -->

2. <cfquery name="abcd" datasource="prototype" dbtype="ODBC">
3. SELECT      userid AS abc
4. FROM        userlogininfo
5. WHERE       (userid = '#userid#')
6. </cfquery>
    <!-- // declaration of a query to read userid -->
7. <CFSET Session.userid="#abcd.abc#">
8. <cfquery name="match" datasource="prototype" dbtype="ODBC">
9. SELECT      password AS passtemp
10. FROM       userlogininfo
11. WHERE      (userid = '#userid#' AND password = '#password#')
12. </cfquery>
    <!-- declaration of a query to read password -->
13. <CFSET Session.userpassword="#match.passtemp#">
    <!-- declaration of a the session variable -->

14. <?xml version="1.0"?>
15. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
    <!-- declaration of WML Prolog -->

16. <wml>
    <!-- declaration of the deck -->

17. <head>
18. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
    <!-- declaration of meta tag -->

19. </head>
20. <card id="card1" >
    <!-- declaration of a new card →

21. <p>
    <!-- declaration of a new paragraph -->

22. <CFIF abcd.recordcount IS 0>
    <!-- declaration of if condition -->

23. Your Record Is not Found Please enter again
24. <cflocation url="login.cfm" addtoken="No">
    <!-- declaration 'to' navigation to new file -->

25. <CFELSEIF match.recordcount IS 0>
```

```

        <!-- checking else part of if condition-->

26. Your passwod Is not Found Please enter again
27. <cflocation url="login.cfm" addtoken="No">
        <!-- declaration 'to' navigation to new file -->

28. </CFIF>
        <!-- end of if condition          -->

29. <cflocation url="readallvalue.cfm" addtoken="No">
        <!-- declaration 'to' navigation to new file -->

30. </p>
        <!-- end of paragraph          -->

31. </card>
        <!-- end of card                -->

32. </wml>
        <!-- end of deck                -->

```

Code output

Figure 3-5 shows the output of checkvalue.cfm.

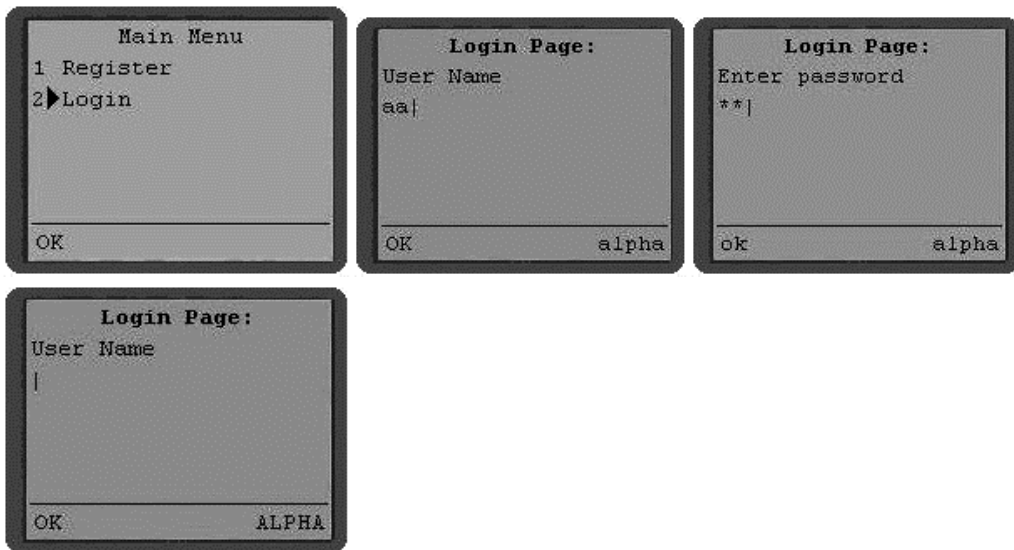


Figure 3-5: Output Screen of checkvalue.cfm

The readallvalue.cfm file (Listing 3-5) is used to read the total number of questions from the table. The code then generates a random number between one and the maximum number of questions. This number is used to display the question on the screen for the quiz. The control is then taken to the new file questiondisplay.cfm for displaying the questions on the mobile screen.

Listing 3-5: readallvalue.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. <CFCCONTENT TYPE="text/vnd.wap.wml">
```



```

2. <cfquery name="quest" datasource="prototype" dbtype="ODBC">
3. SELECT      qno AS tempqno
4. FROM        coursetable
5. </cfquery>
6. <CFSET tempnumber="1">
7. <CFSET rendernumber="#RandRange("#tempnumber#", "#quest.tempqno#")#">
8. <CFSET Session.render="#rendernumber#" >
9. <?xml version="1.0"?>
10. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
11. <wml>
12. <head>
13. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
14. </head>
15. <card newcontext="true">
16. <cflocation url="questiondisplay.cfm" addtoken="No">
17. <p>
18. </p>
19. </card>
20. </wml>

```

Code output

No output is generated from this screen. The whole code listing is doing the background processing.

The `questiondisplay.cfm` file (Listing 3-6) displays the question read from the table on-screen. The particular number of questions to be displayed is generated in the program `readallvalues.cfm`.

Five queries have been defined to read question, three options, and one correct answer code from the table. The correct answer choice is used to check the answer entered by the user and is stored in the session variable for the future use.

Listing 3-6: questiondisplay.cfm

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. <cfquery name="display" datasource="prototype" dbtype="ODBC">
   <!-- declared for indicating the type of page being -->
   <!-- sent to the browser. -->

2. SELECT      question AS tempquest
3. FROM        questiontable
4. WHERE       qid = #session.render#
5. </cfquery>
   <!-- query to find out the question from the questiontable -
->
   <!-- table for display on the screen. -->

6. <cfquery name="choiceid1" datasource="prototype" dbtype="ODBC">
7. SELECT      choice1 AS tempchoice1
8. FROM        answertable
9. WHERE       qid = #session.render#
10. </cfquery>
   <!-- query to find out the answer 1 from the questiontable -
->
   <!-- table for display on the screen. -->

```

```

11. <cfquery name="choiceid2" datasource="prototype" dbtype="ODBC">
12. SELECT      choice2 AS tempchoice2
13. FROM        answertable
14. WHERE       qid = #session.render#
15. </cfquery>
      <!-- query to find out the answer 2 from the questiontable -
->
      <!-- table for display on the screen. -->

16. <cfquery name="choiceid3" datasource="prototype" dbtype="ODBC">
17. SELECT      choice3 AS tempchoice3
18. FROM        answertable
19. WHERE       qid = #session.render#
20. </cfquery>
      <!-- query to find out the answer 3 from the questiontable -
->
      <!-- table for display on the screen. -->

21. <cfquery name="rightanswer" datasource="prototype" dbtype="ODBC">
22. SELECT      answerid AS correct
23. FROM        answertable
24. WHERE       qid = #session.render#
25. </cfquery>
      <!-- query to read the correct answer code from questiontable -->
      <!-- table for checking with user choice -->

26. <CFSET Session.rightanswer="#rightanswer.correct#">
      <!-- setting up session variable to rightanswer -->
      <!-- to check for user input -->

27. <CFCCONTENT TYPE="text/vnd.wap.wml">
28. <?xml version="1.0"?>
29. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
      "http://www.phone.com/dtd/wml11.dtd" >
      <!-- WML Prolog -->

30. <wml>
      <!-- WML deck declaration -->

31. <head>
32. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
      <!-- Meta tag declaration -->

33. </head>
34. <card id="questiodisplay" newcontext="true" >
      <!-- new card declaration to display question & answers -->

35. <do type="accept" label="Submit">
36.     <go href="submit.cfm" method="post" >         </go>
      <!-- action declared to navigate to new file -->

37. </do>
38.     <p align="center">
39.         Your Quotation
      <!-- new paragraph declaration to display heading -->

40.
41.     </p>
42.     <p>

```

```

        <!-- new paragraph declaration to display question -->
43.
44.                <CFOUTPUT>
45.                #display.tempquest#
46.                </CFOUTPUT>
47.                </p>
48.                <p>
        <!-- new paragraph declaration to display first choice -->
49.                <CFOUTPUT>
50.                <b>(a) </b>.. #choiceid1.tempchoice1#
51.                </CFOUTPUT>
52.                </p>
53.                <p>
        <!-- new paragraph declaration to display second choice -->
54.                <CFOUTPUT>
55.                <b>(b) </b>.. #choiceid2.tempchoice2#
56.                </CFOUTPUT>
57.                </p>
58.                <p>
        <!-- new paragraph declaration to display third choice -->
59.                <CFOUTPUT>
60.                <b>(c) </b>.. #choiceid3.tempchoice3#
61.                </CFOUTPUT>
62.                </p>
63. </card>
        <!-- end of card -->
64. </wml>
        <!-- end of deck -->

```

Code output

Figure 3-6 shows the output of questiondisplay.cfm.

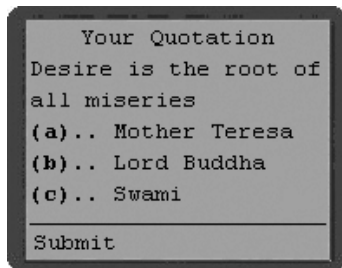


Figure 3-6: Output Screen of questiondisplay.cfm

The submit.cfm file (Listing 3-7) is used to accept the correct answer option from the user. The answer choice entered by the user is taken in the variable `yanswer`, and the navigation is set to `answer.cfm` with the value of the field answer taken as the parameter.

Listing 3-7: Submit.cfm

```

1. <CFCONTENT TYPE="text/vnd.wap.wml">
   <!-- content type declaration for browser -->

2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
   <!-- WML Prolog -->

4. <wml>
5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
   <!-- meta tag declaration -->

7. </head>
8. <card newcontext="true">
   <!-- declaration of card to take input from user -->

9. <do type="accept" label="ok">
   <!-- declaration of action event -->

10.     <go href="answer.cfm" method="post" >
       <!-- declaration of navigation to new file -->

11.         <postfield name="yanswer" value="$(yanswer)" />
       <!-- declaration of variable to be navigated to new file

12.         </go> -->
13.     </do>
       <!-- end of action -->

14. <p>
15. Your Answer<input name="yanswer"/>
       <!-- input taken from user in variable 'yanswer' -->

16. </p>
       <!-- end of paragraph -->

17. </card>
       <!-- end of card -->

18. </wml>
   <!-- end of deck -->

```

Code output

Figure 3-7 shows the output of submit.cfm.

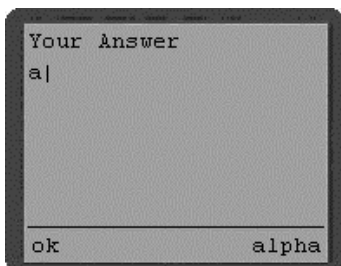


Figure 3-7: Output screen of submit.cfm

The `answer.cfm` file (Listing 3-8) verifies the answer that the user inputs for the displayed question. A new paragraph is declared to check for the right answer. If the user's input matches the right answer from the table, control is taken to a new file named `bingo.cfm`. Otherwise, control is taken to the another file named `tryagain.cfm`.

Listing 3-8: `answer.cfm`

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. <CFCONTENT TYPE="text/vnd.wap.wml">
   <!-- content type declaration for browser -->

2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
   <!-- WML Prolog -->

4. <wml>
   <!-- new deck declaration -->

5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
   <!-- Meta tag declaration -->

7. </head>
8. <card id="card1" >
   <!-- declaration of a new card -->

9. <p>
   <!-- declaration of a new paragraph to check userinput -->

10. <CFIF #session.rightanswer# EQ #yanswer# >
   <!-- Checking IF condition for correct answer -->

11. <cflocation url="bingo.cfm" addtoken="No">
   <!-- navigation declaration to new location -->

12. <CFELSEIF #session.rightanswer# NEQ #yanswer#>
13. <cflocation url="tryagain.cfm" addtoken="No">
   <!-- navigation declaration to new location -->

14. </CFIF>
   <!-- end of If condition -->

15. </p>
   <!-- end of paragraph -->

16. </card>
   <!-- end of card -->

17. </wml>
   <!-- end of deck -->
```

Code output

Because this program is checking the answer, there is no output. Depending upon the value of the condition, the navigation is sent to two different files.

The `bingo.cfm` file (Listing 3-9) displays the success message on the correct answer. The navigation is sent back to `index.cfm` after the display of the congratulation message.

Listing 3-9: bingo.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <CFCCONTENT TYPE="text/vnd.wap.wml">
   <!-- content type declaration for browser -->

2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
   // WML Prolog

4. <wml>
   <!-- declaration of new deck -->

5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
   <!-- meta tag declaration -->

7. </head>
8. <card>
   <!-- declaration of new card -->

9. <do type="accept" label="Home" >
   <!-- declaration of the event -->

10. <go href="index.cfm" />
    <!-- declaration of navigation link -->

11. </do>
12. <p align="center">
    <!-- declaration of new paragraph to display congratulation message
-->

13.                                     <b>Congratulations</b>
14. </p>
15. <p align="center">
    <!-- declaration of paragraph to display right answer choice -->

16.                                     <b> You choose right answer</b>
17. </p>
    <!-- end of paragraph -->

18. </card>
    <!-- end of card -->
19. </wml>
<!-- end of deck -->

```

Code output

Figure 3-8 shows the output of `bingo.cfm`.

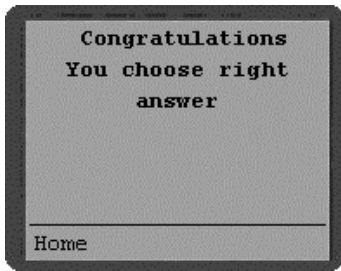


Figure 3-8: Output screen of bingo.cfm

The `tryagain.cfm` file (Listing 3-10) displays the “sorry” message when the user inputs an incorrect answer. The control is sent to `questiondisplay.cfm` after the wrong answer message appears.

Listing 3-10: tryagain.cfm

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <CFCCONTENT TYPE="text/vnd.wap.wml">
   <!-- content type declaration for browser -->

2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >
   <!-- WML Prolog -->

4. <wml>
   <!-- declaration of new deck -->

5. <head>
6. <meta http-equiv="Cache-Control" content="max-age=1" forua="true"/>
   <!-- meta tag declaration -->

7. </head>
8. <card>
   <!-- declaration of new card -->

9.
10.           <do type="accept" label="Back to question" >
   <!-- declaration of new event -->

11.           <go href="questiondisplay.cfm" />
   <!-- navigation declaration to new location -->

12.           </do>
13.
14. <p align="center">
   <!-- declaration of new paragraph to display wrong answer message
-->

15.           <b>Oops.. Wrong answer</b>
16. </p>
17. <p align="center">
   <!-- declaration of new paragraph to display try again message --
>

```

```

18.                <b>Try again</b>
19. </p>
    <!-- end of paragraph -->

20. </card>
    <!-- end of card -->

21. </wml>
    <!-- end of deck -->

```

Code output

Figure 3-9 shows the output of `tryagain.cfm`.



Figure 3-9: Output screen of `tryagain.cfm`

Complete Output

Figure 3-10 shows the completed output of the application.



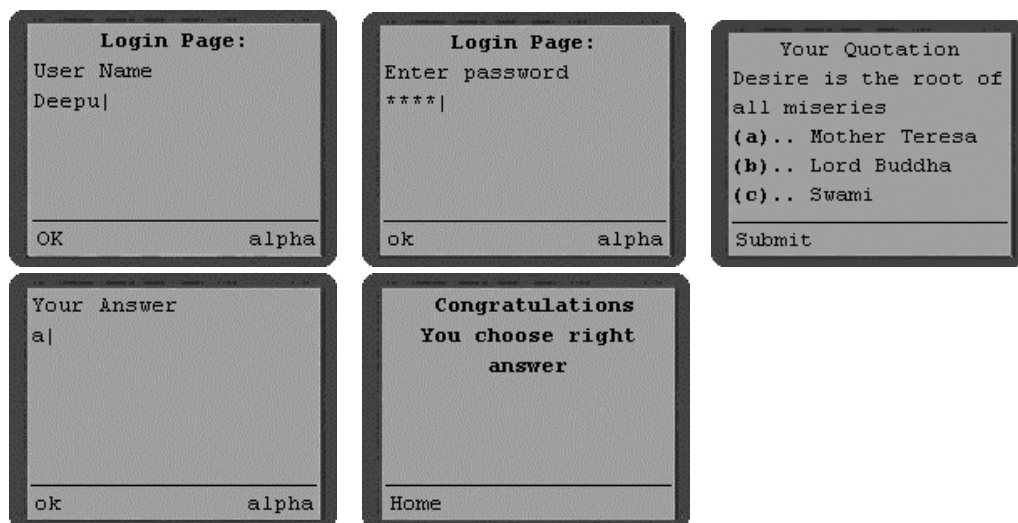


Figure 3-10: Application output

Summary

The Question Quiz application demonstrates how you can implement Cold Fusion on a cell phone. This application greatly helps the user understand the functioning, syntax, and usage of Cold Fusion in developing Web applications.

Chapter 4

WTA: An Advanced Interaction Technique for Mobile Phones

In the preceding chapter, we discussed the layered architecture of WAP. The topmost layer of this architecture is constituted by the WAE (Wireless Application Environment) and the WTA (Wireless Telephony Application). This layer is a telephony interface, which facilitates interaction between the markup language and the wireless client.

WTA is instrumental in equipping the conventional WAP architecture with additional telephony functions. It is primarily used to send the data to the mobile client, without receiving any request from the client. This concept provides the groundwork for what is called “push technology,” which you learn more about in later chapters.

WTA, which is implemented by push technology, is used to integrate voice and data so that both modes of communication can occur simultaneously on a single device. WTA bridges voice services and data services by providing a framework to access voice services using WML and WML Script. WTA, therefore, enables a wireless device to handle real-time events, even if it’s being used for browsing.

This chapter focuses on the WTA technology architecture and interface. We briefly discuss the event-management facet and provide details of real-time implementation of WTA.

Applications of WTA

The WTA applications provided by phone networks to work on mobile devices could interact with the telephony-related functions available on those same devices. Here we’re explaining how WTA functionality can be used for these applications. The list of hotels in a locality, for example, can be pushed to the mobile device as a WML card. The card contains the hotel names and the telephone numbers. When a particular hotel is selected, a voice call is made to the given telephone number — making the voice call is the WTA function call.

The main telephony-related functions available on a common wireless device are:

- ◆ Adding, searching, and updating the phonebook entries.
- ◆ Sending and receiving the short text messages.
- ◆ Making and receiving the call.
- ◆ Working on a keypad when a call is being received.
- ◆ Administering calls made and calls logged (missed and received calls).
- ◆ Writing and receiving the text messages on the wireless device.
- ◆ Sending DTMF tones while an active voice call is being made.

The functions provided by phone network are:

- ◆ Voicemail.

- ◆ Call transfer, hold, and forward.
- ◆ Conferencing.
- ◆ Intelligent Network Services.

Applications of the WTA in a common scenario are:

- ◆ Making a table reservation in a restaurant from a list of restaurants in the wireless device.
- ◆ Getting the status of the reservation done on the wireless device.
- ◆ Getting regular weather forecasts at the specified intervals.
- ◆ Score/regular political updates.
- ◆ Automatically calling a number found in a yellow pages search; visual interfaces to voice-mail systems.
- ◆ Automatically recalling a number until the call goes through.
- ◆ Sending an alert to the user, such as upon the arrival of a message in the user's voice mailbox when the user is using the mobile device for some other application (such as browsing).

Not only does the wireless device handle events, but also other external devices.

Introduction to WTA Architecture

In push technology, the WTA enables trusted application servers to initiate transmission of information to WAP devices. The main beneficiaries of this technology are online packet networks such as GPRS. Although network carriers primarily control the functionality of the WTA, ordinary users can also program the WTA by using the WTA libraries and functions.

The telephony services in WTA communication are managed by the WAP gateway, which delivers the WTA applications. In fact, WTA extends the WAP application environment by providing these services. For supporting the real-time needs of telephony applications, the service provider must have the necessary infrastructure to support WTA applications. The infrastructure required is the WTA server as indicated in Figure 4-1.

Figure 4-1 depicts the logical architecture of WTA.

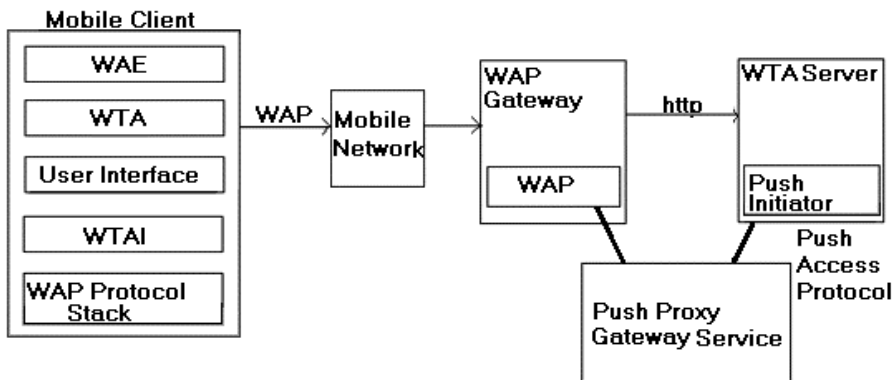


Figure 4-1: WTA architecture

The major components of the WTA architecture are:

- ◆ WTA user agent
- ◆ WTA server
- ◆ WTA interface libraries

- ◆ Storage (also called *Repository*)

We discuss each of these in the following sections.

WTA User Agent

The WAE in the WAP is composed of decks and cards. The user agent is similar to that in the WTA environment; it executes and presents the WML and the WML Script content to a server. The WTA user agent has strong real-time context management components. It doesn't have the stack through which decks are stored in the history and retrieved by using the Back button. The main functions required of the user agent are supporting WTAI libraries, rendering WML, and executing WML Scripts. The WTA user agent architecture is shown in Figure 4-2.

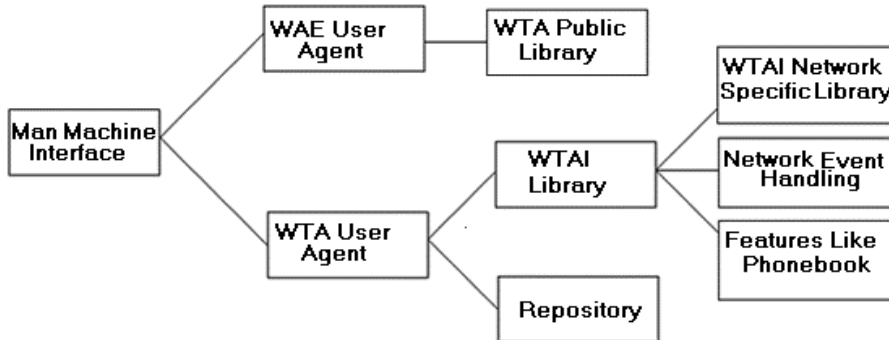


Figure 4-2: WTA user agent

WTA Server

The WTA server is basically a Web server that contains the WTA content and services. It's also responsible for the security of the network — the services are always under the direct or indirect control of the network operators who have the administrative rights to these services; these are called trusted services. The operators can transfer these rights to the third-party WTA providers that have to work according to the specifications of the network operators. Therefore, a content provider cannot provide the complete set of WTA services.

WTA Security Considerations

Security is an important consideration for a telephony architecture like WTA. You have to ensure the security of the original content of the WTA server. Only authorized persons should be able to write or amend the content of the WTA and execute it on a client device. Otherwise, you run the risk of a third party changing the content, such as address-book data, and making illegal or unauthorized calls through a WML and WML Script code.

To avert this situation, the following features are incorporated in the WTA architecture:

- ◆ With the help of a private network, a secure link is established between the WAP gateway and the WTA server.
- ◆ Strong authentication mechanisms have been defined for establishing a secure link between the WAP gateway and the WTA server.
- ◆ User-permission criteria are specified and configured over the mobile device for the execution of different WTAI library functions.
- ◆ A secure WTP port is used on the WAP gateway for the WSP sessions by the WTA user agent.

You can configure three types of user permissions on a mobile device. They are the following:

- ◆ **Blanket permission:** Granted to the executable for the specified WTAI library function.
- ◆ **Context permission:** Granted to the executable for the specified WTAI library function during a specific run-time context.
- ◆ **Single action permission:** Granted to the executable for the specified WTAI library function; it's a single permission.

A special security model is employed for maintaining the security of the WTA architecture. In this model, the WTA user agent sends the request for the WTA application from a mobile device to the WAP gateway through the WTP code by establishing a special WSP session. These sessions are authenticated and encrypted by using WTLS. The WTP ports (port 2805 for connectionless session and port 2923 for connection-oriented sessions) are used by WAP gateways to send the information to trusted WTA servers. The other WSP ports are not trusted, and information passed through those ports is confined to certain specified WTAI function libraries on the mobile client only after user authorization. If an application or content is not received through the trusted port, it is not admitted for the process network event. The WTA security model is shown in Figure 4-3.

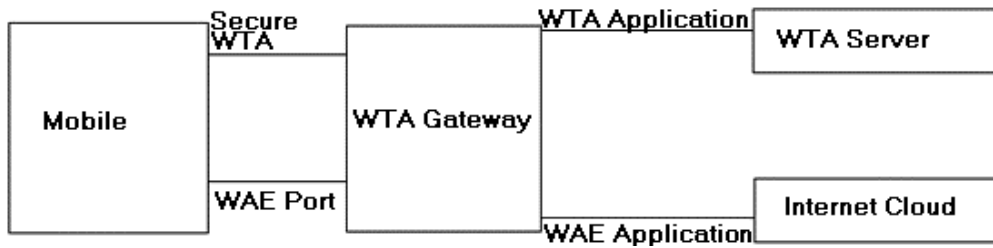


Figure 4-3: WTA security model

WTA Interface Libraries

The WTA libraries help in defining the WTA interface. There are three types of libraries:

- ◆ Network common WTAI—handles the incoming calls; independent of network.
- ◆ Network specific WTAI—depends on network; different for different networks.
- ◆ Public WTAI—for the user to adjust the properties of the device by using WAP architecture.

These libraries can be called in two ways:

- ◆ **URI Schemes:** It begins with `wtai://`. The syntax of the URI schemes is:

```
wtai://<library>/<function>;<parameter>![<result>]; [<result>]
<library> - name of the library. Voice call is vc
<function> - identifier of the specific function in the library
<parameter> - the parameter of the function
<result> - zero or more return values delimited by ;
```

An example is:

```
wtai://vc/rc;0;retval
```

- ◆ **WML Script:** In this method, the type of function is specified first, followed by the function name. The syntax is:

```
Library type.function name ([parameters]);
```

An example is:

```
WTAvoiceCall.release("0");
```

The functions of these libraries are discussed later in the chapter.

Storage or Repository

The common WTA services are stored within the mobile device as a separate storage module. This prevents having to access the network for loading and executing these services. Thus, the time taken to pull the content over the WTA network before executing it is saved, making for instant responses as with wireless communication.

Stored contents fall under the following two categories:

- ◆ **Resources:** This data resource includes decks, WML Script, and wireless bitmap images. It is downloaded to the mobile devices using WSP and is stored with meta-data. Multiple channels are used to share resources to conserve memory of the mobile client.
- ◆ **Channels:** This refers to special resources that contain a set of links to other resources and special control information. A framework of storage is provided on repository. This is programmed using Channel Content Format.

To load the resources and channels into the repository, the following methods are used:

- ◆ A channel is pushed directly into the repository from the WTA server.
- ◆ The application is installed into the repository at the time of manufacturing or during the subscription process.
- ◆ The WTA user agent can install the application by pulling it from the WTA server.

The channel is unloaded if it becomes stale or if space is required for any other application. At the time of unloading the channel, only those resources that are not shared by any other channel are removed from the repository. The WTA repository model is shown in Figure 4-4.

The various models of WTA and their areas of application are as follows:

- ◆ **Voice call model:** This model helps in placing, receiving, and terminating voice calls. With this model, implementation of one or many simultaneous voice calls is possible.
- ◆ **Network message model:** This model helps in sending, receiving, and getting the information regarding network messages.
- ◆ **Phone-book model:** Used to access and modify the phone-book available on the mobile devices.

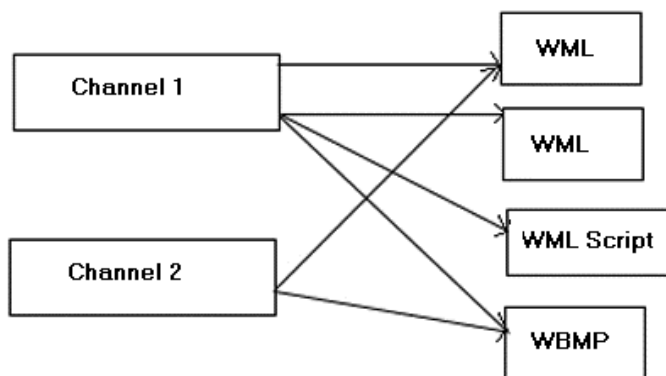


Figure 4-4: WTA repository

- ◆ **Call log model:** Used to maintain the logs of the calls, which include information about missed, dialed, and received calls.
- ◆ **Logical indicator model:** To access the logical indicators of the devices, which are:
 - Fax, voice, or e-mail messages

- Incoming voice, fax, or data calls
- Call waiting
- Voice call messages

WTA services can be initiated by selecting the URL, whose content is hosted on the WTA server, by using push technology over a secure WTA session or by using an event handler.

Using the Interface Components

The WTA application-programming interface has two components:

- ♦ **The event model:** Allows the application to take action depending on the events.
- ♦ **A scripting interface:** Allows the application to initiate and control telephony operations.

The Event Model

This model generates a sequence of events, depending on the calls received or calls made by the user. These events are accepted by the WTA user agents and transferred to the WTA application, which employs the appropriate event handlers to handle these events. The information regarding an event, such as details of the transaction ID and the party called, are stored in the variables in WML. Because the values of the variables change while navigating through the cards, these are stored in card-specific variables. There are two models, which are used to trap telephony events from the voice network:

- ♦ **Originating Call model:** This model handles the events generated when a call is made from the device. The processing occurs as follows: When the user makes a call, the WTA application receives the outgoing call (wtac-cc/cl) event, and the call transitions to the Digits Dialed state. The mobile client detects ringing from the network in this state only. The Call Connecting (wtac-cc/cl) event is received by the WTA user agent, and the call proceeds to the Ringing state. When the call is answered, the Call Connected (wtaev-cc/co) event is received by the WTAI user agent, and the call enters the Answer state. This state is maintained until the user or the called party disconnects, thereby triggering the Call Cleared (wtac-cc/cl) event. The phone returns to the Idle state when the call has been answered. This model is outlined in Figure 4-5.

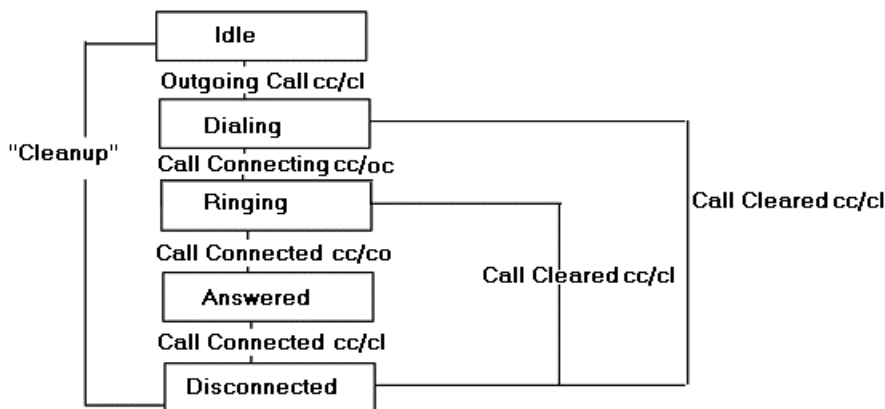


Figure 4-5: Originating Call model

- ♦ **Terminating Call model:** This model handles the events generated when the device receives a call. The processing occurs as follows: An Incoming Call (wtaev-cc/ic) event causes the call to move from the Idle state to the Ringing state. The Call Connected (wtaev-cc/co) event is received by the WTA user agent when the user receives the call and the call moves from the Ringing state to the Answer state. This state is maintained until either the user or the calling party disconnects. Then the

WTA user agent receives the Call Cleared (wtaev-cc/cl) event. The phone gets back to the Idle state after the call is cleared. It's then ready for new incoming or outgoing calls. This model is outlined in Figure 4-6.

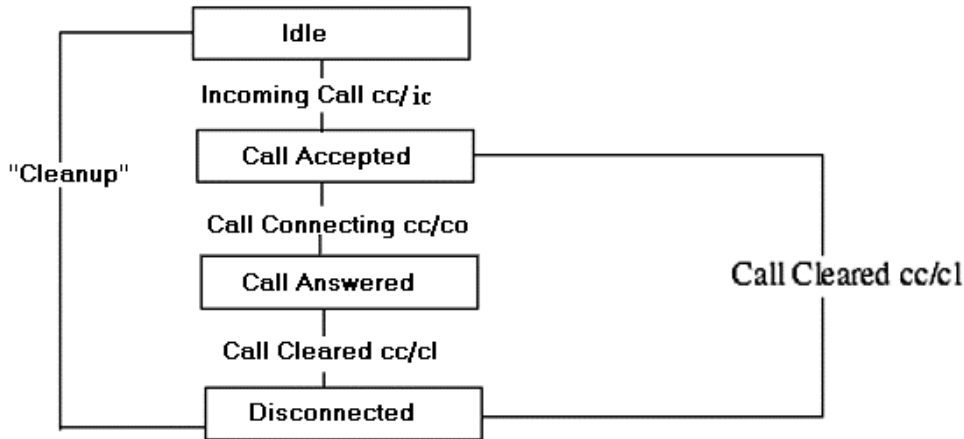


Figure 4-6: Terminating Call model

So far, we have discussed some predefined WTAI network events. Table 4-1 contains the complete list of WTAI events and their meanings.

Table 4-1: WTAI Events

<i>Event name</i>	<i>Event id</i>	<i>Description</i>
cc/ic	Wtaev-cc/ic	Call control/incoming call indicator event that is picked up by the calling WTAI function, accept call.
cc/cl	Wtaev-cc/cl	Call control/Call cleared event indicates that the connected call is disconnected.
cc/co	Wtaev-cc/co	Call control/Call connected event indicates that the call is connected.
cc/oc	Wtaev-cc/oc	Call control/Outgoing call event that indicates that outgoing call is being placed.
cc/cc	Wtaev-cc/cc	Call control/Outgoing call alert event that indicates bell ringing at the destination.
cc/dtmf	Wtaev-cc/dtmf	Call control/dtmf sequence sent event.
nt/it	Wtaev-nt/it	Network text/Incoming text event that indicates the receipt of network text message.
nt/st	Wtaev-nt/st	Network text /Network text sent event that indicates sending of the network text message.

ms/ns	Wteav-ms/ns	Miscellaneous/network status indicator event generated on the change in certain network parameter.

Event Binding

The events, as explained in Table 4-1, are generally used in the WML Script file. The channel and resources use the events as follows:

```
<?xml version="1.0"?>
<!DOCTYPE channel PUBLIC " -//WAPFORUM//DTD CHANNEL 1.2//EN"
"http://www.wapforum.org/DTD/channel.dtd">
<channel
maxspace="2048"
base ="http://wta.operator.com"
eventid="wtae-cc/ic"
channelid="Incoming Call Distributor"
success="wtaSuccess.wml"
failure="wtaFailure.wml"
>
<resource href="first.wml" />
<resource href="first.wmls" />
</channel>
```

WTAI Scripting Interface and Function Libraries

The WTAI scripting interface is made possible through the various WTAI function libraries. The function type dictates the library specification. These libraries can be accessed by using the WML Script, through the scripting function libraries, or through URIs. These steps may initiate an interaction between the network and the mobile device. The function then terminates independently from the started network procedure. Any result delivered by this function call will not reflect the outcome of this procedure, which itself may result in events.

There are three main function libraries.

Network common WTAI

The Network common WTAI is used to control the handling of incoming calls, independent of the network. The related functions are shown in Table 4-2.

Table 4-2: Network Common WTAI Functions

<i>Function</i>	<i>Explanation</i>
WTAVoiceCall.setup	Initiates a mobile-generated call. This is a non-blocking function.
WTAVoiceCall.accept	Accepts an incoming voice call, waiting to be attended.
WTAVoiceCall.release	Releases a voice call.
WTAVoiceCall.sendDTMF	Sends a DTMF sequence through a voice call.
WTAVoiceCall.callStatus	Retrieves information about a voice call.
WTAVoiceCall.list	Returns the call handle that can be controlled within the WTA context that involved the function.
WTANetText.send	Sends a network message.

<code>WTANetText.list</code>	Returns the message handle of the existing message.
<code>WTANetText.remove</code>	Permanently removes incoming or outgoing network message from the device.
<code>WTANetText.getListValue</code>	Retrieves a field value from a specific value.
<code>WTANetText.markAsRead</code>	Marks a message as read.
<code>WTACallLog.dialed</code>	Returns the call log handler of an entry from the dialed call log.
<code>WTACallLog.missed</code>	Returns the call log handler of an entry from the missed call log.
<code>WTACallLog.received</code>	Returns the call log handler of an entry from the received call log.
<code>WTACallLog.getFieldValue</code>	Retrieves a field value from the specified log entry.
<code>WTAPhoneBook.write</code>	Writes a phone-book entry, overwriting the existing one.
<code>WTAPhoneBook.search</code>	Returns the index of phone book entry that matches the search condition.
<code>WTAPhoneBook.remove</code>	Removes an entry from the phone-book making the entry vacant.
<code>WTAPhoneBook.getFieldValue</code>	Retrieves a field value for a specific phone entry.
<code>WTAPhoneBook.change</code>	Stores the given value in the specified field in the phone-book entry.
<code>WTAMisc.setIndicator</code>	Modifies the status of the logical indicator.
<code>WTAMisc.endContext</code>	Terminates the current WTA user agent context.
<code>WTAMisc.setProtection</code>	Sets the protection mode of the WTA context.
<code>WTAMisc.getProtection</code>	Returns the protection mode of the WTA context.

Network specific WTAI

Network-specific WTAI functions and events are applicable to only one specific network and will differ from network to network. To date it has only been developed for GSM, IS136, and PDC (JAPAN) based networks. The network-specific libraries also define additional events in order to support extra capability by the new network.

For example:

- ◆ `location()` — One important GSM specific function used for providing location information (mobile country code, network code, area code, and the cell identifier). This information can be further used to provide area specific information to the subscriber.
- ◆ `sendUSSD()` — The additional function of GSM library to handle USSD (Unstructured Supplementary Service Data).
- ◆ `reject()` — Used for rejecting the incoming call.
- ◆ `hold()` — For call held.
- ◆ `transfer()` — For call transfer.

- ◆ The WTAGSM library provides the functions for multiparty operations, (`multiparty()`, `retrieve()`, and so on) through which a subscriber can be added to an ongoing call or be disconnected from the multiparty call.

Public WTAI

Public WTAI is used to adjust the properties of the device by using the WAP architecture. A public library has the following three functions:

- ◆ Make a call.
- ◆ Send DTMF tones.
- ◆ Add a phone book-entry.

The functions are outlined in Table 4-3.

Table 4-3: Public WTAI Functions

<i>Function</i>	<i>Explanation</i>
<code>WTAIPublic.makeCall</code>	To initiate a mobile-generated voice call.
<code>WTAIPublic.sendDTMF</code>	Send a DMTF sequence through the voice call most recently created using the <code>WTAIPublic.makeCall</code> function.
<code>WTAIPublic.addPBLibrary</code>	Used to write a new phone-book entry.

Functions used in the script file

The following sections describe some functions commonly used in the WML Script file.

This is the most basic function used for making a call. To make a phone call programmatically, rather than manually, use the following code:

```
Var a= WTAIPublic.makeCall("1234567")
```

This makes a phone call from the phone to the number specified as the argument. The return code from this function can be used for application development. The return codes are as follows:

- ◆ Empty string — successful code
- ◆ 105 — busy party
- ◆ 106 — network not available
- ◆ 107 — no answer from called party
- ◆ 1 — an unspecified error

The `WTAIPublic.sendDTMF` (dual-tone multifrequency) is better known as touch-tone dialing. This function gives the added benefit of adding a # and * in the dialing string. For example:

```
WTAIPublic.sendDTMF("123*4567")
```

The preceding code would send the touch-tone sequence associated with that string, just as if the user had entered it on the keypad manually. The error return values are:

- ◆ empty string — all occurred as planned
- ◆ 108 — no active voice connection
- ◆ 1 — an unspecified error

Using the DTMF function, you can automate to glean an account number from a database (granted, you want to ensure some level of security here) and store it temporarily in the phone's memory. The phone subsequently dials the bank and transfers the information directly.

By virtue of this function, one can use the phone-book facility of mobile phones. Through this facility, cell phones can store numbers. The function that adds entries to the phone-book is `WTAPublic.addPBEntry`, which enables a phone to read the bank's automated system, dial out, and enter the information by pressing a single button (or key on the keypad). This provides almost instantaneous access to the bank account.

This command has two parameters, which are the number and the associated name. The return codes are:

- ◆ Empty string — operation completed successfully
- ◆ 100 — *name* parameter is too long or unacceptable
- ◆ 101 — *number* parameter is not a valid phone number
- ◆ 102 — *number* parameter is too long
- ◆ 103 — the phone-book entry could not be written
- ◆ 104 — the phone book is full
- ◆ 1 — an unspecified error

Suppose that a business provides customer service over the phone. The company can add a link in its WAP page that automatically stores its toll-free number in the user's phone to make sure it's always readily available in future. For the action of a key, only the following line of code is added:

```
WTAPublic.addPBEntry("18001234567", "WTAI")
```

The functions we just discussed are generally used in the WML Script file. The user-defined function, in which the WTA library function is used, is called by the WML application file, which is shown in Listing 4-1.

Listing 4-1: appl.wmls

```
function kc(no)
{
var ide;
while ( (ide = WTAVoiceCall.setup(no, 0)) <0)
{
if (ide > -4 && ide < -7 ) break;
}
return ide;
}
```

The function is called from the WML application file (`kca.wml`) as:

```
<do type="accept">
<go href="appl.wmls#kc($no)" />
</do>
..
```

Event and State Management in WTA

Through WTA programming, you can perform various actions on WTA events. All the network events are processed by the WTA user agent, which, in turn, directs the network event to the appropriate handlers within the applications.

There are two types of bindings by which events are bound to event handlers. They are:

- ◆ **Global binding:** When the event handler for a particular event is present in the repository, they are bound globally. These are called every time the event occurs. Unless a temporary binding is not defined with the current deck and the context is not protected, a matching global binding also exists.
- ◆ **Temporary binding:** This type of binding occurs when the event handler is present in the content currently being executed by the WTA user agent. This is executing when the WTA service is running in the WTA user's agent. In this case, the temporary binding also exists.

If there is no binding, the event is handled in the following way:

- ◆ **Callback handling:** The event is not allowed to interrupt the currently executing WTA context, so there isn't any binding.

When a network event is received, it's the function of the user agent to check for the bindings. The user agent tries to locate local binding by dispatching the event to the locally executing program. If there's a matching event handler, the temporary binding exists. Otherwise, the user agent searches for the event handler in the repository. If the event handler is not found in the repository, the event is passed to the phone to be handled by a process of callback handling done in the default fashion.

It's possible to control the WTA event to interrupt the execution of content in a WTA context by using the library function:

```
WTAMisc.protected(1)
```

Table 4-4 shows the different actions taken by the WTA user agent, depending on the state of the WTA user agent, when the WTA context is interrupted.

Table 4-4: WTA User Agent Actions

<i>State</i>	<i>Action</i>
Stable	Current WTA context is interrupted to process WTA event.
WAI function is executing	Wait for completion of the function and then the WTA context is interrupted to process the event.
WML Script is processed	WML Script is interrupted, the current context is interrupted, and the event is executed.
WSP method request in progress	Abort all request of WSP and event is executed.
Local Navigation in progress	Navigation is completed, current context is interrupted, and the WTA is processed.

WTAI Function Call Example

In this example, we illustrate how you can access some of the emergency services. The mobile user obtains a list of services (in text form) on the display. When the desired service is selected through the keypad, a call is made to the corresponding telephone number. Note that in absence of WTA, the WML cards provide only text services. Here, we make use of a WTAI function call to dial the telephone number. Listing 4-2 provides the WML code to achieve this functionality. Note that the code can be tested on devices that support only WTAI functionality.

Listing 4-2: WML Code for Accessing Emergency Services through WTAI

© 2001 Dreamtech Software India Inc.
All Rights Reserved.

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
3. <wml>
4. <card id="service" title="EMERGENCY SERVICES ">
5. <do type="accept" label="EServices">
6. <go href="#ers"/>
7. </do>
8. <p>
9. <br/>
10. <br/>
11. <b>Check Services</b>
12. </p>
13. </card>
14. <card id="ers" title="eServices">
15. <do type="accept" label="GetIt">
16. <go href="wtai://wp/mc;$emergencies"/>
17. </do>
18. <p>
19. Choose Service :
20. <select name="ServiceNum">
21. <option value="5556789">Pharmacy</option>
22. <option value="5551234">Car Service</option>
23. <option value="5553344">Hospital</option>
24. </select>
25. </p>
26. </card>
27. </wml>

```

Code Description

- ◆ Lines 1–2: XML version and Document Type Definition.
- ◆ Line 3: Beginning of WML deck.
- ◆ Line 4: Beginning of the first card with ID as service and Title as emergency services.
- ◆ Lines 5–7: <do> tag with type as accept and title as eServices. When the user presses the accept button, the control is transferred to the card which has the ID as 'ers'.
- ◆ Lines 8–12: Display the text Check Services.
- ◆ Line 13: End tag for the first card.
- ◆ Line 14: Beginning of second card. This card has two attributes: ID and title. The card ID is ers and the title is eServices.
- ◆ Lines 15–17: <do> tag with the task of calling the telephone number that is obtained based on the option selected. The selected telephone number is stored in the variable emergencies. In line 16, we made use of the WTAI function to make a call.
- ◆ Lines 18–25: This code displays the message Choose Service and provides the options of Pharmacy, Car Service, and Hospital. Corresponding to each service, a telephone number is allocated in the option tag with attribute value. When a particular option is selected by the user, the corresponding value will be stored in the variable emergencies and WTA will dial that telephone number.

- ◆ Line 26: End tag for second card.
- ◆ Line 27: End tag for WML.

Code Output

The output on the phone emulator is shown in Figure 4-7.

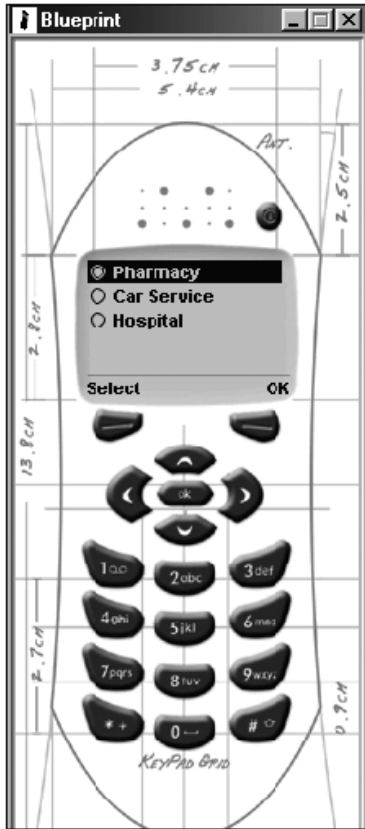


Figure 4-7: Service options displayed on the WAP phone emulator

Summary

This chapter focused on the WTAI as a part of the WAE interface, which is the topmost layer of the WAP architecture. This interface, making use of the push technology, provides standard telephony services apart from Web browsing and searching. We discussed the architecture and implementation of the WTA, its common uses, such as manipulation of phone-book entries, and sending and receiving short text messages. The WTA also maintains the call log, which includes the calls received, missed calls, and calls made. The event management and the state management of the device were also discussed. We also described building the WTA application environment on the standard WAP application by using the WML and the WML Script. The WTA application's ability to give the user interaction control over the incoming and outgoing calls, and its ability to turn the complex calling operations into easy tasks, has made it both popular and necessary.

Chapter 5

Integrating Java with WAP

Storing and accessing the data from the database server, as well as placing the data on the client, are vital aspects of a Web application. Technologies such as Cold Fusion, ASP, PHP, and Java servlets can be useful in server-side processing. We have already considered the suitability of Cold Fusion technology for server-side scripting. Java offers itself for partial client-side and server-side development. This chapter is devoted to the features of Java that make it suitable for WAP.

Because this chapter focuses on explaining the application of JSP and servlets for the wireless client, we will be working with and explaining the application. In order to understand the syntax and commands used in the application, a quick review of the popular Java technologies — with special emphasis on JSPs and servlets — is necessary.

Introduction to Java Technologies

Programmers at Sun Microsystems founded Java in 1991. It was developed as a platform-independent, object-oriented language, which could be used to create software for consumer electronic devices. By 1993, programmers were using it frequently in Internet programming, and it became famous as the “Internet version of C++.”

Since then, Sun Microsystems has developed several editions of Java and related software. Table 5-1 lists some of the main technology groups and products.

Table 5-1: Java Technology Groups

Group	Product	Description
Java 2 Platform, Standard Edition (J2SE)	Java 2 SDK, Standard Edition, v 1.3 (SDK) Java 2 Runtime Environment, Standard Edition, v 1.3 (JRE) Java Plug-in Java Web Start Java HotSpot Server Virtual Machine	Used by developers for writing applications and applets.
Java 2 Platform, Enterprise Edition (J2EE)	ECperf Version 1.0 Java Servlet 2.3 Java Server Pages 1.2 Java Pet Store JDBC API 3.0	Used for building server-side applications as it uses various technologies.
Java 2 Platform, Micro Edition (J2ME)		Used for handling micro devices such as pagers, PDAs, consumer

		electronics, and embedded devices.
Consumer and Embedded Technologies	Java 2 Platform, Micro Edition Connected Device Configuration (CDC) Connected Limited Device Configuration (CLDC) C Virtual Machine (CVM) K Virtual Machine (KVM) Mobile Information Device Profile (MIDP) Java 2 Platform, Micro Edition, Wireless Toolkit Personal Java Application Environment Personal Java Technology, Source Edition Embedded Java Application Environment Embedded Java Technology, Source Edition Java Card JavaPhone API Java TV API	For small devices, having short resources.
Consumer and Embedded Products	Java Dynamic Management Kit Java Embedded Server Software	Used for wireless, home gateway, digital interactive TV, and automotive.
Optional Packages	Java Media Framework (JMF) Java Communications API JavaBeans Activation Framework (JAF) Java Naming and Directory Interface (JNDI) JavaMail InfoBus Java 3D Java Advanced Imaging Java Servlet Java Cryptography (JCE) JavaHelp RMI-IIOP	Extensions to core Java products.

	Java Authentication and Authorization Service Java Secure Socket Extension	
Early Access Optional Packages	Java Management	Used for designing system management solutions for heterogeneous systems.

Java Servlets

Servlets are used to make dynamic pages by getting data from the database server and giving it to the client. These are modules of Java that run on the Java server. The packages required to work with Java are `javax.servlet` and `javax.servlet.http`. In order to run Java servlets, you need either a Web server that understands Java servlets or a self-standing Java servlets engine. A Java servlet must undergo three stages of development:

- ◆ **Initialization:** Done by `init()`, which is called just once in the life span of a servlet. This method is finished before any other method is called in a program.
- ◆ **Service:** Receives the request and sends the response to the user. Done through the `service()` method. `ServletRequest` and `ServletResponse` objects are used to manipulate the user requests and responses. Other objects in the service method are `doGet`, `doPost`, `doPut`, and `doDelete`, which are used for handling Get, Post, Put, and Delete requests of the user, respectively.
- ◆ **Destruction:** By using the `destroy()` method, a servlet can be destroyed from the memory. This is also done just once in the life cycle of a servlet.

A servlet can generate the output in a pure-text format or even generate the client application code. The HTML or WML codes can be generated from a servlet to display dynamic content on the client screen.

An example of creating a servlet follows:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
    public void service (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter out= response.getWriter();
        out.println("Hello World");
    }
}
```

Save the file as `HelloWorld.java` and then compile the code by entering:

```
c:\>javac HelloWorld.java
```

The class file named `HelloWorld.class` is created. Next, copy the class file in `TOMCAT-HOME/examples/web-inf/classes`. Run it by typing the following command in the browser:

```
http://localhost/examples/servlet/HelloWorld
```

Figure 5-1 shows the output of the servlet.



Figure 5-1: Execution of servlet

The Java Server Page

A Java Server Page (JSP) is the implementation of Java code into HTML or WML so that the page becomes dynamic. The JSP talks to Java classes, servlets, applets, and the Web server. A JSP is compiled to a servlet by a JSP engine, and the servlets execute to get the response for the user. The response is inserted in the HTML code, thereby generating a dynamic Web page for the client. JSPs and servlets are related in a complementary way to make data communication possible over the Internet. The JSP technology is an extension of the servlets.

A JSP is a simple text file having HTML, WML, or XML code; JSP tags are integrated between the code. JSP codes are short forms of Java codes. The following is an example:

```
<%= "hello"%>
```

Here's another example:

```
<%
out.println("hello");
%>
```

Now create the following JSP code:

```
<%= "Hello World"%>
```

Save this file as `Hello.jsp` in `Tomcat_HOME\examples\jsp`. To execute the above JSP, enter the following URL in the browser.

```
http://localhost:8080/examples/jsp/Hello.jsp
```

Figure 5-2 shows the output of `Hello.jsp`.

Create Dynamic Content with Servlets and JSPs for WAP Browsers

You can conveniently use Java servlets and JSPs for developing dynamic WML documents for mobile devices. The prerequisite is that the user must be familiar with JSPs, servlets, and WML (wireless markup language). To run the WML document, the MIME type has to be set. You can do this with a servlet in the following way:

```
Response.setContentType("text/vnd.wap.wml");
```



Figure 5-2: Output of Hello.jsp Code

Listing 5-1 demonstrates the development of WML from a servlet.

Listing 5-1: TrialServlet.java

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

    //importing various packages used by the servlets
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
    //class declaration of the name TrialServlet
4.  public class TrialServlet extends HttpServlet {
5.  public void service (HttpServletRequest req, HttpServletResponse
    res) throws ServletException, IOException {
    Setting the mime as wml document
6.  response.setContentType("text/vnd.wap.wml");
    //Response.getWriter gives a stream to write the response to the client
7.  PrintWriter out = response.getWriter();
    // Writing to the stream
8.  out.println("<?xml version=\"1.0\"?>");
9.  out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1/EN\"");
10. out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    // WML Prolog
11. out.println("<wml>");
12. out.println("<card title=\"MobileDemo\">");
    //declaration of new card
13. out.println(" <p align=\"center\">");
14. out.println(" A trial servlet <br/>");
15. out.println(" Prints the welcome message");
16. out.println("</p>");
    //end of paragraph
17. out.println("</card>");
    //end of card
18. out.println("</wml>");
    //Response send to the wap browser or client. The content type is the WML
19. }
20. }

```

Code Description

- ◆ Lines 1–3: Importing various packages used for the servlets to run. These packages include packages for the Input/Output operations, and servlet handling operation.

- ◆ Lines 4–5: A class is declared. This class is inherited from the base class `HttpServlet` and a method — `doGet` — is called, which is responsible for sending the data back to the calling client program.
- ◆ Line 6: Setting the response type of the response to be sent to the client as `text/vnd.wap.wml`.
- ◆ Line 7: Obtaining an object of the class `PrintWriter`. This is used to send the response to the client program.
- ◆ Lines 8–18: These lines throws WML, which is the response sent to the WAP browser on the client device.

Listing 5-2 also demonstrates the development of WML, but using JSP.

Listing 5-2: TrialJsp.JSP

© 2001 Dreamtech Software India Inc
All Rights Reserved

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
3. <%
   //Setting up mime Type as wml document
4. response.setContentType("text/vnd.wap.wml");
5. out.println("<wml>");
6. out.println("<card title=\"MobileDemo\">");
7. out.println("  <p align=\"center\">");
8. out.println("The demo JSP Page <br/>");
9. out.println("Prints the welcome message");
10. out.println("</p>");
11. out.println("</card>");
12. out.println("</wml>");
   //above lines starting from out.println is the response send to the wap
   browser or //client. The content type is the WML
13. %>

```

Both Listings 5-1 and 5-2 print the welcome message on the mobile device.

A JSP and Servlets-Based Application for WAP

Now that we have discussed JSP and servlets in brief, let's look at an application that demonstrates the implementation of the same technologies to create WAP-based pages, and to store and retrieve the data in the MS Access database.

This application generates a WAP page with a menu that contains two options: Weather Report and Question of the Day. On selecting Weather Report, a list of cities appears. From this list, the user selects a city. The weather details of that city then appear on the screen. On selecting the Question of the Day option, the user has the chance to answer a multiple-choice question from a given list of answers. A greeting will appear if the answer selected is correct; otherwise, the user will receive the message “wrong answer.”

Application Structure

Five files are used in this application:

- ◆ `TestWML.java`
- ◆ `Call.java`
- ◆ `Report.java`

- ◆ Solution.java
- ◆ Question&report.mdb

Of the five files, four are Java codes and one (Question&report.mdb) is the database in MS Access in which questions are stored.

Application Work Flow

Figure 5-3 outlines the function of the Weather application.

Description of the Application

This section examines the Java files in detail.

TestWML.Java (Listing 5-3) creates a menu on a wireless device. The menu has two options: Weather Report and Question of the Day. From this file, Call.java is called, in which the user's choice is passed as a parameter.

Listing 5-3: TestWML.Java

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1.import java.io.*;
2.import javax.servlet.http.*;
   //importing java standard packages used by Servlets
3.public class TestWML extends HttpServlet
   // declaration of the main servlet class which extends httpServlet class
4.{
5.    PrintWriter out;
   // A method by the name doGet is called which is responsible for sending the
   // response back to the client program
6.    public void doGet(HttpServletRequest req, HttpServletResponse res)
   // declaration of the service method
7.    {
8.        try
   //Begin of the block for exception checking
9.        {
10.            res.setContentType("text/vnd.wap.wml");
   //Setting MIME type as WAP Document
   //Obtaining the stream to the program
11.            out = res.getWriter();
   // Writing the various parameters on to the stream

12.            out.println("<?xml version=\"1.0\"?>");
13.            out.println("<!DOCTYPE wml PUBLIC\"\" + \"-//WAPFORUM//DTD WML 1.1//EN\" \" +
   \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
   //WML Prolog declaration in the servlet
14.            out.println("<wml>");
15.            out.println("<head>");
16.            out.println("<meta http-equiv=\"Cache-Control\" content=\"max-age=time\"
   forua=\"true\"/>");
   // meta tag declaration
17.            out.println("</head>");
18.            out.println("<card id=\"MyFirst\" newcontext=\"true\">");
   // Event handling tags are written on to the stream
19.            out.println("<onevent type=\"onenterforward\">");
   // declaration of the event

```

```

20. out.println("<refresh>");
    // refreshing the memory
21. out.println("<setvar name=\"choice\" value=\"1\"/>");
    // declaration of the variable to set initial value to 1
22. out.println("</refresh>");
    // end of refresh
23. out.println("</onevent>");
    // end of event
24. out.println("<do type=\"accept\" label=\"Ok\">");
    // declaration of the action
25. out.println("<go
    href=\"http://localhost:8080/examples/servlet/Call?choice=$choice\"/>");
    //command to call.class file with the user's choice as input
    //is taken in the variable choice
26. out.println("</do>");
27. out.println("<p align=\"left\"><b> 1. Weather Report</b></p>");
28. out.println("<p><b> 2. Question of the Day</b></p>");
    // displaying two choices on screen
29. out.println("<p><b> Please enter a choice(1 or 2)</b>");
    // displaying message to input a choice
30. out.println("<input name=\"choice\" format=\"1N\"/>");
    // taking user name as input
31. out.println("</p></card></wml>");
    //Declaring all the commands, which will generate a WML code to
    //display menu on screen
32. }
33. catch(Exception ex)
    // to catch the exception, but nothing is done in the exception
34. {
35. }
36. }
    }
    //End of the java servlet code

```

Code Description

- ◆ Lines 1–2: Importing various packages used for the servlets to run. These include packages for the Input/Output operations, and servlet handling operation.
- ◆ Lines 3–6: A class, which is inherited from the base class HttpServlet, is declared and a method doGet is called, which is responsible for sending the data back to the calling client program.
- ◆ Line 10: Setting the response type of the response to be sent to the client as text/WAP, which enables it to run on a mobile device.
- ◆ Line 11: Obtaining an object of the class PrintWriter. This is used to send the response to the client program.
- ◆ Lines 12–37: Writing the response on the client program in the form of various WML tags.
- ◆ Line 19: Event handling tag in WML is written on to the stream.
- ◆ Line 25: A call to the class Call.java is made after taking the input from the user in the variable name choice.
- ◆ Lines 26–31: Declaring all the commands in WML, which result in a WML menu.

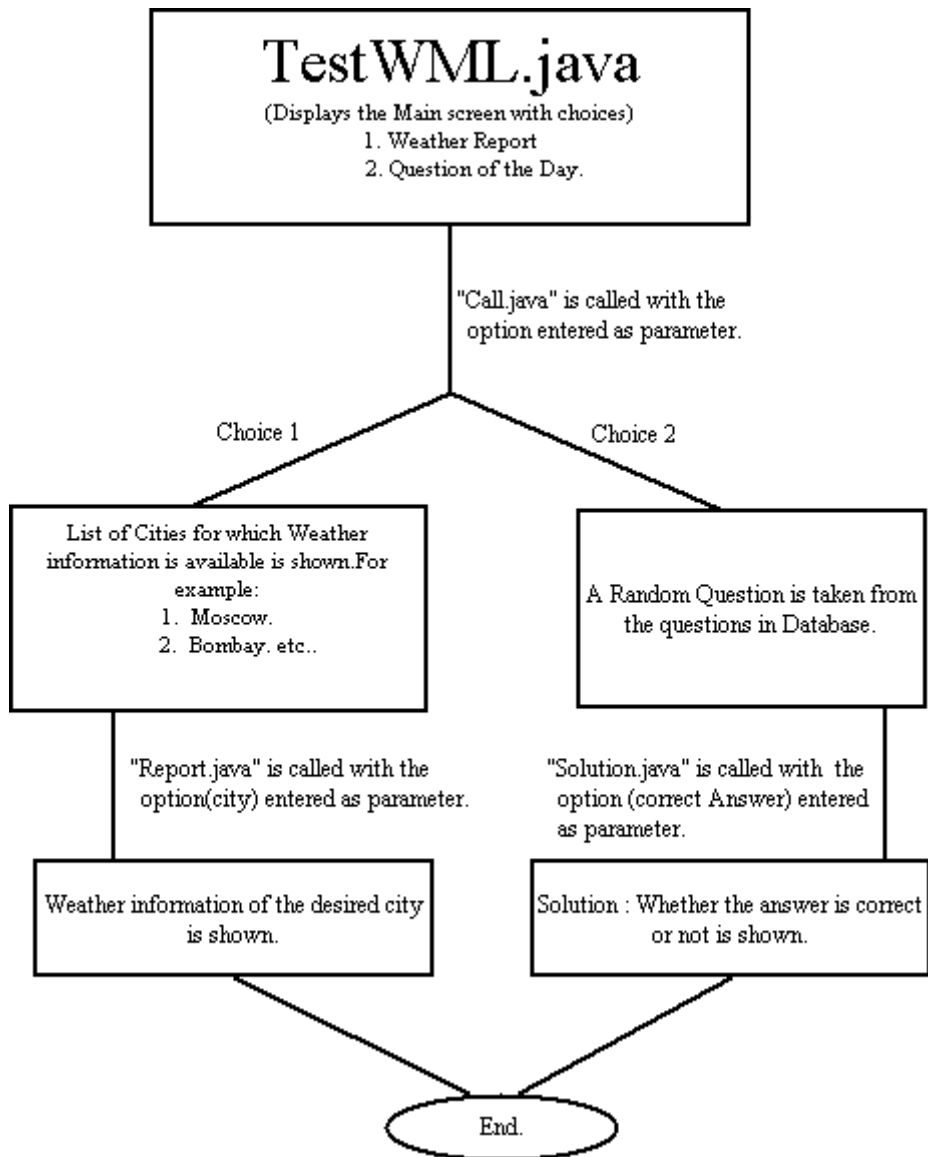


Figure 5-3: Flow diagram of the Weather application

Code Output

Figure 5-4 shows the output of TestWML.java.

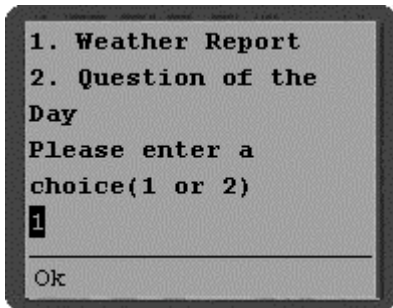


Figure 5-4: Output screen of Test WML.java

Call.java (Listing 5-4) is called from TestWML.java. Further output depends on the choice entered by the user. If the choice entered is “1” it displays the list of cities for Weather Forecast, or if the choice entered is “2” the user is asked a question with multiple-choice answers.

Listing 5-4: Call.java

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1. import java.io.*;
2. import javax.servlet.http.*;
3. import java.util.*;
4. import java.sql.*;
5. public class Call extends HttpServlet
   // declaration of the main class which extends the httpervlet
6. {
7.     PrintWriter out;
8.     String value;
9.     Connection con;
10.    ResultSet rs;
11.    Vector name = new Vector();
12.    Statement s;
   // Start of the servlet request
13.    public void doGet(HttpServletRequest req, HttpServletResponse res)
   // start of the action class
14.    {
15.    try
   // declaration of a block to catch exception
16.    {
17.    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
18.    con = DriverManager.getConnection ("jdbc:odbc:testervlet","", "");
19.    s = con.createStatement();
20.    }
21.    catch(Exception se)
   // catching the exception
22.    {
23.    System.out.println(se);
24.    }
25.    try
   // definition of a new block for catching another exception
26.    {
27.    res.setContentType("text/vnd.wap.wml");
   // Setting MIME type as WAP Document.
28.    out = res.getWriter();

```

```

29. value = req.getParameter("choice");
    // getting the value entered by the user in the field of the name
    // choice and storing the result in a string variable of the type value
30. out.println("<?xml version=\"1.0\"?>");
31. _out.println("<!DOCTYPE wml PUBLIC\" \" + \"-//WAPFORUM//DTD WML 1.1//EN\" \" +
    \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    // WML Prolog definition
32. out.println("<wml>");
    // WML deck declaration
33. out.println("<head>");
34. out.println("<meta http-equiv=\"Cache-Control\" content=\"max-age=time\"
    forua=\"true\"/>");
    // WML meta tag declaration
35. out.println("</head>");
36. if (value.equals("1"))
    //checking the value of variable passed from TestWML,
    // for Choice 1 i.e. Weather Report
37. {
38. rs = s.executeQuery("Select * from Weather");
    //Querying from the weather table
39. int i = 0;
40. while(rs.next())
41. {
42. name.add(i, rs.getString(2));
43. i++;
44. }
45. out.println("<card id=\"CardA\" newcontext=\"true\">");
    // WML card declaration
46. out.println("<onevent type=\"onenterforward\">");
    // Defining the event type for the action to take place on submit button
47. out.println("<refresh>");
    // refreshing the client memory
48. out.println("<setvar name=\"WChoice\" value=\"1\"/>");
    // variable declaration
49. out.println("</refresh>");
50. out.println("</onevent>");
51. out.println("<do type=\"accept\" label=\"Ok\">");
    // WML event declaration
52. out.println("<go

href=\"http://localhost:8080/examples/servlet/report?WChoice=$WChoice\"/>");
    // Calling the file report with the choice accepted by the user
53. out.println("</do>"); out.println("<p> The Choices of cities are");
54. out.println("<select name=\"WChoice\">");
    // Defining pull down list to show options for the city names stored in
table
55. Enumeration e = name.elements();
56. int h = 1;
57. while(e.hasMoreElements())
58. {
59. out.println("<option value = \"\"+h+\"\">\" + (String)e.nextElement() +
    "</option>");
60. h++;
    // Filling the pull down list options with the values taken from recordset
61. out.println("</select>");
62. out.println("</p>");

```

```

        // end of paragraph
63. out.println("</card>");
    // End of WML card
64. out.println("</wml>");
    // End of WML Deck
65. }
66. else
    // The else part is to check If Choice entered in TestWML is
    // Question of the Day
67. {
68. rs = s.executeQuery("Select * from Question");
69. int i = 0;
70. while(rs.next())
    // browsing through the recordset
71. {
72. i++;
73. }
74. java.util.Date date = new java.util.Date();
75. long time = date.getTime();
    // defining date and time variables
78. int rem = (int)time/i;
79. int ij = 0;
    // For random number generation
80. ij = (int)time-rem*i;
81. if (ij == 0)
82. {
83. ij = 5;
84. }
85. rs = s.executeQuery("Select * from Question");
    // Adding data to recordset with the result obtained from query
86. int ai = 0;
87. String question = "";
88. String qid = "";
89. while(rs.next())
    // browsing through the recordset
90. {
91. ai++;
92. if (ai == ij)
93. {
94. qid = rs.getString(1);
        // Store the id of the question
95. question = rs.getString(2) ;
        // Stores the question
96. }
97. }
98. rs = s.executeQuery("Select * from Answer");
    // getting the result
99. Vector choices = new Vector();
100. int index = 0;
101. while(rs.next())
    // searching the question in the recordset on the
    // index value of Question ID
102. {
103. if (((String)rs.getString(1)).equals(qid))
104. {
105. choices.add(index,rs.getString(3));

```

```

        // to store the choices in the recordset on the basis of the id
106. index++;
107. }
108. }
109. out.println("<card id=\"CardC\"  newcontext=\"true\">");
110. out.println("<onevent type=\"onenterforward\">");
    // Defining event type in WML
111. out.println("<refresh>");
112. out.println("<setvar name=\"QChoice\"  value=\"1\"/>");
    // setting the variable Qchoice and initialising it to 1.
113. out.println("</refresh>");
114. out.println("</onevent>");
115. out.println("<do type=\"accept\"  label=\"Ok\">");
116. out.println("<go
href=\"http://localhost:8080/examples/servlet/solution?QChoice=$QChoice*Qid="+q
id+
\"/>");
    // A Call to the solution servlet with the Qchoice as a parameter
    // which will contain the option entered by the user, to check
    //whether the answer provided by the user is correct or not
117. out.println("</do>");
118. out.println("<p>"+question);
    // Displaying the question on screen
119. out.println("<select name=\"QChoice\">");
    // Creating pull down list by taking options from the recordset and
    // Displaying it on screen
120. Enumeration e = choices.elements();
121. int h = 1;
122. while(e.hasMoreElements())
123. {
124. out.println("<option value = \""+h+"\">" + (String)e.nextElement() +
    "</option>");
    // Adding options into pull down list from the values in the recordset
    // and displaying it on screen
125. h++;
126. }
127. out.println("</select>");
    // End of pull down list in WML
128. out.println("</p>");
129. out.println("</card>");
    // End of WML card
130. out.println("</wml>");
    // End of WML Deck
131. }
132. }
133. catch(Exception ex)
    // catching the exception & displaying it on screen
134. {
135. System.out.println( "Exception "+ex );
136. }
137. _}
138. }

    //End of java code

```

Code Description

- ◆ Lines 1–4: Importing various packages used for the servlets to run. These include packages for the Input/Output operations, and servlet handling operation.
- ◆ Lines 5–13: A class is declared. This class is inherited from the base class `HttpServlet` and a method `doGet` is called which is responsible for sending the data back to the calling client program.
- ◆ Lines 17–19: Initializing the database driver — make a connection with the database and initialize the record set.
- ◆ Lines 27–28: Setting the content type as a text/WML document, which allows execution on a WAP-enabled device. Obtaining an object of the class `PrintWriter`. This is used to send the response to the client program.
- ◆ Line 29: Getting the choice entered by the user in `TestWML.java`.
- ◆ Line 30: Writing the response on the client program in the form of various WML tags.
- ◆ Lines 36–38: First the program checks whether the choice entered is 1 or 2. If it is 1, the Weather Report is generated and the database is queried for the names of all the cities for which information is available. If the choice entered is 2, the Question of the Day class is generated.
- ◆ Line 46: Event-handling routine for the event generated by the user for determining the weather of a particular city.
- ◆ Lines 54–63: Defining the pull-down menu to show the cities available after querying the database.
- ◆ Line 66: Question of the Day option is selected.
- ◆ Line 79: A random question is generated.
- ◆ Lines 80–114: Showing the question on-screen in a proper format.
- ◆ Line 116: Call to the solution servlet is made, which takes as a parameter the value entered by the user and matches it with results from the database.
- ◆ Lines 117–138: Exception handling and closing of the database connection is complete.

Code Output

Figure 5-5 shows what happens if choice 1 in `Call.java` is selected; Figure 5-6 shows what happens when choice 2 in `Call.java` is selected.

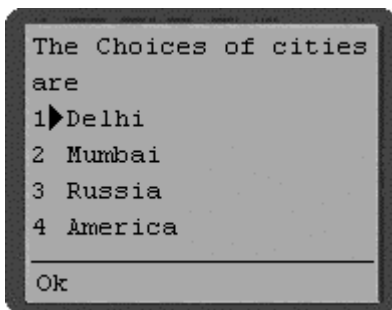


Figure 5-5: List of cities

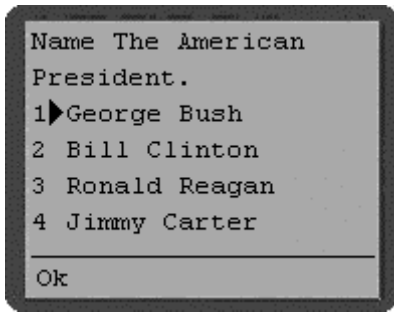


Figure 5-6: Question of the day

The `Report.java` program (Listing 5-5) displays the maximum and the minimum temperature of the cities by accessing the data from the MS Access database.

Listing 5-5: Report.java

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```
// importing the various java packages used by servlets
1. import javax.servlet.http.*;
2. import java.io.*;
3. import java.util.*;
4. import java.sql.*;
5. public class Report extends HttpServlet
6. {
    // Declaration block defining variables and recordset
7.    PrintWriter out;
8.    String value;
9.    Connection con;
10.   ResultSet rs;
11.   Vector name = new Vector();
12.   Statement s;
13.   String temp_max = "";
14.   String temp_min = "";
    // Declaration block for declaring string, connection and resultsets etc.
15.   public void doGet(HttpServletRequest req, HttpServletResponse res)
16.   {
17.   try
        // Begin of the try block for the exception checking
18.   {
19.   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
20.   con = DriverManager.getConnection("jdbc:odbc:testervlet","","");
        // Setting up of the database connection
21.   s = con.createStatement();
22.   }
23.   catch(Exception se)
        // catching the exception. If any while creating the database connection
24.   {
25.   System.out.println(se);
26.   }
27.   try
28.   {
29.   res.setContentType("text/vnd.wap.wml");
        //setting up MIME type as WML page.
```

```

30. out = res.getWriter();
31. value = req.getParameter("WChoice");
    // To store the value of variable wchoice passed from the calling program
32. int val = Integer.parseInt(value);
33. String city = "";
34. rs = s.executeQuery("Select * from Weather");
    // Executing query to read all the records from table and store in recordset
35. while(rs.next())
36. {
37. if (Integer.parseInt(rs.getString(1)) == val )
38. {
39. city = rs.getString(2);
40. temp_min = rs.getString(3);
41. temp_max = rs.getString(4);
    // Storing the city, minimum and maximum temp in local variables taken from
    //recordset having result passed through query from database
42. }
43. }
44. out.println("<?xml version=\"1.0\"?>");
45. out.println("<!DOCTYPE wml PUBLIC\" \" + \"-//WAPFORUM//DTD WML 1.1//EN\" \" +
    \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
46. out.println("<wml>");
47. out.println("<card id=\"cardB\">");
    // Declaration of card
48. out.println("<p align=\"center\">The temperatures at <b> "+city+" </b> are
as follows </p>");
49. out.println("<p align=\"left\">Min. temp      <b> "+temp_min+"</b></p>");
50. out.println("<p align=\"left\">Max. temp      <b> "+temp_max+"</b></p>");

    // Displaying maximum and minimum temperature of various cities
51. out.println("</card>");
    // End of card
52. out.println("</wml>");
    // End of WML
53. }
54. catch(Exception e)
    // catching the exception and displaying it on the screen
55. {
56. System.out.println( e );
57. }
58. }
// end of the program.
59. }
    End of java code

```

Code Description

- ◆ Lines 1–4: Importing various packages used for the servlets to run. These packages include packages for the Input/Output operations and servlet handling operation.
- ◆ Lines 5–15: A class is declared. This class is inherited from the base class HttpServlet and a method doGet is called, which is responsible for sending the data back to the calling client program.
- ◆ Lines 19–21: Initializing the database driver — make a connection with the database and initialize the record set.

- ◆ Lines 29–30: Setting the content type as a text/WML document, which allows execution on a WAP-enabled device. Obtaining an object of the class `PrintWriter`. This is used to send the response to the client program.
- ◆ Line 31: Setting the choice entered by the user in `Call.java`.
- ◆ Line 35: Writing the response on the client program in the form of various WML tags.
- ◆ Line 34: When the user indicates the choice as 1, the list of all the cities for which the weather information is available is retrieved from the database and displayed.
- ◆ Line 50: Displaying the temperature along with the city name in a proper format to the user.
- ◆ Lines 51–59: Exception handling and closing the WML tags `Output`.

Code output

Figure 5-7 shows the output of `Report.java`.

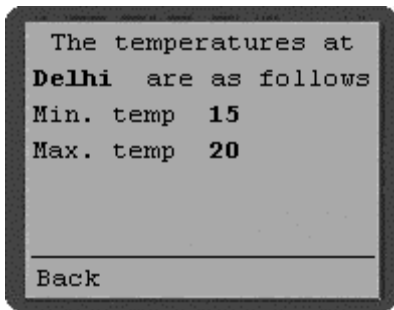


Figure 5-7: Display of max and min temperature

`Solution.java` (Listing 5-6) compares the user's answer to the question of the day with data from the Access database. It then displays a message indicating whether the user's answer is correct or incorrect.

Listing 5-6: `Solution.java`

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1. import java.io.*;
2. import javax.servlet.http.*;
3. import java.util.*;
4. import java.sql.*;
   // Importing the various java packages used by the servlets
5. public class solution extends HttpServlet
6. {
   // Defining a servlet class based on HttpServlet. This class checks the
   //solution on the basis of the choices entered by the user.
   // Variable declarations block
7.   PrintWriter out;
8.   String value;
9.   String id;
10.  Connection con;
11.  ResultSet rs;
12.  Vector name = new Vector();
13.  Statement s;
14.  String temp_max = "";
15.  String temp_min = "";
   // variable declarations
16.  public void doGet(HttpServletRequest req, HttpServletResponse res)

```



```

    // The doGet() method of the servlet having request and response objects
17. {
18. try
    // Begin of the try block for exception checking
19. {
20. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // Setting up the data base connection to access the data from the database
21. con = DriverManager.getConnection("jdbc:odbc:testervlet","","");
22. s = con.createStatement();
23. }
24. catch(Exception se)
25. {
26. System.out.println(se);
27. }
    // End of the try block for the exception checking and displaying the name
    //of the exception
28. try
    // Begin of the try block for the exception checking
29. {
30. res.setContentType("text/vnd.wap.wml");
    // Setting up content MIME type as WML Page
31. out = res.getWriter();
32. value = req.getParameter("QChoice");
    // Setting up variable value with the data of the variable Qchoice, received
    //from the calling program
33. id = value.substring(value.indexOf("=")+1);
34. value = value.substring(0,value.indexOf("*"));
35. System.out.println( value + "    " + id );
    // Breaking the variable value into id and value where the delimiters are
    '=' &
    '*'
36. boolean right = false;
37. rs = s.executeQuery("Select * from Answer");
    // Matches the choices on the basis of question_id and answer supplied by
    //the user
38. while(rs.next())
39. {
40. if (rs.getString(1).equals(id))
41. {
42. if (rs.getString(2).equals(value))
43. {
44. if (rs.getString(4).equals("r"))
45. {
46. right = true;
47. }
    // Checking for the displayed question in the recordset and on finding the
    //question,
    // the result entered by user is matched with the answer in the recordset,
    // if the answer matches, then the variable right is set to true else false,
    // which is later used for condition checking
48. }
49. }
50. }
51. out.println("<?xml version=\"1.0\"?>");
52. out.println("<!DOCTYPE wml PUBLIC\" \" + \"-//WAPFORUM//DTD WML 1.1//EN\" \" +
    \"http://www.wapforum.org/DTD/wml_1.1.xml\">");

```

```

// WML Prolog
53. out.println("<wml>");
54. out.println("<head>");
55. out.println("<meta http-equiv=\"Cache-Control\" content=\"max-age=time\"
forua=\"true\"/>");
    // Definition of meta tag in WML
56. out.println("</head>");
    //Generates a Card to show option entered by the users is correct or not

57. out.println("<card id=\"cardB\">");
58. if (right)
    // Checking for Right or wrong answer in wml
59. {
60. out.println("<p align=\"left\">Your answer is right</p>");
    // Displaying Right answer in wml
61. }
62. else
63. out.println("<p align=\"left\">Your answer is wrong</p>");
    // Displaying wrong answer in wml
64. out.println("</card>");
    // end of WML card
65. out.println("</wml>");
    // end of WML Deck
66. }
67. catch(Exception e)
68. {
69. System.out.println( e );
70. }
    //End of the try block for the exception checking and displaying the name of
    //exception executed
71. }
72. }

```

Code Description

- ◆ Lines 1–4: Importing various packages used for the servlets to run. These packages include packages for the Input/Output operations, and servlet handling operation.
- ◆ Lines 5–15: A class is declared. This class is inherited from the base class HttpServlet and a method doGet is called, which is responsible for sending the data back to the calling client program.
- ◆ Lines 20–22: Initializing the database driver — make a connection with the database and initialize the record set.
- ◆ Lines 30–31: Setting the content type as a text/WML document, which allows the execution on a WAP-enabled device. Obtaining an object of the class PrintWriter. This is used to send the response to the client program.
- ◆ Line 32: Getting the choice entered by the user in Call.java.
- ◆ Line 35: Writing the response on the client program in the form of various WML tags.
- ◆ Lines 38–47: Checking for the displayed question in the recordset, and on finding the question, the answer entered by the user is matched with the answer in the recordset. If the answer matches, then the variable 'right' is set to true.
- ◆ Lines 51–60: Displaying to the user whether the answer is right or wrong.
- ◆ Lines 63–72: Exception handling and closing the WML tags.

Output

Figure 5-8 shows the output of `Solution.java`.

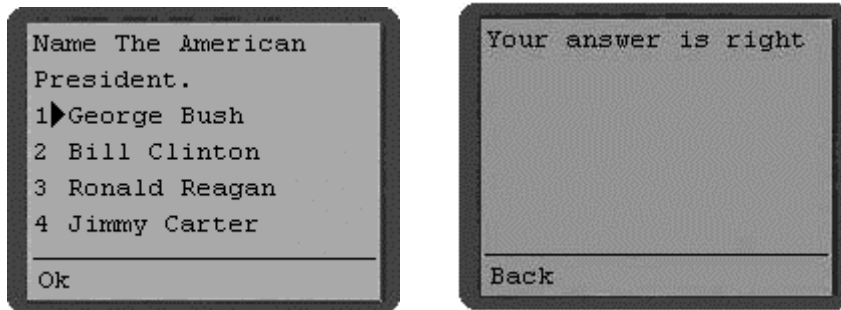


Figure 5-8: Display of the correct option

Complete Execution of the Application

Figure 5-9 shows the complete output of the Weather application.

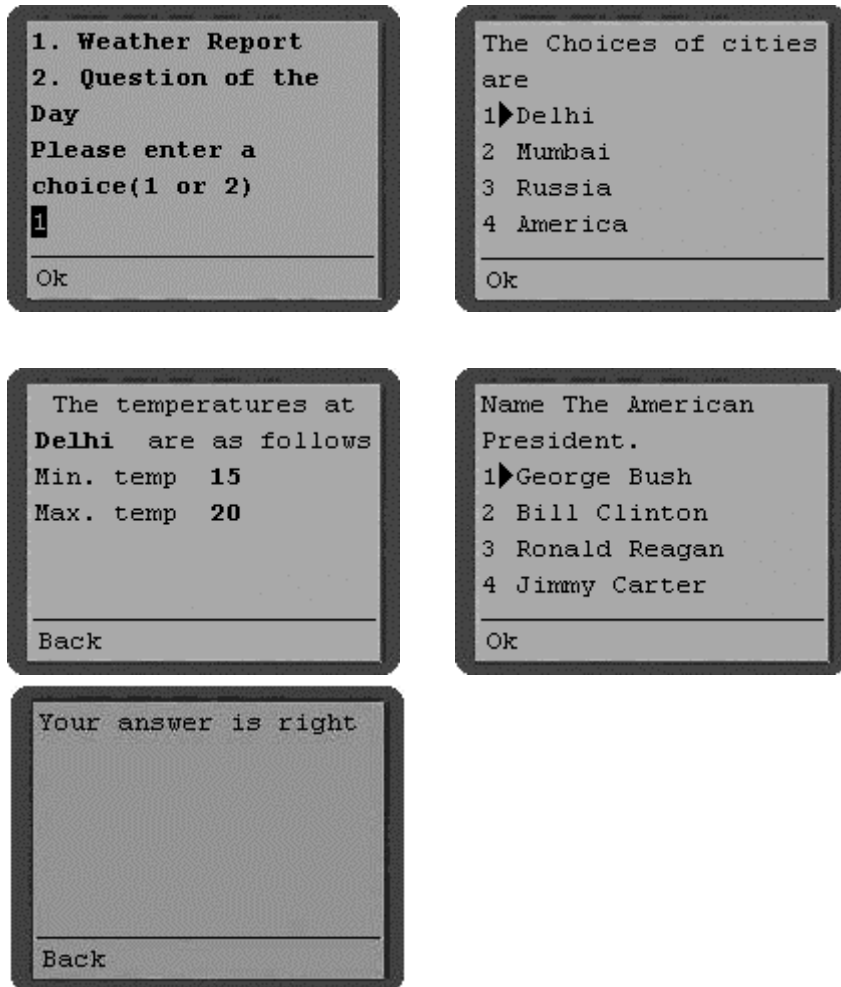


Figure 5-9: Complete Output of the Application

The Weather application has demonstrated the integration of WAP with servlets. The data for the application is taken from the MS Access database.

Summary

Java and Java-related technologies occupy a remarkable position in the area of Internet and intranet application development. Recent research has shown that most of the development in wireless is done on the Java technology platform. This can be attributed to the cross-platform and multiple-device support of Java.

The main objective of this chapter is to provide insight into some of the popular Java technologies that work with WAP, which is used for mobile communication. This chapter only skimmed the surface of the concepts of JSP and servlets. To learn more about these technologies, please refer to the following books and links:

Books

- ◆ Burd, B., *JSP: JavaServer Pages*, Hungry Minds, Inc., 2001.
- ◆ Whitehead, P. and Morasn, R., *Java Server Pages: Your Visual Blue Print to Designing Dynamic Content with JSP*, Hungry Minds, Inc., 2001.

Links

- ◆ http://www.webdevelopersjournal.com/articles/intro_to_servlets.html
- ◆ <http://java.sun.com/docs/books/tutorial/servlets/index.html>
- ◆ <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>
- ◆ <http://www.orionserver.com/taglibtut/>

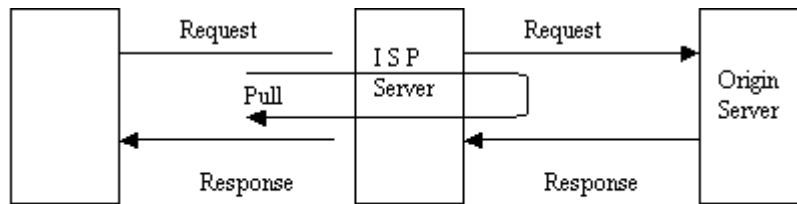
Chapter 6

Push Technology in WAP

One of the most attractive features of WAP is its support for push technology. Push technology allows information to be sent to handsets without an explicit request from the user. Push technology is of immense use in applications such as m-commerce (mobile commerce) and m-advertising (mobile advertising). Special protocols are defined in the WAP framework to support such applications. In this chapter, we will discuss push technology in detail: the protocols, the network elements required to carry out push functions, and how to develop applications using the push model. The complete code listings for practical push applications are also presented; these can be tested with a tool kit that supports the push model.

Pull Technology for Accessing Internet Content

While accessing the Internet, we generally use the pull model, as shown in Figure 6-1. From the desktop, we invoke the Web browser and give a request in the form of a URL (<http://www.iseeyes.com>) to the server of the Internet Service Provider (ISP).



Client with
Web browser

Figure 6-1: The pull model

The server uses a Domain Name Server (DNS) to find out where the information corresponding to the URL is located (called the origin server), and a connection is established with the origin server. The origin server sends the information in the form of an HTML document, which is interpreted by the browser on the desktop. The same methodology is followed while accessing the Internet through the mobile terminals (the handsets). The user gives a request in the form of a URL, which is sent to the WAP gateway, and the gateway forwards it to the origin server to obtain the WML content. The WML content is interpreted by the micro-browser in the handset, and the content is presented to the user. All the programs, which we developed in the earlier chapters, use the same methodology for obtaining the Internet content from the handsets.

This mode of accessing the content is known as the pull model, as we “pull” the information from the server by giving an explicit request. So, the pull model facilitates information on demand.

What Is Push Technology?

In contrast to the pull model, the push model facilitates content to be sent by a server without the need for explicitly requesting information by the user.

As shown in Figure 6-2, a user can register with a server for obtaining information (say, stock quotes) and the server sends the information automatically without the user making explicit requests. In the push model, the following procedure takes place:

1. The user logs into a server and registers her preferences for obtaining the push information (for example, she can log on to the stock exchange portal and make a request for obtaining information on the stock quotes of four companies in which she has interest). While registering the preferences, she will also give the mobile number and the periodicity with which she would like to receive the information (daily, weekly, and so on).
2. The server keeps sending the information with the given periodicity.
3. While sending the push information, the server first sends a “service indication” — a small message to the effect that some information from the stock exchange is to be sent along with the URL that has the actual information. The service indication is just a prompt to the user to find out whether the user would like to view it. The user can respond with a “yes,” after which the actual information will be sent by the server. If the user does not want to see the information at that point in time, the user can respond with a “no.” The user can see the information later, by retrieving the service indication message stored in the handset and invoking the URL present in the message.

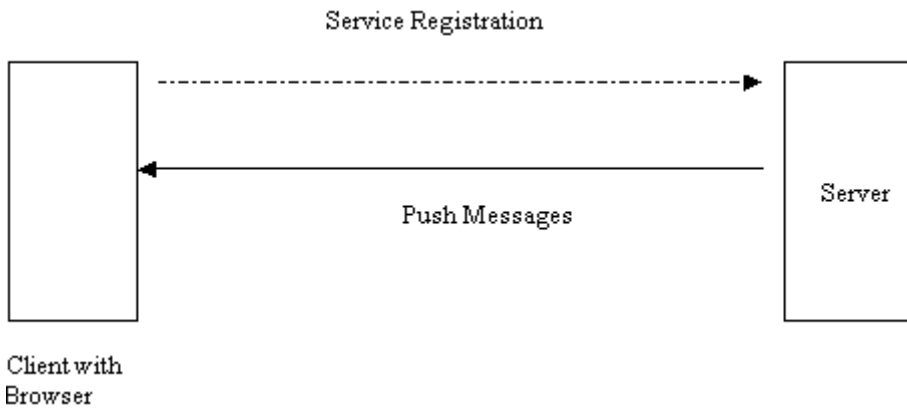


Figure 6-2: The push model

Push Technology Applications

Push technology is now being used for providing a number of value-added services to the users. Some of these applications are

- ◆ Users can be sent daily information on stock quotes. For example, every day when the NASDAQ closes at 1 p.m., the stock prices can be sent immediately to the interested users.
- ◆ Users can obtain sports information such as soccer/cricket scores periodically when the match is in progress — there’s no need to send a request every time.
- ◆ Users can register with an astrology portal and obtain a daily horoscope.
- ◆ Service organizations can keep their customers updated with the latest information and new services to be offered.
- ◆ When used in intranet and extranet environments, push technology helps in providing effective customer-relations management.
- ◆ Sales/customer support staff who are on the move can be informed of the arrival of new mail in their mailboxes or about a service call that needs immediate attention.

- ◆ Business houses can use the push framework for advertising. Known as m-advertising, this is expected to be a great revenue earner for the mobile operators. E-commerce proponents feel that push technology would be the right mechanism to encourage impulse buying.

To use an old cliché, imagination is the limit for effectively using push technology in every business sector — service, manufacturing, transportation, hospitality, education, entertainment, and so on.

Push Technology Implementation

The push model can be implemented in two ways:

- ◆ Using the Short Messaging Service (SMS) Server
- ◆ Using Push Proxy Gateway (PPG)

We briefly discuss the Short Messaging Service and then go into the details of the Push Proxy Gateway.

Short Messaging Service

Short Messaging Service (SMS) allows text messages to be sent and received on mobile handsets. The maximum size of the message is limited to 160 characters. SMS has become popular because it doesn't require a heavy infrastructure for the mobile operator, and many value-added services can be provided to the subscribers. You can gauge the popularity of SMS by the fact that nearly one billion messages are exchanged in Europe every month using it.

SMS messages can be exchanged between two handsets or between a handset and a PC (through the Internet). The message sent is stored at the SMS Center and then forwarded to the recipient — SMS uses the store-and-forward mechanism. If the recipient's handset is turned off when the SMS Center has a message to forward, the SMS Center will forward the message as soon as the handset is turned on. If the user is not within the service area, the message will be delivered as soon as he comes within the service area.

The only problem in using SMS is that the text messages are difficult to input. But many handsets support predictive input — software capable of predicting and completing the word being typed.

SMS Architecture

To provide SMS, the mobile operator has to install an SMS Server (known commonly as SMS Center), which is connected to the Mobile Switching Center (MSC) of the GSM network. The operation of the SMS involves two steps:

- ◆ Mobile-originated SMS (from handset to the SMS-C)
- ◆ Mobile-terminated SMS (from SMS-C to the handset)

Mobile-Originated SMS

As shown in Figure 6-3, mobile-originated SMS is handled in two steps, which are as follows:

1. The handset establishes a connection to the network, just as it does for a normal voice call.
2. If the authentication is successful, the handset sends a short message to the SMS-C via the MSC. SMS-C then forwards the message to the destination. The destination can be another handset or a terminal in the fixed network.

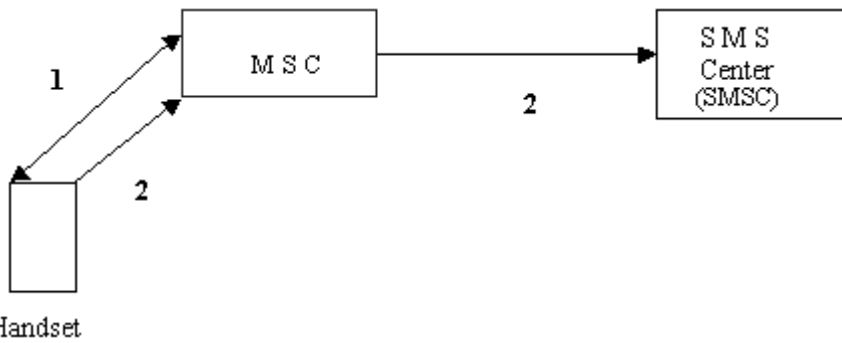


Figure 6-3: Handset-originated short message transfer to SMS Center.

Mobile-Terminated SMS

As shown in Figure 6-4, the short message from a mobile or a fixed terminal is sent to the SMS-C, which in turn is sent to the destination. The procedure is as follows:

1. A user sends a short message to the SMS-C (using the procedure just described).
2. SMS-C sends the message to the Gateway MSC (GMSC). Note that when a service area has more than one MSC, one of the MSCs will be designated as Gateway MSC (GMSC), which forwards the calls/messages to the correct MSC.
3. GMSC queries the Home Location Register (HLR) and obtains the routing information.
4. HLR sends the routing information to the GMSC.
5. GMSC routes the message to the corresponding MSC.
6. The handset is paged and a connection is set up like a normal call setup.
7. MSC delivers the message if the authentication is successful.
8. If the delivery is successful, a delivery report is sent by the MSC to the SMS-C. If the delivery is not successful, the information that the delivery was not successful is stored in the HLR. Whenever the network can access the handset, the HLR informs the SMS-C to resend the message.

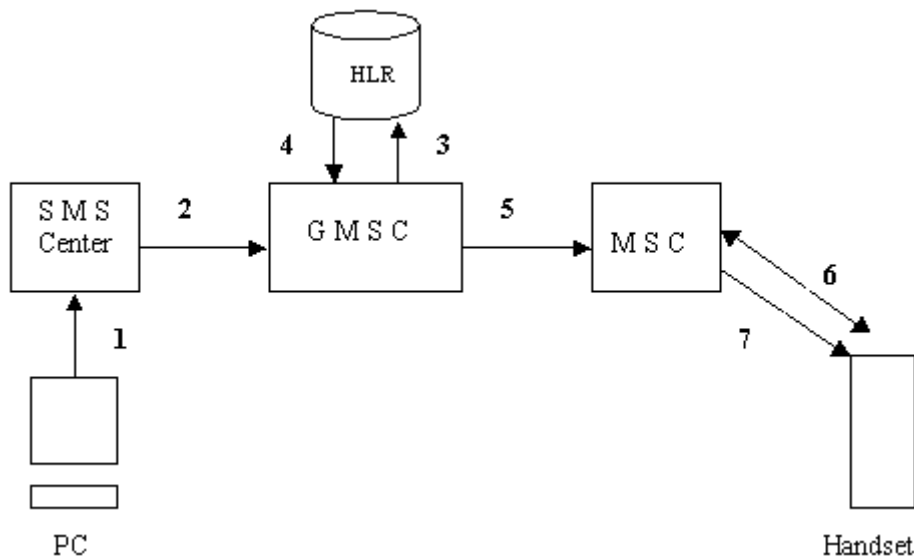


Figure 6-4: Handset-originated short message transfer to SMS Center

Configure the Handset for SMS

To use the SMS, you must configure the handset to register with the SMS Center. The procedure is as follows:

1. Go to the Messages option in your handset.
2. Under Message Service Center number, enter the number (a mobile number given by the operator). This is a one-time process. The handset is now ready for SMS.
3. To send a message, go to the Text Messages menu and select Write Message.
4. Type the message.
5. Press OK and enter the mobile number to which the message will be delivered. The message is first sent to the SMS Center. The SMS Center then forwards the message to the destination number. The procedure explained in the preceding section will be used for these two operations.
6. You will hear a beep when a new message arrives in your handset. Go to the menu options, and select Read Messages.

Applications Using SMS

The most widely used application of SMS is exchanging short text messages. In addition, a number of value-added services can be provided.

The SMS Center can be configured by the mobile operator to send information services such as news, stock quotes, weather information, and local information. The mobile operator can assign short codes such as NEWS for the news service. To access the news service, the user has to type this short code (NEWS) under the Write Messages option. This message has to be sent to the special number allocated by the operator (for example, 500). The SMS Center will forward the news headlines within a few seconds. Similarly, a user can obtain information about net losers and net gainers in the day's stock trading by using a simple code such as STOCK.

You can effectively use SMS to send and receive e-mail from normal PCs as well. To use this option, you must configure the handset just as you do with SMS, with the given Message Service Center number given by the operator.

For sending e-mail from handset to PC: Go to the messages menu and select Write Messages. Type the e-mail ID of the person to whom a message is being sent, enter a space, and then type the message. Send the message to the given number (such as 500 or 600, as specified by the operator).

For sending e-mail from PC to mobile: The handset's e-mail ID will be in the form `yourmobilenumber@operator.com`.

Enter this e-mail ID in the To field, compose the message, and send it (as you have done before). The message will go to the SMS Center, which will forward the mail to the handset.

You can also do mobile banking by using the SMS. For this service, the operator will again assign a special SMS Center number. You must configure this number in the handset as described earlier. To access the services of the bank PQR, you must type the message in one of the following ways:

- ◆ PQR BAL — to obtain the balance in the bank account
- ◆ PQR CHKBKREQ — to request a checkbook

Special services: You can configure the SMS Center to provide special services such as renting a car, ordering a bouquet, and so on.

To make use of the SMS bearer for WAP services by using the push model, the only requirement is to configure the WAP server to use the SMS bearer. In the WAP server manager, you must complete bearer adapter configuration settings to indicate that the bearer is SMS.

Limitations of SMS

Although SMS can be used to provide many value-added services, the maximum size of the message is limited to only 160 characters. To send a message longer than 160 characters, you must split the message. This is not a user-friendly mechanism.

Another limitation of SMS is that there's no effective interaction between the server and the user — SMS is based on a store-and-forward mechanism. To carry out transactions, as in m-commerce, interactivity is a must.

If the push messages are limited to 160 characters, SMS can work as the bearer for the WAP push services. If the messages are longer or when interactivity is required, SMS is not an attractive bearer.

Another problem in using SMS for WAP services is overloading. Because the SMS Server has to cater to the normal text messages in addition to the WAP services, the SMS server can get overloaded due to heavy traffic and may even crash. To avoid overloading, WAP service providers can use a separate front-end server and not connect directly to the SMS Server.

Push Framework in WAP

The push framework is shown in Figure 6-5.

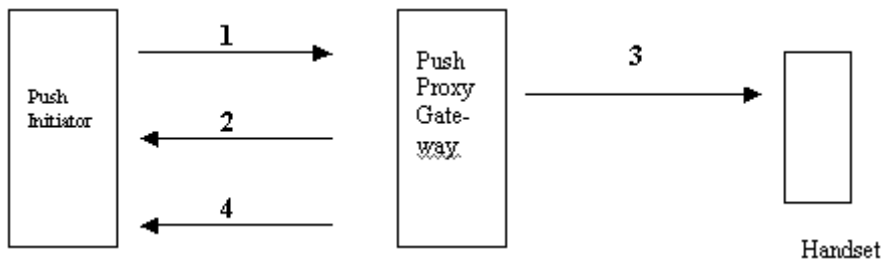


Figure 6-5: Push framework

A Push Initiator (PI) in the Internet domain initiates the push messages. The Push Initiator can be a Web server, which also provides the WAP services, or a WAP server, providing only WML content. A Push Proxy Gateway (PPG) interfaces between the Internet domain and the wireless network domain. The PPG obtains the push messages from the Push Initiator and sends it to the designated handset over the mobile network. Note that the PPG is a logical entity and physically can be combined with the WAP server.

You need two special protocols for pushing the messages, as shown in Figure 6-6. One protocol is required between the Push Initiator and the Push Proxy Gateway to exchange control information and the content to be sent to the handset; this protocol is known as Push Access Protocol (PAP). Another protocol is required to transmit the push message to the handset; this protocol is the Push Over The Air (OTA) protocol.

As the Push Initiator (PI) and the Push Proxy Gateway (PPG) are connected to the Internet, the Push Access Protocol uses the HTTP for exchanging the information between the PI and PPG. As the PPG communicates with the handsets in the WAP domain, the Push Over The Air Protocol uses the Wireless Session Protocol (WSP) for exchanging information between the PPG and handset.

Push Message Processing

The push messages are delivered to the handset by using a five-step procedure. The four steps indicated in Figure 6-5 and described below are compulsory, and the fifth step is optional.

1. The Push Initiator sends a Push initiation message to the PPG. This message consists of control information such as delivery instructions and the actual content in WML format.

2. The PPG sends a push submission acceptance or rejection message to the Push Initiator. The PPG must send an acceptance or rejection. The message can be rejected if it does not meet the Document Type Definition (DTD).
3. The PPG sends the push message to the handset using the Push Over The Air Protocol. Before sending the message to the handset, the PPG may encode the WML content. The WML content is encoded into compact binary format for a faster transfer on the mobile network. The PPG implementation may include tests to be carried out on the push messages, such as whether the message has crossed the expiration time, whether the push message has been cancelled, and so on.
4. The PPG sends a result notification to the PI. This is to indicate whether the push message has been successfully delivered, is pending, or is undeliverable.
5. The PI may send a message to the PPG in the form of a status query to find out the status of a particular push message. The PPG has to respond with the message status on receipt of the status query message from the PI. This is an optional step. Some PPGs support cancellation of the push messages. In this case, the PI can send a cancellation message to the PPG, and the PPG will cancel the delivery of the push message.

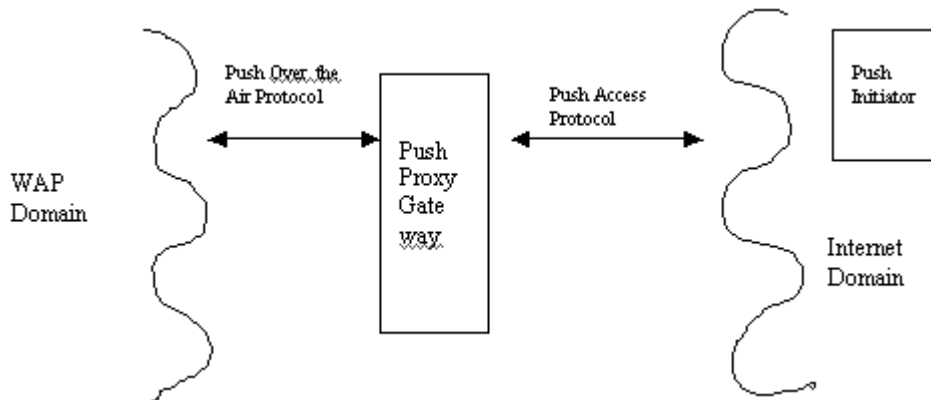


Figure 6-6: Protocols for push model

Push Access Protocol

The Push Access Protocol (PAP) is used to exchange information between the PI and PPG. The messages are exchanged in the form of XML entities. The PAP carries out five operations:

- ◆ Push Submission (from PI to PPG)
- ◆ Result Notification (from PPG to PI)
- ◆ Push Cancellation (from PI to PPG)
- ◆ Status Query (from PI to PPG)
- ◆ Client Capability Negotiation (from PI to PPG)

Push submission

The push submission message contains the control information in the form of an XML entity containing the delivery instructions and the content in WML format.

Result notification

The result notification is in XML format from PPG to PI to indicate whether content is delivered to the handset. Table 6-1 gives the various PAP attributes and the status codes to be sent by the PPG.

Table 6-1: PAP Attributes and Status Codes

<i>PAP Attribute</i>	<i>Status</i>
Message-state	undeliverable pending expired delivered aborted canceled
Code	transformation failed
Desc	implementation dependent value (description of problem)
event-time	time of failure time of delivery
delivery method	unconfirmed confirmed

Push cancellation

The push cancellation message is sent from PI to PPG. The PI sends this message if a push message is already submitted but should not be dispatched. This message is also in XML format.

Status query

The PI can send a status query in XML format to the PPG to find out the status of a specific push message. The PPG will respond with a message indicating whether the message has been delivered, is pending, or is undeliverable.

Client capability negotiation

The PI can send a message indicating the assumed capabilities of the client (the handset). Note that in a wireless network, the clients can have different capabilities in terms of memory, display size, and so on. Because of this, the WAP content may not appear uniformly on all handsets. More work needs to be done in the specifications arena to take care of handsets with different capabilities.

Because the Push Access Protocol is in the Internet domain, PI and PPG exchange messages by using the HTTP/1.1 protocol over the TCP/IP protocol suite.

Push Over The Air Protocol

Push Over The Air (OTA) Protocol runs above the Wireless Session Protocol (WSP). Three types of push functionality are supported:

- ◆ **Confirmed data push during a session:** An existing Wireless Session Protocol (WSP) session is used to push the data. The server receives an acknowledgement after successful push operation.
- ◆ **Non-confirmed data push during a session:** An existing WSP session is used to push the data, but there is no acknowledgement after the push operation.
- ◆ **Non-confirmed data push without an existing session:** In this case, the server pushes the data without establishing a session. Connectionless push is carried out on a Wireless Datagram Protocol (WDP) port, which the PPG uses to make a request to the client (the handset) to establish a connection.

Only a client can create a session — the PPG cannot create a session to push the message. The PPG will send a request to the client to establish a session. In the client, a small program called Service Initiation Application (SIA) will be running. The PPG sends a request to the SIA in the client to create a push session. The client then establishes a session with the PPG to receive the push message. Another piece of software that has to run on the client is the Application Dispatcher. When a client receives pushed content, the dispatcher looks at the push message header to find out to which application the message must be sent — the message, for example, can be sent to a microbrowser, an e-mail reader, or a scheduler.

Push Messages

Two types of push messages exist:

- ◆ Service Indication (SI)
- ◆ Service Loading (SL)

Service Indication (SI) messages are to send an alert to the handset. The Service Indication is sent to the handset based on the preferences the user has registered earlier for obtaining the push content. The message can be used to inform a user that a new mail message has arrived in a mailbox, to indicate that news headlines are waiting to be displayed, to send advertisements, to remind about the credit card payment to be made, or to inform that the latest stock quotes are to be displayed. SI messages are used to avoid intruding into the present activity. The content type format of this message is text/vnd.wap.si.

The Service Indication is just that — only an indication; this means the actual service will not be loaded (the actual information will not be displayed). The SI consists of a small message to the user about the event and a Uniform Resource Locator (URL) from which the service can be loaded. If the user gives her consent to load the message, the actual message will be obtained from the server and displayed on the user's handset. This is known as Service Loading.

The mechanism of the Service Indication and Service Loading are shown in Figure 6-7. The operation involves the following steps:

1. The PI sends the SI to the PPG in XML format.
2. The PPG forwards the SI to the handset after necessary encoding. The message will contain a small alert such as “Would you like to view the stock quotes?” and the URL. The user may respond with a Yes to receive the actual message.
3. The URL associated with the alert message is sent to the PPG by using the WSP GET method.
4. The PPG sends the request to the server by using the HTTP GET method.
5. The server sends back the content (in WML content) to the PPG.
6. The PPG encodes the WML content and transmits to the handset. The content is interpreted by the micro-browser and presented to the user.

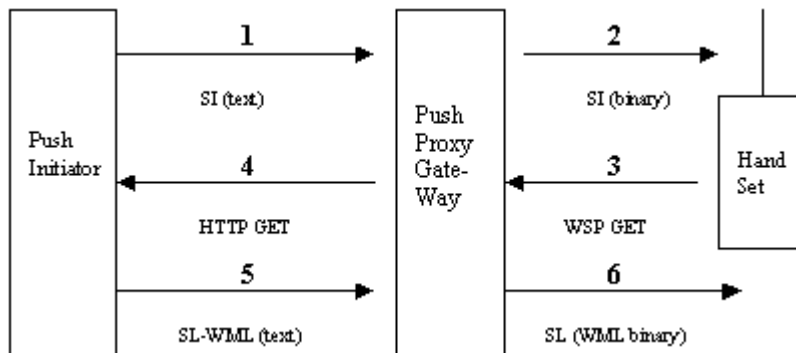


Figure 6-7: Service Indication and Service Loading.

Service Indication (SI)

A typical Service Indication message is as follows:

```

1. <si>
2. <indication href=http://stockquote.com/wel.wml si-id=customerno1020"
   created='2001-07-01T13:00:00Z"
   si-expires="2001-07-02-2001T23:59:05"
   action="signal-medium"

```

```

    Would you like to view stock quotes now?"
3. </indication>
4. <info>
5. <item class="moreinfo">
    Your last update was on 05-01-2001
6. </item>
7. </info>
8. </si>

```

The Service Indication attributes are shown in Table 6-2.

Table 6-2: Service Indication Attributes

Attribute	Explanation
Href	URL to a service application designed to provide the content. Along with the alert message, this URL is sent to the handset, and if the user decides to obtain the actual information, this URL is invoked.
si-id	Unique IS assigned to each message
created	Date and time of creation of content specified by the URI. This will be in the format YYYY-MM-DDThh:mm:ssZ, where YYYY represent four digits for the year; MM represent two digits for the month; DD represent two digits for the day; hh represent two digits for the hours; mm represent two digits for the minutes; ss represent two digits for the seconds. The time has to be represented in the 24-hour time keeping format. The letters T and Z should appear literally.
si-expires	Service indication expiration date and time
Action	Level of intrusiveness. The action can be any of the following: signal-none, signal-low, signal-medium, signal-high, delete, with default as the signal-medium. (See the next section).
Class	Subelement of info element, for additional information to be provided.

Levels of intrusion

To ensure that the push messages do not disturb the user, levels of intrusion have been defined in the WAP framework. Suppose that the user is composing a short message, and suddenly the push message is delivered, to the dismay of the user. The action attribute in the Service Indication is used to indicate to the user what type of action needs to be taken based on the present activity of the user. But this is only indicative because of the varying capabilities of the handsets; the `action` attribute may not provide the desired result.

When action is signal-high, there may be intrusion — even if the user is busy with something else, the message will be displayed. When the action is signal-medium, the Service Indication must be presented in a non-intrusive manner. When the action is signal-low, the SI can be postponed and presented later. When the action is delete, the message can be deleted. The default action is signal-medium. Generally, all actions are client-implementation dependent. Based on the implementation, the SI can be stored in the handset and presented later. Expired messages need not be presented to the user.

Service Loading (SL)

As mentioned earlier, the Service Indication gives an alert to the user. This alert also contains the URL from which the actual content will be obtained. When the user responds to the alert message indicating his interest in seeing the content, the content will be obtained from the URL and presented to the user. If the user responds negatively to the alert message, the Service Indication will be stored in the local memory of the device for later retrieval by the user.

The content type format for SL message is text/vnd.wap.sl.

The following is a Service Loading example:

```
<sl>
  href http://www.stockquote.com/abc.wml"
  action="execute-low"
</sl>
```

The action can be:

- ◆ Execute-low — without intrusion
- ◆ Execute-high — may result in intrusion
- ◆ Cache — placed in cache.

Push Proxy Gateway

Based on the discussion on the push protocols, we can summarize the functions of the PPG:

- ◆ **Push initiation identification and authentication:** The PPG has to get the push initiation messages from the PI. It also has to ensure that the messages are being received from a genuine PI using the authentication mechanisms generally used in the Internet environment.
- ◆ **Protocol conversion:** Because the PPG has to talk to the PI in the Internet domain and the handset in the wireless domain, it has to do the protocol conversion (like the WAP server). So the PPG can be integrated with a WAP server as well.
- ◆ **Binary encoding:** For transmitting the WML content on the wireless network efficiently, the WML code is converted into binary format by the PPG. Again, this functionality is the same as that of the WAP server.
- ◆ **Content transmission:** The PPG has to use the Push OTA protocol to transmit the content to the handsets. As a part of this functionality, the job includes ensuring that the messages are delivered to the handset, and if not, to inform the PI using the Push Access Protocol.

To test the applications in a laboratory environment, you can use the tool kit. Invoke the tool kit and select the Device Settings; then check the following items:

- ◆ Listen to push messages
- ◆ Auto-activate push messages when received
- ◆ Set the push settings
- ◆ Connectionless mode
- ◆ Connection-oriented mode

In the push message simulator, specify the URL for service indication. Note that the Service Indication is automatically generated in the tool kit. But in practical applications, the developer needs to write the code as illustrated in the section “Service Indication.”

We will now build two applications by using push technology — one for pushing stock quotes and one for pushing advertisements with a mobile cart. To develop these applications, we must write servlet applications.

Develop the Database and Servlet Applications

Here's a brief review of how to build servlets for WML content. The requirements for building servlets are the following:

- ◆ JDK1.2.2 or above
- ◆ JSWDK 1.0.1 (Java servlets development kit)
- ◆ Nokia tool kit 1.2 or 1.3 or 2.1

Create a directory in the `jswdk` installation directory to keep all class files in that directory.

Perform the following to test the servlet application:

1. Create a subclass of HTTP Servlet.
2. Set content type to WML; write WML content to the output stream.
3. Compile the servlet and move the servlet class to the JSWDK servlet directory.
4. Add the servlet entry to the servlet property file.
5. Set the mime types on the server.
6. Add an entry to the Web server XML file to map the URL to your servlet directory.
7. Start the servlet server.
8. Use the Nokia tool kit to test the application.

Configure the servlet engine

To configure the servlet engine, you must edit the `mime.properties` file and add the following lines:

```
wml=text/vnd.wap.wml
wbmp=image/vnd/wap/wbmp
wmlc=application/vnd.wap.wmlscript
wmls=text/vnd.wap.wmlscript
wmlsc=application/vnd.wap.wmlscript
```

Edit `webserver.xml` as follows to group your servlets into separate services and locations:

```
<webapplication id="myprogram" mapping="/myfile" docBase="myprogram"/>
```

Test the Application

To test the application, follow these steps:

1. Start the server by typing **startserver** in the base directory of the `jswdk` directory.
2. Start the Nokia emulator and click the load location item on the Go menu.
3. Enter the URL of the servlet in the location box.
4. The content appears.

Push Application Development

For push application development, follow these steps:

1. Register personal preferences by registering with a Web server. Design an HTML page that takes the personal preferences for the given application, the mobile number, and the periodicity with which the information needs to be pushed.
2. Store the information in a database.
3. Create a servlet that checks periodically for when the information must be pushed.
4. Use the push simulator in the tool kit to simulate the push application.

Application: Pushing the Stock Quotes

The objective of this project is to push stock quotes to a handset periodically. A user has to register with a server, providing her mobile number, the periodicity with which she would like to receive the stock quotes (daily, weekly), and the names of the companies for which she would like to get the stock quotes. When the user logs on to the server, a registration form (Figure 6-8) must appear, which she can fill up and submit. The information is then stored in a database. A servlet will be running on the server that keeps track of the information to be pushed to the handsets and pushes the information with the required periodicity.

Listing 6-1 shows the HTML code for creating the registration form.

Listing 6-1: HTML Code for Registration

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1. <html>
2. <head>
3. <title>STOCK REGISTRATION</title>
4. </head>
5. <body bgcolor="#FFFFFF">
6. <div align="center">
7. <table width="640" border="0" height="350" bordercolordark="#FFFFFF">
8. <tr align="left" valign="top">
9. <td>
10. <form method="post" action="http://localhost:8080/RegStock" name="reg">
11. <h3 align="center"><font face="Arial, Helvetica, sans-serif">STOCK
    REGISTRATION FORM</font></h3>
12. <pre><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
13. <!--Code For User Input -->
14. NAME : <input type="text" name="name">
15. MOBILE PHONE NO : <input type="text" name="cellno">
16. ADDRESS : <input type="text" name="add1">
17. <input type="text" name="add2"> <input type="text" name="add3">
18. CITY : <input type="text" name="city">
19. E-MAIL ID : <input type="text" name="mail">
20. PREFERENCES :
21. <input type="checkbox" name="satyam" value="Yes">Satyam
22. <input type="checkbox" name="infosys" value="Yes">Infosys
23. <input type="checkbox" name="wipro" value="Yes">Wipro

```

STOCK REGISTRATION FORM

NAME	:	
MOBILE PHONE NO	:	
ADDRESS	:	
CITY	:	
E-MAIL ID	:	
PREFERENCES	:	
		☐ Satyam
		☐ Infosys
		☐ Wipro
		☐ IBM
		☐ Reliance
SEND DETAILS:		
		☒ Hourly During Stock Hours
		☐ Once A Day
		☐ Twice A Day
Submit Reset		

Figure 6-8: Registration form

```

24. <input type="checkbox" name="IBM" value="Yes">IBM
25. <input type="checkbox" name="reliance" value="Yes">Reliance
    SEND DETAILS:
26. <input type="radio" name="send" value="hourly" checked>Hourly
    During Stock Hours
27. <input type="radio" name="send" value="onceaday">Once A Day
28. <input type="radio" name="send" value="twiceaday">Twice A Day
29. <center>
30. <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
31. <!-- Code For Submitting User form -->
32. <input type="submit" name="Submit" value="Submit"><input type="reset"
    name="Reset" value="Reset"></font>
33. </div>
34. </form>
35. </center>
36. </td>
37. </tr>
38. </table>
39. </body>
40. </html>

```

Code Description

- ◆ Lines 1–9: These lines contain the basic header tags for creating the HTML page and setting the color and alignment.
- ◆ Line 10: This line is for creating a form for registration and to post the data in to the database when the form is submitted. “localhost:8080” indicates where the page needs to be submitted.

- ◆ Lines 11–12: The code is for centering the text “STOCK REGISTRATION FORM” and for font definitions.
- ◆ Lines 14–19: This code is for obtaining the user input using the input tags. The name of the user, mobile phone number, address, city, and e-mail ID are obtained from the user.
- ◆ Lines 21–25: This code obtains the preferences — names of companies for which stock quotes are to be obtained. Check boxes are provided and the user has to check the box to obtain the stock quotes corresponding to that company.
- ◆ Lines 27–28: This code is to obtain the periodicity with which the push messages have to be sent. Hourly, once a day, and twice a day are the options provided.
- ◆ Lines 29–30: Centers the input and defines fonts.
- ◆ Line 32: This code is for submitting the form.
- ◆ Lines 33–40: This code corresponds to the end tags. As a good programming practice, close all tags (table, body, HTML).

You must create a database before creating the application. Create two tables like the following two examples (in an application such as Access). Fill the tables in with data corresponding to the names of the companies and their stock quotes today.

CustomerDetails

<i>Field Name</i>	<i>Data Type</i>
CustomerID	Number (generated automatically)
CustomerName	Text
MobileNo	Number (Primary key)
Address1	Text
Address2	Text
Address3	Text
City	Text
Mail	Text
Satyam	Text
Infosys	Text
Wipro	Text
IBM	Text
Reliance	Text
Forward	Text (to indicate the periodicity)

StockPrice

<i>Field Name</i>	<i>Data Type</i>
Company	Text
Price	Number

Listing 6-2 provides the Java program for storing the preferences in the database.

Listing 6-2: Java Program for Storing the Preferences in the Database

© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1.  //Push Application
2.  import java.io.*;
3.  import javax.servlet.*;
4.  import javax.servlet.http.*;
5.  import java.util.*;
6.  import java.sql.*;
7.  public class RegStock extends HttpServlet
8.  {
9.      String[] cuno;
10.     Connection con;
11.     String url,mg="";
12.     String cname="";
13.     String ccellno="";
14.     String cadd1="",cadd2="",cadd3="",ccity="";
15.     String cmail="";
16.     String csatyam="No",cinfosys="No",cwipro="No",cibm="No",creliance="No";
17.     String csend="",temp="";
18.     String s1,s2,s3,s4,s5,ino1,max1,s6,s7,ii;
19.     ResultSet rs,rs1,rs2,rs3,rs4,rs5,rs6,rs7;
20.     Statement stmt,stmt1,stmt2,stmt3,stmt4,stmt5,stmt6,stmt8;
21.     int c1=0,c11,c11,ino=0,ino2,cno1,phno,cfx,ccount,cuno1,ii2,max=0;
22.     public void init(ServletConfig sc) throws ServletException{
23.         /* CONNECTING TO THE DATABASE */
24.         try{
25.             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
26.             con = DriverManager.getConnection("jdbc:odbc:stock");
27.             stmt = con.createStatement();
28.             stmt1= con.createStatement();
29.         }catch(Exception e){System.out.println(e);}
30.     }
31.     public void doGet(HttpServletRequest request,
32.         HttpServletResponse response) throws ServletException,IOException
33.     {
34.         /* DESIGN A FORM FOR DISPLAYING DATA */
35.         PrintWriter out = response.getWriter();
36.         try{
37.             /*          Count No of Customers in the Database          */
38.             rs=stmt1.executeQuery("select count(*) from CustomerDetails");
39.             while(rs.next())
40.             {
41.                 /* Retrieve the Record from the Database          */
42.                 ino1=rs.getString(1);
43.                 /* Parse and Increment the Customer Id          */
44.                 ino =Integer.parseInt(ino1);
45.                 ino2 =ino + 1;
46.             }
47.             rs.close();
48.             /* Get Info of the Customer          */
49.             cname=request.getParameter("name");
50.             ccellno=request.getParameter("cellno");
51.             cadd1=request.getParameter("add1");

```

```
52. cadd2=request.getParameter("add2");
53. cadd3=request.getParameter("add3");
54. ccity=request.getParameter("city");
55. cmail=request.getParameter("mail");
56. /*      Get Info of the Company, Assign it to a temp Variable and Check for
    !NULL
57. */
58. temp=request.getParameter("satyam");
59. if(!(temp == null))
60. {
61. csatyam= temp;
62. }
63. temp=request.getParameter("infosys");
64. if(!(temp == null))
65. {
66. cinfosys= temp;
67. }
68. temp=request.getParameter("wipro");
69. if(!(temp == null))
70. {
71. cwipro= temp;
72. }
73. temp=request.getParameter("IBM");
74. if(!(temp == null))
75. {
76. cibm = temp;
77. }
78. temp=request.getParameter("reliance");
79. if(!(temp == null))
80. {
81. creliance = temp;
82. }
83. csend=request.getParameter("send");
84. /*      Insert Details of the User into the Database */
boolean x=stmt.execute("insert into CustomerDetails values("+ ino2 +",""+
cname +",""+ ccellno +",""+ cadd1 +",""+ cadd2 +",""+ cadd3 +",""+
ccity +",""+ cmail +",""+ csatyam +",""+ cinfosys +",""+ cwipro +",""+
cibm +",""+ creliance +",""+ csend +")");
85. }
86. catch(Exception e1){System.out.println(e1.toString());}
87. /*Write the response to the User */
88. out.println("<html>");
89. out.println("<head>");
90. out.println("<title>Thank You...</title>");
91. out.println("</head>");
92. out.println("<body>");
93. out.println("<p>");
94. out.println("<h3>Thank You For Registering</h3>");
95. out.println("Your CustomerID is : "+ino2);
96. out.println("</p>");
97. out.println("</body>");
98. out.println("</html>");
99. }
100. public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,IOException
101. { doGet(request,response);
```

```
102. }
103. }
```

Code Description

- ◆ Lines 2–6: The code is for importing the necessary class libraries: viz, io, servlet, servlet.http, util, and sql.
- ◆ Line 7: Class declaration.
- ◆ Lines 9–21: The code declares the variables used in the program and the necessary initializations. The name, mobile phone number, address, city, and mail ID obtained in the registration form are taken as the variables for storing in the database. A number of temporary variables are declared. These will be used subsequently.
- ◆ Line 22: Function declaration.
- ◆ Lines 24–30: The code is for connecting to the database using JDBC-ODBC connectivity. Exceptions, if any, are caught through a `try` block.
- ◆ Lines 31–33: Function declaration.
- ◆ Lines 35–47: For every customer who registers, a new customer ID is generated. To achieve this, the already-existing last customer ID in the database has to be obtained. This number is incremented by one and assigned to the new customer. This code is to increment the customer ID based on the previous records stored in the database.
- ◆ Lines 49–55: The information input by the user in the registration form (name, mobile phone or cell number, address fields, city, mail ID) is obtained through this code.
- ◆ Lines 58–82: In the registration form, the user has the option to select any or all of the four companies for which he/she would like to obtain the stock quotes. This code is to check whether the user has selected each of the four companies.
- ◆ Lines 58–62 Used to check for the first company, Lines 63–67 for the second company, and so on.
- ◆ Lines 83–86: Whatever information the user has input in the registration form is inserted in the database.
- ◆ Lines 88–103: A response is sent to the user with the message “Thank You For Registering.” A customer ID is also given.

Listing 6-3 shows the servlet code used to push the stock quote information. The job of this servlet code is to check the user preferences in the database and push the stock quote information to the user. If the customer ID is not valid, a message to that effect will be sent.

Listing 6-3: Servlet to Push the Stock Quote Information

```
//© 2001 Dreamtech Software India Inc.
//All Rights Reserved
```

```
1. //Push Application
2. import java.io.*;
3. import javax.servlet.*;
4. import javax.servlet.http.*;
5. import java.util.*;
6. import java.sql.*;
7. public class StockPrice extends HttpServlet
8. {
9.     String[] cuno;
10.    Connection con;
11.    String url,mg=" ";
12.    String cname=" ";
```

```

13. String ccellno="";
14. String cadd1="",cadd2="",cadd3="",ccity="";
15. String cmail="";
16. String custidstr="";
17. String csend="",temp="";
18. String s1,s2,s3,s4,s5,ino1,max1,s6,s7,s8,ii;
19. ResultSet rs,rs1,rs2,rs3,rs4,rs5,rs6,rs7,rs8,rs9;
20. Statement stmt,stmt1,stmt2,stmt3,stmt4,stmt5,stmt6,stmt8;
21. int c1=0,c11,c11,ino=0,ino2,custidint=0,norows=0;
22. public void init(ServletConfig sc) throws ServletException{
23.     /* CONNECTING TO THE DATABASE */
24.     try{
25.         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
26.         con = DriverManager.getConnection("jdbc:odbc:stock");
27.         stmt = con.createStatement();
28.         stmt1= con.createStatement();
29.     }catch(Exception e){System.out.println(e);}
30.     }
31.     public void doGet(HttpServletRequest request,
32.         HttpServletResponse response) throws ServletException,IOException
33.     {
34.         /* DESIGNIN A FORM FOR DISPLAYING DATA */
35.         PrintWriter out = response.getWriter();
36.         /* Get the Identity of the Customer */
37.         custidstr = request.getParameter("CustId");
38.         custidint = Integer.parseInt(custidstr);
39.         /* Set Content-type Header */
40.         response.setContentType("text/vnd.wap.wml");
41.         /* Write the Response */
42.         out.println("<?xml version=\"1.0\"?>");
43.         out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
44.         \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
45.         out.println("<wml>");
46.         out.println("<template>");
47.         out.println("<do type='prev' label='Back'>");
48.         out.println("<prev/>");
49.         out.println("</do>");
50.         out.println("</template>");
51.         try{
52.             /* Information on the Customer From the Database */
53.             rs=stmt1.executeQuery("select CustomerID from CustomerDetails where");
54.             CustomerID="+custidint);
55.             while(rs.next())
56.             {
57.                 ino =rs.getInt(1);
58.             }
59.             rs.close();
60.             if(custidint == ino)
61.             {
62.                 try{
63.                     rs1 = stmt.executeQuery("select Satyam,Infosys,Wipro,IBM,Reliance from");
64.                     CustomerDetails where CustomerID="+custidint);
65.                     if(rs1.next())
66.                     {
67.                         s1 = rs1.getString(1);
68.                         s2 = rs1.getString(2);

```

```

69. s3 = rs1.getString(3);
70. s4 = rs1.getString(4);
71. s5 = rs1.getString(5);
72. System.out.println("S1 :"+s1+" S2 :"+s2+" S3 :"+s3+" S4:"+s4);
73. }
74. rs1.close();
75. }
76. catch(Exception e1){System.out.println(e1.toString());}
77. /*Print out info on Customer in a Table in 2 Columns */
78. out.println("<card id='card1' title='Stock Quotes'>");
79. out.println("<p>");
80. out.println("<table columns='2' align='LL'>");
81. out.println("<tr><td><strong>Company</strong></td>");
82. out.println("<td><strong>Price</strong></td></tr>");
83. if(s1.equals("Yes"))
84. {
85. try{
86. /* Information on Stock Quotes Customer has Selected(ie,Company &
Price from the Database)*/
87. rs2 = stmt.executeQuery("select * from StockPrice where Company='Satyam'");
88. while(rs2.next())
89. {
90. s6 = rs2.getString(1);
91. s7 = rs2.getString(2);
92. }
93. rs2.close();
94. }
95. catch(Exception e2){System.out.println(e2.toString());}
96. /* Print out info of Customer in a Row */
97. out.println("<tr><td>"+s6+"</td>");
98. out.println("<td>"+s7+"</td></tr>");
99. if(s2.equals("Yes"))
100. {
101. try{
102. rs3 = stmt.executeQuery("select * from StockPrice
where Company='Infosys'");while(rs3.next())
103. {
104. s6 = rs3.getString(1);
105. s7 = rs3.getString(2);
106. }
107. rs3.close();
108. }
109. catch(Exception e3){System.out.println(e3.toString());}
110. out.println("<tr><td>"+s6+"</td>");
111. out.println("<td>"+s7+"</td></tr>");
112. }
113. if(s3.equals("Yes"))
114. {
115. try{
116. rs4 = stmt.executeQuery("select * from StockPrice where Company='Wipro'");
while(rs4.next())
117. {
118. s6 = rs4.getString(1);
119. s7 = rs4.getString(2);
120. }
121. vrs4.close();

```



```

122. }
123. catch(Exception e4){System.out.println(e4.toString());}
124. out.println("<tr><td>"+s6+"</td>");
125. out.println("<td>"+s7+"</td></tr>");
126. }
127. if(s4.equals("Yes"))
128. {
129. try{
130. rs5 = stmt.executeQuery("select * from StockPrice where Company='IBM'");
    while(rs5.next())
131. {
132. s6 = rs5.getString(1);
133. s7 = rs5.getString(2);
134. }
135. rs5.close();
136. }
137. catch(Exception e5){System.out.println(e5.toString());}
138. out.println("<tr><td>"+s6+"</td>");
139. out.println("<td>"+s7+"</td></tr>");
140. }
141. if(s5.equals("Yes"))
142. {
143. try{
144. rs6 = stmt.executeQuery("select * from StockPrice
    where Company='Reliance'");while(rs6.next())
145. {
146. s6 = rs6.getString(1);
147. s7 = rs6.getString(2);
148. }
149. rs6.close();
150. }
151. catch(Exception
152. e6){System.out.println(e6.toString());}
152. out.println("<tr><td>"+s6+"</td>");
153. out.println("<td>"+s7+"</td></tr>");
154. }
155. out.println("</table>");
156. out.println("</p>");
157. out.println("</card>");
158. out.println("</wml>");
159. }
160. //if ends
161. else
162. {
163. out.println("<card id='card1' title='Sorry'>");
164. out.println("<p>");
165. out.println("Your ID Is Not Valid.");
166. out.println("</p>");
167. out.println("</card>");
168. out.println("</wml>");
169. }
170. }catch(Exception e1){System.out.println(e1.toString());}
171. }
172. }
173. public void doPost(HttpServletRequest request,
174. HttpServletResponse response) throws ServletException, IOException

```

```
175. { doGet(request,response); }
176. }
```

Code Description

- ◆ Lines 2–6: This code imports the necessary Java class libraries.
- ◆ Lines 7–21: This code is for declaration of the class and initialization of variables. The same variables used in the earlier program are also used here.
- ◆ Line 22: Function declaration.
- ◆ Lines 24–30: This code is to connect to the database (named stock). The code is kept in a `try-catch` block to catch any exceptions.
- ◆ Lines 31–50: This code generates a WML card to obtain the customer ID from the user. The code consists of a series of `out.println` statements with WML tags to generate the WML code.
- ◆ Lines 51–59: This code checks whether the customer ID is valid.
- ◆ Lines 60–76: The names of companies selected by the customer in the registration form are obtained from the database. This code is kept in a `try-catch` block to catch the exceptions.
- ◆ Lines 78–82: This code generates a WML card containing a table with two columns — company name and price with the card title as “Stock Quotes.” This code is again a set of `out.println` statements with WML tags as arguments.
- ◆ Lines 83–95: If the user selects the first company, the corresponding stock quote is obtained from the database using a `select` query. The company name and the price are kept in the two columns of the first row in the table generated in the code in Lines 74–80.
- ◆ Lines 97–112: When the user selects the second company, again the stock quote is obtained from the database and the company name and the price are kept in the table columns. This process is repeated for the third and fourth company as well in the following lines.
- ◆ Lines 113–126: The above process is repeated for the third company.
- ◆ Lines 127–140: Again, for the fourth company, the above process is repeated.
- ◆ Lines 141–154: This code completes the process for the fifth company.
- ◆ Lines 155–159: These lines generate the WML code for closing the table, the card, and the WML page.
- ◆ Lines 161–176: If the customer ID is not valid (the user sends a wrong ID from the handset), the message “Sorry, Your ID Is Not Valid” appears. The code in these lines generates a WML card to pass this message to the handset.

Figure 6-9 shows the push message simulator. In the push simulator, you don’t need to create the service indication; it is automatically generated in the format described earlier. But in commercial WAP servers that support push, you need to create the Service Indication messages.

Test the Application

Compile the Java files `RegStock.java` and `StockPrice.java`. If you are using a Java Web server, place the class files in the `servlets` folder and the `stock.html` file in the public HTML folder. If you are using JSWDK, place the class files in the `Web-inf` folder and `stock.html` in the `Web pages` folder. Through a browser, invoke the HTML registration form (`stock.html`) and enter the data. A customer ID will appear to the user.

Push Message Editor

SI | SL | CO

Service Indication Headers

Content-Type: text/vnd.wap.si

X-Wap-Application-Id:

X-Wap-Content-URI:

X-Wap-Initiator-URI:

Service Indication Content

action: signal-high

href: http://localhost:8080/Stock/servlet/StockPrice?CustId=1

si-id:

si-expires: 2001-08-22T11:13:50Z

created: 2001-08-21T11:13:50Z

Text Message: This is displayed text.

Class 1:

Info 1:

Class 2:

Info 2:

OK Store Source Store Binary Cancel

Figure 6-9: Push message simulator.

Select the tool kit and the phone emulator (for example, the Blueprint device in the Nokia tool kit). Also select the push view in the tool kit. In the push view, select Create Message. In the HREF, type the URL `http://localhost:8080/Stock/servlet/StockPrice?CustId=1` if you are using the JSWDK and `http://localhost:8080/servlet/StockPrice?CustId=1` if you are using a Java Web server.

Figure 6-10 shows the pushed content on the WAP phone emulator. It also shows the stock information displayed on the WAP phone through push framework. The company names and the stock prices are displayed with the card title Stock Quotes. Figure 6-11 shows the shopping cart registration form.

You can certainly appreciate the power of push technology now; you no longer need to browse the newspaper to search for the stock prices of your favorite companies!

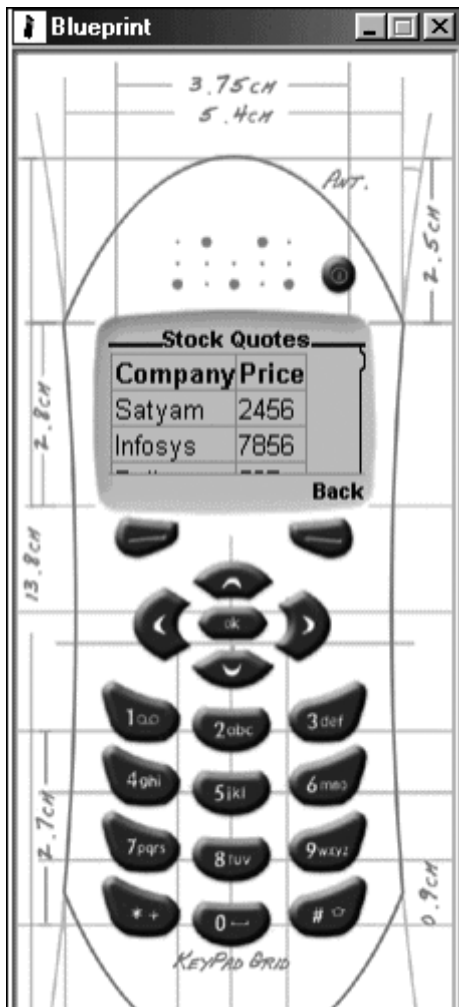


Figure 6-10: Push information displayed on the WAP phone

Application: Shopping Cart with Advertisement Push

In this example, we will develop a virtual mall. A user can register his preferences for selected items (groceries, accessories, clothes) and also request information whenever new products are launched into the market (a very polished way of pushing advertisements!). We will integrate a shopping cart — the user can order an item from the handset. In this example, we will not worry about the payment details (one possibility is to obtain the credit card information, or the bill can arrive with the mobile phone bill if the cellular operator has an agreement with our virtual shopping mall).

SHOPPING CART REGISTRATION FORM

NAME :

MOBILE PHONE NO :

ADDRESS :

CITY :

E-MAIL ID :

PREFERENCES :

☐ GROCERIES

☐ ACCESSORIES

☐ COSMETICS

☐ FASHION WEAR

☐ Inform When ever new Products arrive

SEND DETAILS:

☒ Weekly

☐ Fortnightly

☐ Monthly

☐ Bi-Monthly

Figure 6-11: Shopping cart registration form

As in the previous example, create a database as per the following examples:

CustomerDetails

<i>Field Name</i>	<i>Data Type</i>
CustomerID	Number
CustomerName	Text
MobileNo	Number (Primary key)
Address1	Text
Address2	Text
Address3	Text
City	Text
Mail	Text

Groceries	Text
Accessories	Text
Cosmetics	Text
Fasionwear	Text
NewProduct	Text
Forward	Text

Groceries

<i>Field Name</i>	<i>Data Type</i>
ItemCode	Text
Itemname	Text
ItemPrice	Number

Accessories

<i>Field Name</i>	<i>Data Type</i>
ItemCode	Text
Itemname	Text
ItemPrice	Number

Cosmetics

Same field names and data types as Accessories.

Fashionwear

Same field names and datatypes as Accessories.

NewProduct

<i>Field Name</i>	<i>Data Type</i>
Category	Text
ItemCode	Text
ItemCode	Text
Itemname	Text
ItemPrice	Number

OrderInfo

<i>Field Name</i>	<i>Data Type</i>
OrderNo	AutoNumber
CustomerId	Text
OrderItemsAndQty	Text

Now we need to create an HTML page for the registration form. The registration form is shown in Figure 6-11. The corresponding HTML code is shown in Listing 6-4.

Listing 6-4: HTML Code for Registration

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1.  <html>
2.  <head>
3.  <title>SHOPPING CART REGISTRATION</title>
4.  </head>
5.  <body bgcolor="#FFFFFF">
6.  <div align="center">
7.  <table width="640" border="0" height="350" bordercolordark="#FFFFFF">
8.  <tr align="left" valign="top">
9.  <td>
10. <form method="post" action="http://localhost:8080/RegShopping" name="reg">
11. <h3 align="center"><font face="Arial, Helvetica, sans-serif">SHOPPING CART
    REGISTRATION FORM</font></h3>
12. <pre><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
13. /*      Code For User Input      */
14. NAME          :      <input type="text" name="name">
15. MOBILE PHONE NO :      <input type="text" name="cellno">
16. ADDRESS       :      <input type="text" name="add1">
17. <input type="text" name="add2">
18. <input type="text" name="add3">
19. CITY          :      <input type="text" name="city">
20. E-MAIL ID     :      <input type="text" name="mail">
21. PREFERENCES  :
22. <input type="checkbox" name="groceries" value="Yes">GROCERIES
23. <input type="checkbox" name="accessories" value="Yes">ACCESSORIES
24. <input type="checkbox" name="cosmetics" value="Yes">COSMETICS
25. <input type="checkbox" name="fashion" value="Yes">FASHION WEAR
26. <input type="checkbox" name="newproduct" value="Yes">Inform whenever new
    Products arrive
27. SEND DETAILS:
28. <input type="radio" name="send" value="weekly" checked>Weekly
29. <input type="radio" name="send" value="fortnight">Fortnightly
30. <input type="radio" name="send" value="monthly">Monthly
31. <input type="radio" name="send" value="bimonthly">Bi-Monthly</font></pre>
32. <center>
33. <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
34. <!--      Code For Submitting the form      -->
35. <input type="submit" name="Submit" value="Submit"><input type="reset"
    name="Reset" value="Reset"></font>
36. </div>
37. </form>
38. </center></td>
39. </tr>
40. </table>
41. </body>
42. </html>

```

Code Description

- ◆ Lines 1–9: The basic header tags of the HTML page. This is followed by table tag, along with the necessary attributes.

- ◆ Line 10: Form tag along with the necessary attributes. The attribute method is "post", that posts the information of the form in a database located in the local host.
- ◆ Line 11: The title of the registration form is generated with the necessary font and alignment at center.
- ◆ Line 12: Defines the font for the registration form input fields.
- ◆ Lines 14–20: Code for user-input details such as name, mobile number, address, city, and e-mail ID.
- ◆ Lines 21–26: Code to obtain the user preferences for different items such as groceries, accessories, cosmetics, fashion, new products. The type of input is a check box.
- ◆ Lines 27–31: The periodicity with which the information has to be pushed — weekly, fortnightly, monthly, or bimonthly.
- ◆ Lines 32–33: Tags to center the Submit and Reset buttons and set the font.
- ◆ Lines 34–42: To create Submit and Reset buttons for obtaining the registration form (filled in) from the user. Lines 38–43 are the closing tags for the HTML page.

Now we need to write the Java code that will take the inputs from the registration form and store the information in the database. Listing 6-5 provides this Java code.

Listing 6-5: Java Program for Storing the Preferences in a Database

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.  import java.util.*;
5.  import java.sql.*;
6.  public class RegServlet extends HttpServlet
7.  {
8.      String[] cuno;
9.      Connection con;
10.     String url,mg="";
11.     String cname="";
12.     String ccellno="";
13.     String cadd1="",cadd2="",cadd3="",ccity="";
14.     String cmail="";
15.     String
        cgroceries="No",caccessories="No",ccosmetics="No",cfashion="No",cnew="No";
16.     String csend="",temp="";
17.     String s1,s2,s3,s4,s5,ino1,max1,s6,s7,ii;
18.     ResultSet rs,rs1,rs2,rs3,rs4,rs5,rs6,rs7;
19.     Statement stmt,stmt1,stmt2,stmt3,stmt4,stmt5,stmt6,stmt8;
20.     int c1=0,c11,c11,ino=0,ino2,cno1,phno,cfx,ccount,cuno1,ii2,max=0;
21.     public void init(ServletConfig sc) throws ServletException{
22.         /* CONNECTING TO THE DATABASE */
23.         try{
24.             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
25.             con = DriverManager.getConnection("jdbc:odbc:shop");
26.             stmt = con.createStatement();
27.             stmt1= con.createStatement();
28.         }catch(Exception e){System.out.println(e);}
29.     }
30.     public void doGet(HttpServletRequest request,

```



```
31. HttpServletResponse response) throws ServletException,IOException
32. {
33.     /* DESIGN A FORM FOR DISPLAYING DATA */
34.     PrintWriter out = response.getWriter();
35.     /*          Count No of Customers in the Database          */
36.     try{
37.         rs=stmt1.executeQuery("select count(*) from CustomerDetails");
38.         while(rs.next())
39.         {
40.             /* Retrieve the Record from the Database          */
41.             ino1 =rs.getString(1);
42.             /* Parse and Increment the Customer Id            */
43.             ino =Integer.parseInt(ino1);
44.             ino2 =ino + 1;
45.         }
46.         rs.close();
47.         /* Get Info of the Customer                            */
48.         cname=request.getParameter("name");
49.         ccellno=request.getParameter("cellno");
50.         cadd1=request.getParameter("add1");
51.         cadd2=request.getParameter("add2");
52.         cadd3=request.getParameter("add3");
53.         ccity=request.getParameter("city");
54.         cmail=request.getParameter("mail");
55.         /*          Get Info of the Company, Assign it to a temp Variable and Check for
56.         */
57.         temp=request.getParameter("groceries");
58.         if(!(temp == null))
59.         {
60.             cgroceries= temp;
61.         }
62.         temp=request.getParameter("accessories");
63.         if(!(temp == null))
64.         {
65.             caccessories= temp;
66.         }
67.         temp=request.getParameter("cosmetics");
68.         if(!(temp == null))
69.         {
70.             ccosmetics= temp;
71.         }
72.         temp=request.getParameter("fashion");
73.         if(!(temp == null))
74.         {
75.             cfashion= temp;
76.         }
77.         temp=request.getParameter("newproduct");
78.         if(!(temp == null))
79.         {
80.             cnew = temp;
81.         }
82.         csend=request.getParameter("send");
83.         /*          Insert Details of the User into the Database          */
84.         boolean x=stmt.execute("insert into CustomerDetails values("+ ino2 +",
            '"+ cname
```

```

+ "','"+ ccellno + "','"+ cadd1 + "','"+ cadd2 + "','"+ cadd3 + "','"+
ccity + "','"+
cmail + "','"+ cgroceries + "','"+ caccessories + "','"+ ccosmetics + "','"+
cfashion + "','"+cnew+ "','"+ csend + "')");
85. }
86. catch(Exception e1){System.out.println(e1.toString());}
87. /*Write the response to the User */
88. out.println("<html>");
89. out.println("<head>");
90. out.println("<title>THANK YOU...</title>");
91. out.println("</head>");
92. out.println("<body>");
93. out.println("<p>");
94. out.println("<h3>Thank You For Registering</h3>");
95. out.println("Your CustomerID is : "+ino2);
96. out.println("</p>");
97. out.println("</body>");
98. out.println("</html>");
99. }
100. public void doPost(HttpServletRequest request,
101. HttpServletResponse response) throws ServletException,IOException
102. { doGet(request,response); }
103. }

```

Code Description

- ◆ Lines 1–5: This is the familiar code to import class libraries.
- ◆ Lines 6–20: Code for declaration of the class RegServlet and initialization of variables. All the fields in the registration form are given variable names, and some temporary variables are defined here.
- ◆ Lines 21–29: This code is to connect to the database using JDBC-ODBC connectivity. The code is kept in a try-catch block, just to ensure that if something goes wrong, the exception is caught.
- ◆ Lines 30–46: As in the previous example (stock quotes), each customer is given a unique ID, which is generated automatically. This code checks for the present customer ID, increments by one, and assigns an ID to the new user.
- ◆ Lines 48–54: This code gets the information of the customer (name, cell number, address fields, city, and mail ID).
- ◆ Lines 57–82: This code gets the preferences selected by the user. It discovers whether the check box has been checked and assigned to a temporary variable temp. This is a repetitive code for each of the items in the registration form.
- ◆ Lines 84–103: This code generates an HTML page with the message “Thank You For Registering” and gives the customer ID (or security ID because we’re dealing with online shopping). This response is given to the user on completion of successful registration.

Now comes the actual push application. We need to write the code that will check the database for each user’s registered preferences and push the content to the user based on the periodicity indicated in the registration form.

Listing 6-6 gives the Java program for validating the user based on the customer ID and displaying the information on the browser of the handset.

Listing 6-6: Java Program for Validating and Displaying on the Browser

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.  import java.util.*;
5.  import java.sql.*;
6.  public class ShoppingCart extends HttpServlet
7.  {
8.      String[] cuno;
9.      Connection con;
10.     String url,mg="";
11.     String cname="";
12.     String ccellno="";
13.     String cadd1="",cadd2="",cadd3="",ccity="";
14.     String cmail="";
15.     String custidstr="";
16.     String cnew="";
17.     String csend="",temp="";
18.     String s1,s2,s3,s4,s5,ino1,max1,s6,s7,s8,ii,s9;
19.     ResultSet rs,rs1,rs2,rs3,rs4,rs5,rs6,rs7,rs8,rs9,rs10,rs11;
20.     Statement stmt,stmt1,stmt2,stmt3,stmt4,stmt5,stmt6,stmt8;
21.     int c1=0,c11,c11,ino=0,ino2,custidint=0;
22.     public void init(ServletConfig sc) throws ServletException{
23.         /* CONNECTING TO THE DATABASE */
24.         try{
25.             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
26.             con = DriverManager.getConnection("jdbc:odbc:shop");
27.             stmt = con.createStatement();
28.             stmt1= con.createStatement();
29.         }catch(Exception e){System.out.println(e);}
30.     }
31.     public void doGet(HttpServletRequest request,HttpServletResponse response)
32.         throws
33.         ServletException,IOException
34.     {
35.         /* DESIGN A FORM FOR DISPLAYING DATA */
36.         PrintWriter out = response.getWriter();
37.         /* Get Identity of the Customer and Parse it */
38.         custidstr = request.getParameter("CustId");
39.         custidint = Integer.parseInt(custidstr);
40.         /* Set Content-type Header */
41.         response.setContentType("text/vnd.wap.wml");
42.         /* Write the Response */
43.         out.println("<?xml version=\\"1.0\\"?>");
44.         out.println("<!DOCTYPE wml PUBLIC \\"-//WAPFORUM//DTD WML 1.1//EN\\"
45.             \\"http://www.wapforum.org/DTD/wml_1.1.xml\\">");
46.         out.println("<wml>");
47.         out.println("<template>");
48.         out.println("<do type='prev' label='Back'>");
49.         out.println("<prev/>");
50.         out.println("</do>");
51.         out.println("</template>");
52.     }

```

```

52.  /* Information on the Customer From the Database      */
53.  rs=stmt1.executeQuery("select CustomerID from CustomerDetails here
54.  CustomerID="+custidint);
55.  while(rs.next())
56.  {
57.    ino =rs.getInt(1);
58.  }
59.  rs.close();
60.  if(custidint == ino)
61.  {
62.    try{
63.      /* Based on the CustId Retrieve the Details from Database      */
64.      rs1 = stmt.executeQuery("select
65.      Groceries,Acessories,Cosmetics,FashionWear,NewProduct from CustomerDetails
        where
66.      CustomerId="+custidint);
67.      if(rs1.next())
68.      {
69.        s1 = rs1.getString(1);
70.        s2 = rs1.getString(2);
71.        s3 = rs1.getString(3);
72.        s4 = rs1.getString(4);
73.        s9 = rs1.getString(5);
74.        System.out.println("S1 :"+s1+" s2 :"+s2+" S3 :"+s3+" S4:"+s4+"S9 :"+s9);
75.      }
76.      rs1.close();
77.    }
78.    catch(Exception e1){System.out.println(e1.toString());}
79.    /* Print out Shopping Cart info for the User to Order      */
80.    out.println("<card id='card1' title='ShoppingCart'>");
81.    out.println("<p>");
82.    //if starts
83.    if(s1.equals("Yes"))
84.    {
85.      System.out.println("S1 ");
86.      /* Anchor for Navigation to Groceries Card      */
87.      out.println("<a href='#groc'>Groceries</a><br/>");
88.    }
89.    if(s2.equals("Yes"))
90.    {
91.      System.out.println("S2");
92.      /* Anchor for Navigation to Accessories Card      */
93.      out.println("<a href='#aces'>Accessories</a><br/>");
94.    }
95.    if(s3.equals("Yes"))
96.    {
97.      System.out.println("S3");
98.      out.println("<a href='#cosm'>Cosmetics</a><br/>");
99.    }
100.   if(s4.equals("Yes"))
101.   {
102.     System.out.println("S4");
103.     out.println("<a href='#fash'>FashionWear</a><br/>");
104.   }
105.   if(s9.equals("Yes"))
106.   {

```

```
107. System.out.println("S5");
108. out.println("<a href='#new'>New Products</a><br/>");
109. }
110. out.println("<anchor title='Link'>Order");
111. /* Link for Navigation to Order Servlet */
112. out.println("<go href='http://localhost:8080/Shopping/servlet/Order'
    method='get'
113. >");
114. /* Posting Parameters to Order Servlet */
115. out.println("<postfield name='CustId' value='"+custidint+"' />");
116. /* Check If User has selected the Item in the Registration Form */
117. if(s1.equals("Yes"))
118. {
119. try
120. {
121. rs6 = stmt.executeQuery("select * from Groceries");
122. while(rs6.next())
123. {
124. /*Retrieving First Column From the Database */
125. s8 = rs6.getString(1);
126. /* Set the Parameter Name and Value */
127. out.println("<postfield name='"+s8+"' value='"+s8+"' />");
128. }
129. rs6.close();
130. }
131. catch(Exception e11){System.out.println(e11.toString());}
132. }
133. if(s2.equals("Yes"))
134. {
135. try
136. {
137. rs7 = stmt.executeQuery("select * from Acessories");
138. while(rs7.next())
139. {
140. s8 = rs7.getString(1);
141. out.println("<postfield name='"+s8+"' value='"+s8+"' />");
142. }
143. rs7.close();
144. }
145. catch(Exception e12){System.out.println(e12.toString());}
146. }
147. if(s3.equals("Yes"))
148. {
149. try
150. {
151. rs8 = stmt.executeQuery("select * from Cosmetics");
152. while(rs8.next())
153. {
154. s8 = rs8.getString(1);
155. out.println("<postfield name='"+s8+"' value='"+s8+"' />");
156. }
157. rs8.close();
158. }
159. catch(Exception e13){System.out.println(e13.toString());}
160. }
161. if(s4.equals("Yes"))
```

```

162. {
163. try
164. {
165. rs9 = stmt.executeQuery("select * from FashionWear");
166. while(rs9.next())
167. {
168. s8 = rs9.getString(1);
169. out.println("<postfield name='"+s8+"' value='"+s8+"' />");
170. }
171. rs9.close();
172. }
173. catch(Exception e14){System.out.println(e14.toString());}
174. }
175. if(s9.equals("Yes"))
176. {
177. try
178. {
179. rs11 = stmt.executeQuery("select * from newproduct");
180. while(rs11.next())
181. {
182. s8 = rs11.getString(2);
183. out.println("<postfield name='"+s8+"' value='"+s8+"' />");
184. }
185. rs11.close();
186. }
187. catch(Exception e14){System.out.println(e14.toString());}
188. }
189. out.println("</go></anchor>");
190. out.println("</p>");
191. out.println("</card>");
192. /* Write the Response to the Groceries Card if User Registers for
    item */
193. if(s1.equals("Yes"))
194. {
195. out.println("<card id='groc' title='Groceries'>");
196. out.println("<p>");
197. //out.println("Groceries Here");
198. try
199. {
200. rs2 = stmt.executeQuery("select * from Groceries");
201. out.println("Item Name Rate Qty");
202. while(rs2.next())
203. {
204. s7 = rs2.getString(1);
205. s5 = rs2.getString(2);
206. s6 = rs2.getString(3);
207. out.print(s5+" "+s6);
208. out.println("<input name='"+s7+"' type='text' value='0' />");
209. out.println("<br/>");
210. }
211. rs2.close();
212. }
213. catch(Exception e3){System.out.println(e3.toString());}
214. out.println("</p>");
215. out.println("</card>");
216. }

```

```
217. /* Write the Response to the Accessories Card if User Registers for
    item */
218. if(s2.equals("Yes"))
219. {
220. out.println("<card id='aces' title='Accessories'>");
221. out.println("<p>");
222. //out.println("Accessories Here");
223. try
224. {
225. rs3 = stmt.executeQuery("select * from Accessories");
226. out.println("Item Name Rate Qty");
227. while(rs3.next())
228. {
229. s7 = rs3.getString(1);
230. s5 = rs3.getString(2);
231. s6 = rs3.getString(3);
232. out.print(s5+" "+s6);
233. out.println("<input name='"+s7+"' type='text' value='0' />");
234. out.println("<br/>");
235. }
236. rs3.close();
237. }
238. catch(Exception e4){System.out.println(e4.toString());}
239. out.println("</p>");
240. out.println("</card>");
241. }
242. /* Write the Response to the Cosmetics Card if User Registers for
    item */
243. if(s3.equals("Yes"))
244. {
245. out.println("<card id='cosm' title='Cosmetics'>");
246. out.println("<p>");
247. //out.println("Cosmetics Here");
248. try
249. {
250. rs4 = stmt.executeQuery("select * from Cosmetics");
251. out.println("Item Name Rate Qty");
252. while(rs4.next())
253. {
254. s7 = rs4.getString(1);
255. s5 = rs4.getString(2);
256. s6 = rs4.getString(3);
257. out.print(s5+" "+s6);
258. out.println("<input name='"+s7+"' type='text' value='0' />");
259. out.println("<br/>");
260. }
261. rs4.close();
262. }
263. catch(Exception e5){System.out.println(e5.toString());}
264. out.println("</p>");
265. out.println("</card>");
266. }
267. /* Write the Response to the FashionWear Card if User Registers
    for item */ if(s4.equals("Yes"))
268. {
269. out.println("<card id='fash' title='FashionWear'>");
```

```

270. out.println("<p>");
271. //out.println("FashionWear Here");
272. try
273. {
274. rs5 = stmt.executeQuery("select * from FashionWear");
275. out.println("Item Name Rate Qty");
276. while(rs5.next())
277. {
278. s7 = rs5.getString(1);
279. s5 = rs5.getString(2);
280. s6 = rs5.getString(3);
281. out.print(s5+" "+s6);
282. out.println("<input name='"+s7+"' type='text' value='0' />");
283. out.println("<br/>");
284. }
285. rs5.close();
286. }
287. catch(Exception e6){System.out.println(e6.toString());}
288. out.println("</p>");
289. out.println("</card>");
290. }
291. /* Write the Response to the New Products Card if User Registers for
    item */
292. if(s9.equals("Yes"))
293. {
294. out.println("<card id='new' title='New Products'>");
295. out.println("<p>");
296. //out.println("FashionWear Here");
297. try
298. {
299. rs10 = stmt.executeQuery("select * from newproduct order by category");
300. //out.println("Item Name Rate Qty");
301. String temp1 = "",temp="";
302. boolean flag=false;
303. int i=0;
304. while(rs10.next())
305. {
306. {
307. temp = rs10.getString(1);
308. if(flag)
309. {
310. if(!temp.equals(temp1))
311. {
312. i=0;
313. }
314. }
315. if(i == 0)
316. {
317. temp1=temp;
318. out.println(temp);
319. out.println("<br/>");
320. out.println("Item Name Rate Qty");
321. out.println("<br/>");
322. i++;
323. }
324. s7 = rs10.getString(2);
325. s5 = rs10.getString(3);

```



```

326. s6 = rs10.getString(4);
327. out.print(s5+" "+s6);
328. out.println("<input name='"+s7+"' type='text' value='0' />");
329. out.println("<br/>");
330. flag = true;
331. }
332. rs10.close();
333. }
334. catch(Exception e6){System.out.println(e6.toString());}
335. out.println("</p>");
336. out.println("</card>");
337. }
338. out.println("</wml>");
339. }
340. //if ends
341. else
342. {
343. out.println("<card id='card1' title='Sorry'>");
344. out.println("<p>");
345. out.println("Your ID Is Not Valid.");
346. out.println("</p>");
347. out.println("</card>");
348. out.println("</wml>");
349. }
350. }catch(Exception e1){System.out.println(e1.toString());}
351. }
352. public void doPost(HttpServletRequest request,
353. HttpServletResponse response) throws ServletException,IOException
354. { doGet(request,response); }
355. }

```

Code Description

- ◆ Lines 1–5: As usual, we import the necessary class libraries.
- ◆ Lines 6–21: This code is for declaration the class and initialization of variables. All the fields in the registration form are given variable names, and also some temporary variables are declared.
- ◆ Lines 22–30: This code connects to the database using JDBC-ODBC connectivity. For exception handling, the code is kept in a try-catch block.
- ◆ Lines 31–40: A WML card is created in this code to get the customer ID from the handset. At line 40, the content type is set to WML.
- ◆ Lines 42–50: The customer ID obtained from the user is verified in the database. If the customer ID is valid, only the following code is executed. Otherwise, a message saying that the ID is not valid is sent to the handset as in the code in Lines 321–329.
- ◆ Lines 51–78: Based on customer ID, this code retrieves the details (items selected) from the database.
- ◆ Lines 80–104: Based on the items selected, a WML card containing the shopping cart will be generated. Line 71 generates the card title. Lines 74–79 generated the code for groceries only if groceries are selected by the user. Lines 80–85 are for accessories; Lines 86–90 are for cosmetics; Lines 91–95 are for fashion wear; Lines 96–100 are for new products.
- ◆ Lines 105–115: If the user selects groceries when the code given in Lines 80-104 is executed in the handset, the shopping cart has to display the item name, rate, and quantity fields that can be input by the user. When the user inputs the data, it has to be stored in the database through another servlet

called Order. The items selected in the shopping cart selected by the user will be posted in the database using this code.

- ◆ Lines 117–132: Here, the process of displaying the item name, rate, and quantity fields that can be input by the user is performed for groceries.
- ◆ Lines 133–146: The above process (displaying the item name, rate and quantity fields) is repeated for accessories.
- ◆ Lines 147–160: The above process is repeated for cosmetics.
- ◆ Lines 161–174: This code corresponds to the above process for fashionwear.
- ◆ Lines 175–191: This code corresponds to the same process for new products.
- ◆ Lines 193–216: If user selects groceries, the WML card is generated along with fields for item name, rate, and quantity.
- ◆ Lines 218–241: If the user selects accessories in the shopping cart, this card is generated with fields for item name, rate, and quantity.
- ◆ Lines 243–266: The process of card generation is repeated for cosmetics.
- ◆ Lines 268–290: The card generation process is repeated for fashionwear.
- ◆ Lines 292–339: The card generation process is repeated for new products.
- ◆ Lines 341–351: If the customer ID is not valid, the message “Sorry, Your ID Is Not Valid” is generated using this code.
- ◆ Lines 352–355: All the inputs will be posted in the database, and the recordset will be closed.

The above program obtains the information from the user — the information is the items of interest and the quantity the user would like to procure. This has to be posted in the database for later processing of the order. Listing 6-7 gives the Java program for saving the customer order in the database.

Listing 6-7: Java Program for Saving the Customer Order in the Database

//© 2001 Dreamtech Software India Inc.

//All Rights Reserved

```

1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4. import java.util.*;
5. import java.sql.*;
6. public class Order extends HttpServlet
7. {
8.     String orderstr,custidstr;
9.     Connection con;
10.    ResultSet rs,rs1,rs2;
11.    Statement stmt,stmt1;
12.    public void init(ServletConfig sc) throws ServletException
13.    {
14.        //Connecting To the Data Base
15.        try{
16.            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
17.            con = DriverManager.getConnection("jdbc:odbc:shop");
18.            stmt = con.createStatement();
19.            stmt1 = con.createStatement();
20.        }
21.        catch(Exception e){System.out.println(e.toString());}
22.    }
23.    public void doGet(HttpServletRequest request, HttpServletResponse response)

```

```

24. throws      ServletException, IOException
25. {
26. /*          Order Information is Being Saved in the DataBase      */
27. orderstr = request.getQueryString();
28. custidstr = request.getParameter("CustId");
29. System.out.println(" Order Information :"+orderstr);
30. PrintWriter out = response.getWriter();
31. /*          Set Content-type Header      */
32. response.setContentType("text/vnd.wap.wml");
33. out.println("<?xml version=\"1.0\"?>");
34. out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\" \"
35. \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
36. /*Write the response to the User      */
37. out.println("<wml>");
38. out.println("<template>");
39. out.println("<do type='prev' label='Back'>");
40. out.println("<prev/>");
41. out.println("</do>");
42. out.println("</template>");
43. try{
44. /*   Insert Details of the User into the Database      */
45. boolean b = stmt1.execute("insert into OrderInfo (CustomerId,
    OrderItemsAndQty)
46. values ('"+custidstr+"','"+orderstr+"')");
47. /*   Print Info of Order being Saved      */
48. out.println("<card id='card1' title='OrderSaved'>");
49. out.println("<p>");
50. out.println("Your Order Is Saved<br/>");
51. out.println("</p>");
52. out.println("</card>");
53. out.println("</wml>");
54. }catch(Exception e){
55. System.out.println(e.toString());
56. out.println("<card id='card1' title='Error'>");
57. out.println("<p>");
58. out.println(e.toString());
59. out.println("</p>");
60. out.println("</card>");
61. out.println("</wml>");
62. }
63. }
64. }

```

Code Description

- ◆ Lines 1–5: To import the necessary class libraries.
- ◆ Lines 6–11: This code is for declaration of the class and initialization of the variables.
- ◆ Lines 12–13: Function declaration.
- ◆ Lines 15–22: This code is to connect to the database using JDBC-ODBC connectivity.
- ◆ Lines 23–42: The order placed by the user is saved in the database through the Servlet. Lines 32–37 generate the necessary WML template by using the `wml`, `template`, and `do` tags.
- ◆ Lines 43–46: The order is saved with details of customer ID, ordered items, and quantity for each item.

- ◆ Lines 48–53: A WML card is generated, which is used to confirm to the user that his order is saved with the message: “Your Order Is Saved.”
- ◆ Lines 54–64: This code is to take care of any exceptions; a WML card is generated containing the exception (Error) message.

The shopping cart and order placement is done through a servlet. Now we need a small WML code, which calls this shopping cart. Listing 6-8 provides the WML code for calling the shopping cart servlet and placing the order. This is a simple program with just one card with the title Virtual Mall, having a welcome message that says, “Hello customer, enter your security ID to order items.” When the user enters the ID, the ID must be obtained by the card, the servlet `ShoppingCart` has to be invoked, and the customer ID has to be posted on the server.

Listing 6-8: WML Code for Calling the Shopping Cart Servlet and Placing the Order

//© 2001 Dreamtech Software India Inc.
//All Rights Reserved

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
3. <wml>
4. <card id="card1" title="Virtual Mall">
5. <p>
6.
   Hello Customer
7. <br/>
   Enter Your Security ID To Purchase Order
8. <br/>
9. <input name="Cid" type="text"/>
   Then Select Order Option
10. <!-- Link to Shopping Cart Servlet When you Click Order -->
11. <anchor title="Orders">Order
12. <go href="http://localhost:8080/Shopping/servlet/ShopingCart" method="get">
13. <postfield name="CustId" value="${Cid}"/>
14. </go>
15. </anchor>
16. </p>
17. </card>
18. </wml>

```

Code Description

- ◆ Lines 1–2: The header for the WML card giving the Document Type Definition.
- ◆ Lines 3–4: The familiar WML and card tags with the card titled Virtual Mall.
- ◆ Lines 5–8: To welcome the customer with a Hello Customer message.
- ◆ Line 9: To prompt the customer to enter the security number.
- ◆ Line 11: To obtain the customer ID through an input tag.
- ◆ Lines 12–14: To invoke the servlet by using the anchor tag.
- ◆ Line 15: To take the customer ID and assign it to a variable `Cid`. `Cid` is a variable that contains the customer ID.
- ◆ Lines 16–18: Closing tags for `p`, `card`, and `wml`.

The procedure for executing this program is exactly same as that of the preceding example. The WML code shown in Listing 6-8 executes first and appears on the handset. The entry screen for the virtual mall is shown in Figure 6-12.



Figure 6-12: Virtual mall entry screen for user to input the security ID

The user has to enter the security ID (or the customer ID) and select an order. This invokes the servlet for the shopping cart; the shopping cart appears on the handset in Figure 6-13. The user can select a category, which causes the display to appear as shown in Figure 6-14.

The Item name and Price are displayed, and a field for entering quantity is available for the user. The user can input the quantity and go back to the previous menu and select the Order option. The order is saved in the database and the user sees the Order Saved screen, as shown in Figure 6-15.

This example shows the power of mobile commerce and mobile advertisements. The operators and content providers can use the WAP push technology to create simple but useful applications for m-commerce. You can enhance the application shown here to take care of payment — it can be done through a credit card or by transferring money from a bank account.



Figure 6-13: Shopping cart displayed on the WAP Phone



Figure 6-14: Selection of Item and Quantity



Figure 6-15: Order saving confirmation

Pros and Cons of Push Framework

Push framework has been defined in WAP version 1.2. To periodically obtain content from the Internet without making a request every time is a very attractive option to users. Think of a situation where you walk into a music store with your handset. Imagine that a server in the store can display the latest titles and specials on your handset. This sort of thing is a possibility with the help of WAP technology.

Although the Internet supports push in a limited way (for example, pushing the newsletters that you subscribe to), we still need to download items by connecting to the Internet. On the other hand, the push framework is more attractive for the mobile devices because they are always on, and the advertisements can be pushed without the user explicitly getting connected to the Net.

Push technology is still in a nascent stage; the commercial WAP servers that support push applications are just being released. The impact of push technology on users and whether users will really utilize the technology is yet to be seen. But proponents of push technology say that it is the market pull rather than technology push that is driving the WAP push framework.

Summary

In this chapter, we studied push technology in the WAP environment. The WAP push services can be provided through the SMS bearer, which puts a limitation of 160 characters for each message.

Alternatively, by using the special protocols defined for the push services (Push Access Protocol and Push Over The Air Protocol), innovative services can be provided to the user. The Push Proxy Gateway bridges the wireless network domain and the Internet domain to push the content from the Push Initiator to the handset. Push Access Protocol, which uses the HTTP protocol, is used for communication between the Push Proxy Gateway and the Push Initiator. Push Over The Air Protocol, which runs above the Wireless Session Protocol, is used for communication between the Push Proxy Gateway and the handset. Development of push applications has been discussed in detail, and code for two applications — pushing stock quotes and mobile cart integrated with m-advertising — have been presented.

Push technology in WAP holds great promise both for the user and the operator. The user is freed from the burden of making requests every time she wants to get the information — whether it's stock quotes, cricket scores, betting odds, or astrological predictions for the day. The operator has a great potential for revenue, as m-advertising promises to be very lucrative.

Chapter 7

Bluetooth: A Basic Introduction

Due to the widespread use of computers and other electronic gadgets, every office and home is now a labyrinth of wires that connect computers, peripherals, and appliances. This situation can create a lot of maintenance problems. Interconnecting these devices without wires through radio offers tremendous advantages in terms of less maintenance, more reliability, and better appearance. Bluetooth provides a solution by interconnecting devices through low-cost, reliable radio. Over the next few years, every office and every home will have Bluetooth-enabled devices. Many manufacturers have come out with Bluetooth hardware and software, but the current cost of making a Bluetooth-enabled device is high. In the coming years, however, a \$5 Bluetooth solution is expected, and every electronic device can become Bluetooth-enabled. In this chapter, we study the Bluetooth technology with an emphasis on protocols that enable you to develop Bluetooth applications.

Introduction to Personal Area Networks (PANs)

A typical office is equipped with a number of electronic gadgets, such as a PC, laptop, printer, fax machine, modem, and so on. These devices are interconnected through wires for the purpose of using a service (a print service) or for sharing information (transferring a file from a desktop to a laptop) — they form a Personal Area Network (PAN). A PAN is an ad hoc network because the topology and the number of nodes at any time are not fixed; they change dynamically. All the headaches associated with administering such networks can be avoided if these devices are made to communicate through radio links and if one device can discover the presence and capabilities of other devices. The need for PANs is everywhere — in offices, at homes, and in cars.

A number of technologies have been proposed for PANs, notably Bluetooth, HomeRF, IrDA, and IEEE 802.11. Of these technology standards, Bluetooth is considered the most attractive choice.

Overview of Bluetooth

Bluetooth technology is a low-cost, low-power, short-range, radio technology open standard for development of personal area networks (PANs). Bluetooth will replace cable connections because it's as cheap, if not cheaper, than the cable. Many Bluetooth-enabled devices operate through a battery; consequently, the power consumption to make the device communicate over radio is very low — hence, the radio should emanate low power. Because Bluetooth helps in creating a small network of devices that are close to one another, the range is very short, typically about 10 meters. It uses radio as the transmission medium to avoid wire connections. It is based on open standards — standards framed by an industry consortium. Bluetooth devices made by different manufacturers can, therefore, interoperate, thereby paving the way for competition and lowered costs. Because of all these features, Bluetooth is likely to be one of the most popular technologies for wireless personal area networking.

Bluetooth Special Interest Group (SIG) was founded in February 1998 by Ericsson, Intel, IBM, Toshiba, and Nokia. Bluetooth specification version 1.0 was released in July 1999 and version 1.1 in February 2001. A device such as a PC, digital camera, headset, or cordless phone can become Bluetooth-enabled by means of an attached module that contains the hardware and software for running the Bluetooth protocols. A Bluetooth-enabled device can exchange information or transfer data with another Bluetooth-

enabled device over a radio. It's estimated that by 2002, nearly 100 million mobile phones will be Bluetooth-enabled. Bluetooth-enabled devices will be common in homes, offices, and cars.

Any electronic device that needs to communicate with another electronic device can be Bluetooth-enabled. Such devices include

- ◆ Desktop PCs
- ◆ Laptops and Personal Digital Assistants
- ◆ Keyboard and mouse
- ◆ Mobile phones and two-way pagers
- ◆ Cordless phones
- ◆ Fax machines and scanners
- ◆ Overhead and LCD projectors
- ◆ Headsets and loud speakers
- ◆ Televisions and music systems
- ◆ LAN access points
- ◆ Domestic appliances such as microwave ovens, washing machines, and refrigerators
- ◆ Set top boxes and Web TVs
- ◆ Point of sale terminals and ATMs

Because Bluetooth is a nascent technology, the cost of making a device Bluetooth-enabled is currently quite high. As the technology matures and demand picks up, more and more devices will be Bluetooth-enabled.

Bluetooth in the Office

As shown in Figure 7-1, all the devices in an office can form an ad hoc network whenever they come together. If the user wants to send a fax, the laptop will discover the presence of the fax machine, establish a connection, and send the fax. If the fax machine isn't working, the laptop will know that fax service is not available. The user can bring his digital camera and download the picture on to the desktop and create a printout without connecting any cable. The user can synchronize the appointments stored in the laptop and mobile phone and ensure that both contain the same set of appointments for the next day. The headset doesn't need to be connected to the mobile phone through a wire to answer a call received on the phone. This is a typical Bluetooth office scenario.

Bluetooth in the Home

Consider the PAN in the home shown in Figure 7-2. Just as with the PAN in the office, various devices can communicate with one another by using Bluetooth radio. Consider the following example, which illustrates the potential of this technology: When you go home, tired after a hard day's work, you can put dinner in the microwave oven and relax in your living room while watching the TV. When dinner is done, the microwave oven will flash a message on the TV indicating that it's time to eat. (Despite all the conveniences that Bluetooth affords, you still need to eat dinner with your own hands!) Perhaps this Bluetooth scenario has the potential of making you a couch potato, but this is just one example of the advantages of wireless networking in the home.

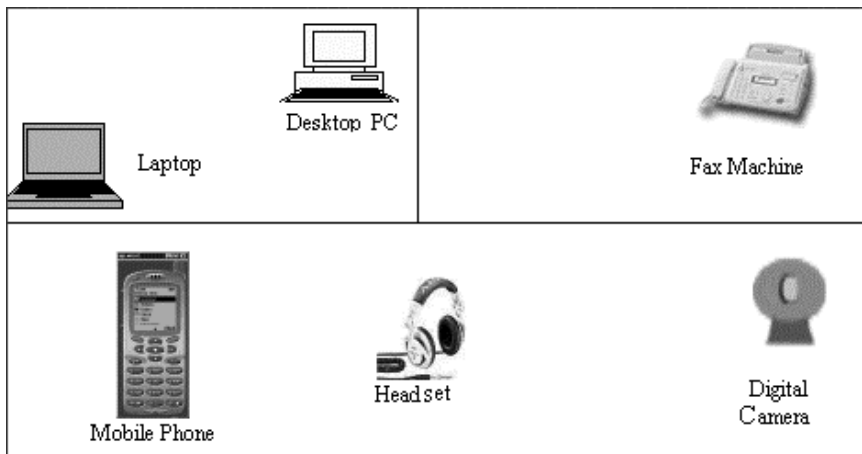


Figure 7-1: PAN in an office

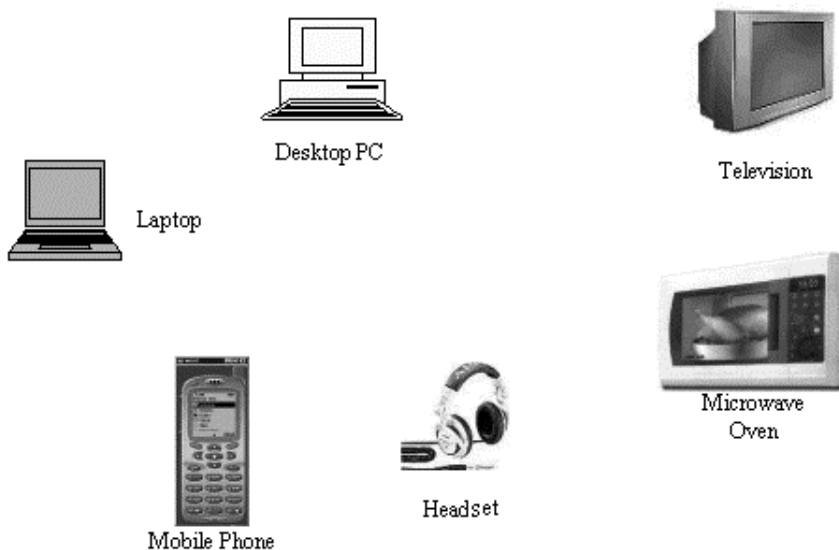


Figure 7-2: PAN at home

Bluetooth in the Car

The various gadgets in a car, such as a sound system, laptop, mobile phone, and headset, can be made to communicate with each other through Bluetooth. Someone sitting in the back seat can compose e-mail on the laptop and send the mail through the mobile phone without physically connecting the two. A person can hear the music emanating from the music system through the headphones, again without a wired connection.

As shown in Figure 7-3, the devices in the car will form a PAN or a personal operating space. If the car has a navigation system with a GPS (Global Positioning System) receiver, it can send the instructions to the driver on the headset without wires.

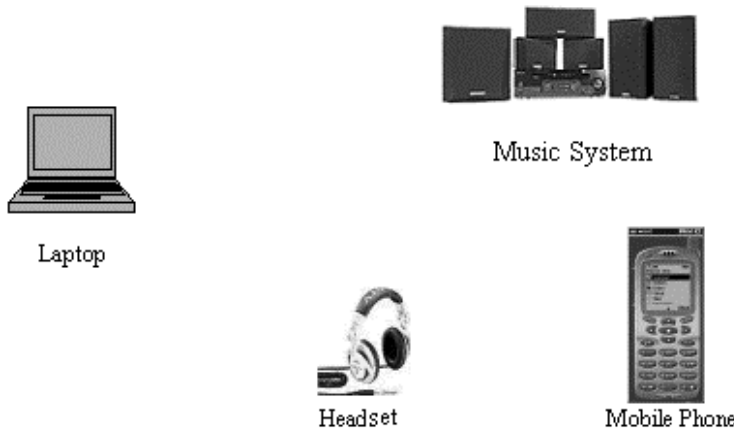


Figure 7-3: PAN in a car

Now that you've seen some of the typical applications of the Bluetooth technology, you're going to take a look at the broad specifications of a Bluetooth system.

Bluetooth System Specifications

The specifications of the Bluetooth system are discussed in the following sections.

Operating Frequency

The Bluetooth operating frequency is 2400 – 2483.5 MHz; every Bluetooth device transmits in this frequency band. The band consists of 79 channels, each of 1-MHz bandwidth. Bluetooth radio uses frequency hopping — the frequency of transmission changes for every packet. In normal radio communication systems, the transmitting device sends the data using only one frequency, which the receiving device is tuned to. In frequency hopping, there is a hop sequence where the transmitting device changes the frequency, and the receiving device has to automatically tune to the changed frequency to receive the data. This certainly creates complexity in the design of the radio, but the advantage of frequency hopping is that the communication link is secure — unless the receiver knows the hopping sequence, it cannot receive the data. So, the Bluetooth radio transmits the first packet by using one frequency, the second packet in another frequency, and so on, using the 79 frequencies that are available in the band.

Operating Range

The normal operating range of Bluetooth is 10 meters (the devices that form the network should be within a radius of 10 meters). The range is dependent on the power of the radio transmitter — the higher the power, the higher the range. In Bluetooth specifications, three classes of devices are defined. Class 1 devices transmit a maximum of 100 mW, and they can have a range of 100 meters. Class 2 devices transmit 10 mW and the range is 50 meters. Class 3 devices transmit 1mW and have a range of 10 meters. Most of the commercially available devices have a transmitting power of 1 mW and a range of 10 meters.

Services Supported

Bluetooth supports both voice and data services. Because voice communication is done in circuit-switching mode and data communication in packet-switching mode, both types of connections are supported in Bluetooth. The link established between devices for voice communication is an Synchronous Connection Oriented (SCO) link, and the link established for data communication is an Asynchronous Connection Less (ACL) link.

Data Rates

The Bluetooth device can support one asynchronous channel and up to three synchronous voice channels. For voice communication, 64 Kbps data rate is supported in both directions. For asynchronous links, two types of channels are possible. In an asymmetric channel, the data rates are different in the two directions — 723.2 Kbps in one direction and 57.6 Kbps in the other direction. In a symmetric channel, 433.9 Kbps data rate is supported in both directions.

Network Topology

In a PAN, a set of devices forms a piconet (a small network). In each piconet, there is one *master*, which all other devices (called *slaves*) tune to. The master decides the hop-frequency sequence, and the slaves synchronize with the master to establish links. Any device can become a master — the master/slave terminology is only to define the protocols. A cellular phone, for example, can be a master or a slave to a desktop. In a piconet, a maximum of seven slaves can actively communicate with the master. If two devices in a piconet communicate with each other, with one acting as master and one as slave, the communication is called point-to-point communication. If more than two devices communicate with one another, with one acting as master and others as slaves, the communication is point-to-multipoint.

These are the broad specifications of the Bluetooth system. You study the detailed architecture of a Bluetooth system later in the chapter.

Bluetooth versus Other Technologies

You can choose from a number of technologies to develop a small network. These include IrDA standards formulated by InfraRed Data Association, HomeRF formulated by HomeRF group, and IEEE 802.11 Wireless Local Area Network (LAN) standards formulated by Institute of Electrical and Electronics Engineers (IEEE).

Bluetooth versus IrDA

The Infrared Data Association (<http://www.irda.org>) was founded in 1993 to develop standards for low-cost solutions for point-to-point infrared (IR) communication. The IR communication can be implemented in PDAs, cameras, printers, overhead projectors, bank ATMs, fax machines, and so on, for communication with other devices within a range of 1 meter. Serial IR (SIR) supports data rates up to 115 Kbps and Fast IR (FIR) supports data rates up to 4 Mbps. The directed IR systems allow one-to-one communication in point-to-point mode and are not subject to regulations. IrDA-based products, which include laptops, keyboards, and mobile phones, are now commercially available.

The main disadvantage of IrDA is that infrared rays cannot penetrate walls. Bluetooth is better than IrDA in this regard, because radio frequencies can do so. IrDA systems support only point-to-point communication. Another drawback of IrDA is that it supports only data services, though data rates are higher.

Bluetooth versus HomeRF

HomeRF is another radio technology for home-networking of computers and peripherals and is Bluetooth's strongest competitor. Users having high-tech homes with multiple PCs and peripherals would like having a low-cost solution to network the PCs and peripherals for applications such as the following:

- ◆ To transfer a file from a laptop to a desktop without connecting the two with wires.
- ◆ To allow multiple PCs to share the same modem and telephone line for high-speed Internet connectivity.
- ◆ To activate home appliances through cordless telephones.

- ◆ To download music from the Internet on to the PC and listen to the music through a headset, but without connecting the PC to the headset through wires.
- ◆ To forward calls to different cordless handsets.

Home networks are now gaining popularity to support such services. Though wired networks based on standards (such as IEEE 1394) are available, wireless home networks are certainly more attractive because of fast installation, no need for cables, and easier maintenance. The two standards that have been proposed for wireless home networks are 802.11b (covered in the next section) and SWAP by HomeRF.

Shared Wireless Access Protocol (SWAP), sponsored by HomeRF working group www.homerf.org/, provides low-cost solutions to home-networking needs. The broad specifications of this standard are:

- ◆ **Range:** Up to 150 feet (covering a home, backyard, and garage, if the home is not a mansion)
- ◆ **Number of devices:** Up to 127 per network
- ◆ **Frequency band:** 2.4 GHz (ISM band)
- ◆ **Transmit power:** 100 mW
- ◆ **Speed:** 10 Mbps peak data rate with fall back mode to 5 Mbps, 1.6 Mbps, 0.8 Mbps (data rates are likely to be increased up to 20 Mbps in the future versions of the standard)
- ◆ **Access:** Frequency-Hopping Spread Spectrum with 50 hops per second

Each network is given a 48-bit ID, so concurrent operation of multiple coallocated networks is possible. The network ID, frequency hopping, and the 128-bit data encryption provide the necessary security for the network and the data.

HomeRF certainly meets the requirements of home networking. Its main attractions are higher data rate support than Bluetooth and support for a higher number of devices, as compared to the number of active devices in the piconet of Bluetooth. Both HomeRF and Bluetooth operate in the same frequency band and use frequency-hopping technique. HomeRF is a definite competitor to Bluetooth in home networking space.

Bluetooth versus 802.11 Wireless LAN

IEEE developed the wireless LAN standards through the 802.11 committee. The wireless LANs based on the IEEE 802.11 standard have become popular and have a large installation base. The 802.11b standard is sponsored by the Wireless Ethernet Compatibility Association (WECA) for the wireless networking of home devices. This standard supports 10 to 100 access points with a data rate of 11 Mbps. It uses Direct Sequence Spread Spectrum (DSSS) for medium access.

The current networks based on this standard are costlier than HomeRF and Bluetooth. Installing multiple networks in multi-storied apartments causes interference — unless the radio base stations are installed at carefully chosen places. Another drawback of the systems based on this standard is that its support for voice over IP (Internet Protocol) is limited, whereas HomeRF can provide better voice over IP support.

A number of technologies are available to meet the needs of personal and home-networking by using radio. Each technology meets the needs of a specific segment of the user community, and all these technologies may coexist. Ultimately, it's cost that decides the large deployment of the technologies.

Commercial Bluetooth Solutions

Because of the great business potential in Bluetooth technology, many equipment manufacturers are introducing a variety of Bluetooth-enabled devices. To ensure that the devices supplied by different vendors can work together (interoperability), Bluetooth SIG released the Bluetooth profiles, which ensure that all vendors interpret and implement the same set of features in their devices. These profiles are

defined for a variety of devices, such as headphones, cordless phones, and LAN access points. The following are some of the commercial Bluetooth solutions available:

- ◆ **Bluetooth headset:** The headset provides a wireless connectivity to the cellular phone, desktop phone, PDA, or desktop/notebook PC. These headsets are compliant with headset profile version 1.1. These devices should have low power consumption and be lightweight.
- ◆ **Bluetooth cordless phone:** This consists of a Bluetooth cordless handset and a base station. The Bluetooth-enabled base station acts as a gateway to the telephone network so that the handset can be used to make telephone calls through a normal telephone network. These phone solutions are compliant with Bluetooth cordless telephony/intercom profile version 1.1. A Bluetooth-enabled desktop/notebook PC can receive dial-up connectivity through the base station. Two handsets can communicate with each other over the Bluetooth link.
- ◆ **Bluetooth LAN access point:** The LAN access point integrates Bluetooth-enabled computing and communication devices into a wireless network. It can also provide connectivity between existing LANs and Bluetooth-enabled wireless networks. This solution is compliant to LAN access profile version 1.1.
- ◆ **Bluetooth LCD projector:** An LCD projector can obtain the data from a notebook PC over the Bluetooth radio link, thereby allowing for presentations without wires.
- ◆ **Bluetooth desktop/notebook PCs:** A desktop PC can be Bluetooth-enabled by connecting the Bluetooth module to the USB port and running the software module in the desktop. Alternatively, an add-on card can be plugged into the PC, and the associated software can be loaded in the system. Other designs include a compact flash card with an adapter module that can be plugged into the notebook PC or other handheld devices and a PCMCIA card with Bluetooth hardware integrated into it.
- ◆ **Bluetooth speakers:** Home stereo systems can be connected to speakers over the radio link by making the speakers Bluetooth-enabled.

To ensure that different vendors implement the same set of features so that devices are compatible, Bluetooth SIG released the Bluetooth profiles. The following profiles are presently defined in version 1.1:

- ◆ **Generic access profile:** Generic procedures to discover Bluetooth devices and establishing connections with other devices and security aspects.
- ◆ **Service discovery application profile:** Procedures for a device to discover the services available on other devices.
- ◆ **Cordless telephony profile:** To make use of a cellular phone to access a fixed telephone network.
- ◆ **Intercom profile:** For cordless phones.
- ◆ **Serial communication profile:** For two devices (two laptops) to communicate with each other through serial communication.
- ◆ **Headset profile:** For devices such as a mobile phone and a desktop PC to connect to a headset.
- ◆ **Dial-up networking profile:** To access the cellular network from a PC through a mobile phone, or to access a fixed network from a PC through a line modem.
- ◆ **Fax profile:** To send a fax message from a PC through a cellular phone.
- ◆ **LAN access profile:** To access a LAN from a Bluetooth-enabled device (such as PC).
- ◆ **Object push profile:** To push an object (such as an electronic business card) from a cellular phone to another cellular phone.
- ◆ **File transfer profile:** To transfer files from one device to another.
- ◆ **Synchronization profile:** To synchronize objects (such as appointments) in different systems such as a notebook PC and cellular phone.

Note that these profiles specify the features that have to be implemented in various devices to make them Bluetooth-enabled. The profiles help the developers in ensuring that the same sets of features are implemented.

Network of Bluetooth Devices: Piconet and Scatternet

The network formed by a set of Bluetooth devices is called a piconet. In a piconet, there's a device called a master, and a number of other devices called slaves.

A piconet can support two types of communication, as shown in Figure 7-4. In the part of the figure marked a), a piconet has one master and one slave. In this case, only these two devices are involved in communication, so it's a point-to-point communication. In b), a piconet has one master and a number of slaves. In this case, the master can communicate with all the slaves, so it's a point-to-multipoint communication. In a point-to-multipoint configuration, several Bluetooth devices share a channel. Formally defined, a piconet is a small network with two or more devices sharing the same channel.

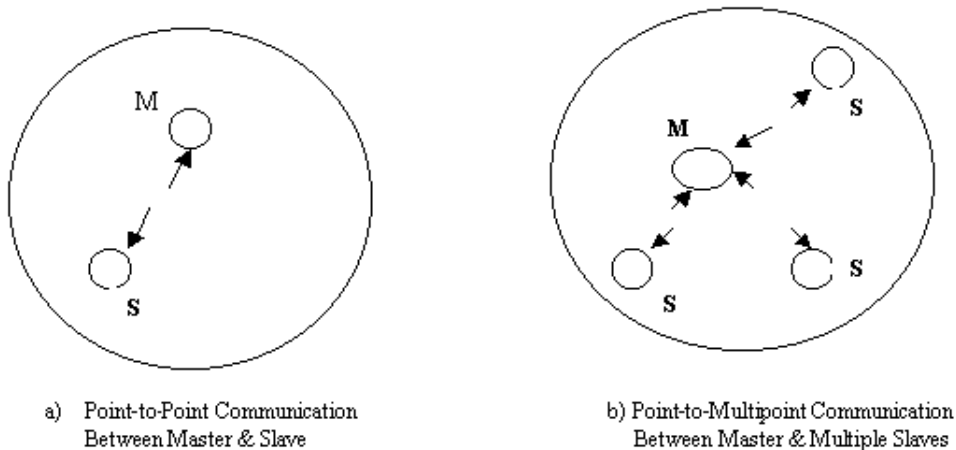


Figure 7-4: Bluetooth piconet

To communicate, the master's and slave's frequency and time must be synchronized. The slave should know in which frequency the master is transmitting and tune to that frequency. The slave should also know the time of the packet transmission. In a piconet, up to seven slaves can be active, and many more can be in the parked state. The slaves in the parked mode are locked to the master: They cannot be active on the channel, but they are synchronized to the master. The master controls the channel access. A master and slave can switch roles — the slave can become a master and the master can become a slave.

Multiple piconets with overlapping coverage form a scatternet, as shown in Figure 7-5. Each piconet will have a master, but a master of one piconet can be a slave in another piconet. Each piconet has its own frequency-hopping sequence, so that there's no interference between two piconets.

Data and Voice Support

Bluetooth supports data services such as file transfer, wherein two devices can exchange data by using packet transmission. The data is divided into packets and sent over the radio channel. The receiving device acknowledges the packet or reports that the packet is received in error. If a packet is received with errors, the packet is retransmitted. It's also possible to broadcast packets to all the slaves. In broadcast mode, however, there's no acknowledgement or indication that the packet is received with errors. The master must then indicate to the slaves how many times a broadcast packet will be transmitted so that every slave will receive the packet without errors at least once.

In the case of voice, the first requirement is to convert the voice (which is an analog signal) into digital format. Two types of voice coding techniques are used: Pulse Code Modulation (PCM), which is a coding technique to convert analog voice signal into 64-Kbps format based on G.711 standards, and Continuously Variable Slope Delta Modulation (CVSD) technique, which also converts the voice into 64-Kbps data rate. Voice packets are not retransmitted.

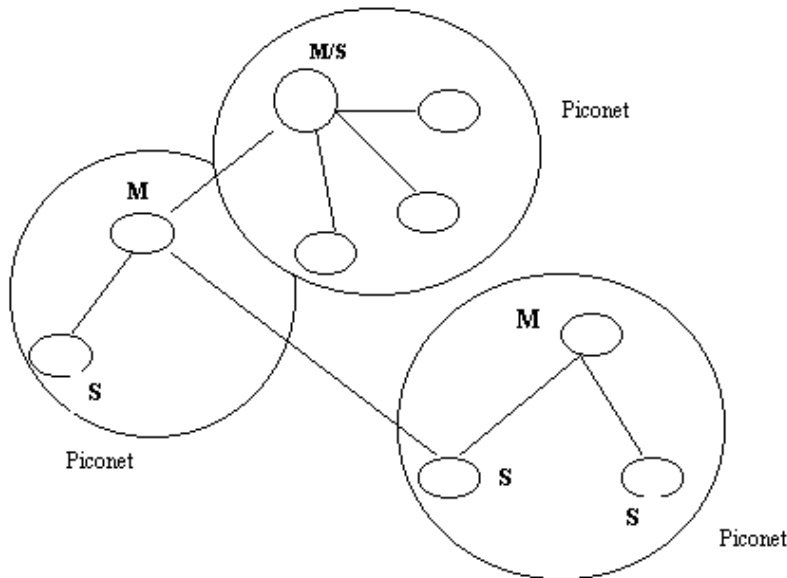


Figure 7-5: Bluetooth scatternet

Security Issues in Bluetooth

A radio system is generally considered to be prone to attacks because anyone can have a radio receiver tuned to the transmitter frequency and receive the data. Bluetooth is a highly secure system for the following reasons:

- ◆ Every Bluetooth device is given a 48-bit address that uniquely identifies the device. Every Bluetooth device on earth will have a unique address.
- ◆ When one device wants to communicate with another device, the second device is authenticated.
- ◆ Data on the channel is encrypted so that only the intended recipients can receive it.
- ◆ Every Bluetooth device has a random number generator; these numbers are used for authentication.
- ◆ A frequency-hopping scheme provides built-in security — only those devices that know the hopping sequence can decode the data sent by the master.

Architecture of a Bluetooth System

Figure 7-6 shows a Bluetooth module consisting of a Radio, a Link Controller, and a Link Manager. The module consisting of these three blocks is interfaced to the host, which can be a laptop, a mobile device, and so on. The following sections describe each of these modules.

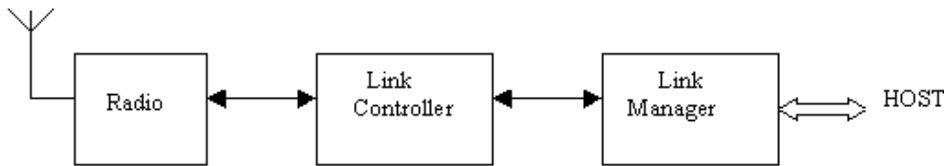


Figure 7-6: Functional block diagram of a Bluetooth module

Radio Hardware

Bluetooth radio operates in the 2.4 GHz ISM (Industrial, Scientific and Medical) band. The frequency spectrum allocated is in the range 2000 to 2483.5 MHz. There are 79 RF channels with a channel spacing of 1 MHz, a lower-guard band of 2 MHz, and an upper-guard band of 3.5 MHz. The guard bands ensure that there will be no interference with radio equipment operating in the adjacent frequency bands. The radio operates in frequency-hopping mode — each packet is transmitted in a different frequency. The master decides the hop sequence, and each slave synchronizes with the master in a piconet. Each piconet has a different frequency-hop sequence, and thus security is built-in. Nominal frequency-hop rate is 1600 hops per second.

Gaussian Frequency Shift Keying (GFSK) is used as the modulation technique. A positive frequency deviation represents 1 and a negative frequency deviation represents 0. The radio receive must be designed so that the Bit Error Rate (BER) of minimum 0.1% is ensured. In other words, the radio should provide a link that ensures that there won't be more than one error for every 1000 bits sent.

Three power classes are defined based on the power radiated by the Bluetooth radio. Table 7-1 shows the various power classes and the radiated power.

Table 7-1: Power Classes

<i>Power Class</i>	<i>Maximum Output Power</i>	<i>Minimum Output Power</i>
1	100 mW	1 mW
2	2.5 mW	0.25 mW
3	1 mW	--

Based on the power radiated, the range of the Bluetooth device can be 100 meters, 50 meters, and 10 meters, respectively. The minimum distance between two Bluetooth devices should be 10 centimeters.

Link Controller

A link controller carries out baseband protocols and other low-level link routines. Information is exchanged between Bluetooth devices in the form of packets, and each packet is transmitted in a different frequency. Each packet is normally transmitted in a slot of 625 microseconds, though some packets can extend up to five slots. To achieve full-duplex communication between the master and slaves, a Time Division Duplex (TDD) scheme is used. Normally, in the radio communication systems, two frequencies are used — one frequency in each direction. This is known as Frequency Division Duplex (FDD). In TDD, only one frequency is used for communication in both directions. During one time slot, one device will send the data, which is received by the other device; in the next time slot, the other device will send the data.

Time slot

Each time slot has a duration of 625 microseconds. These slots are numbered from 0 to $(2^7)-1$. The master starts the transmission in even slots by sending a packet addressed to a slave; the slave sends the packets in odd numbered slots. A packet generally occupies one time slot, but can extend to up to five slots. If a packet extends more than one slot, the hop frequency will be the same for the entire packet.

The single packet transmission is shown in Figure 7-7. In this figure, the master transmits in slot 0 to the slave using frequency f_1 , the slave transmits to the master in slot 1 using frequency f_2 , the master transmits in slot 2 using frequency f_3 , and so on.

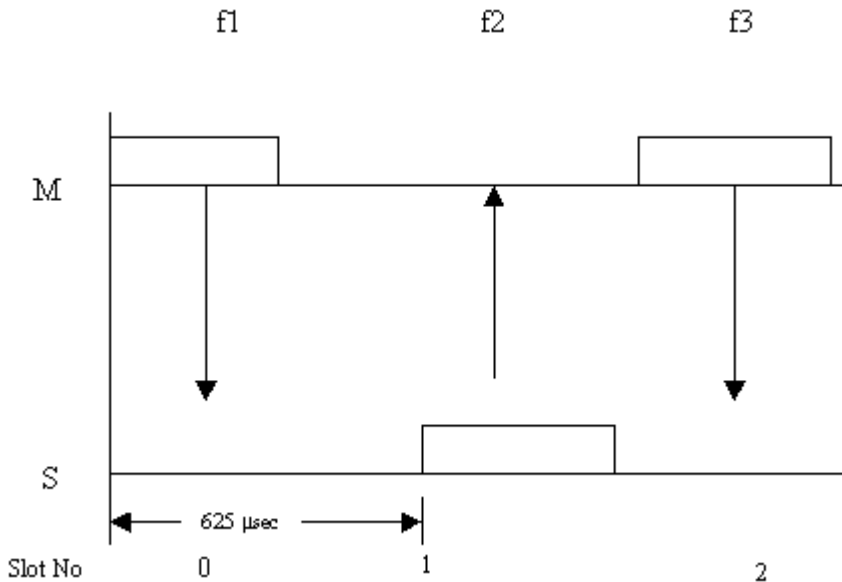


Figure 7-7: Communication between master and slave

States of a Bluetooth device

A Bluetooth device can be in one of the two major states: connection state and standby state. In connection state, the device is communicating with another device by exchanging packets. In standby state, the device is not communicating with another device and will be in low-power mode to save battery power. This is the default state. There can be seven substates: page, page scan, inquiry, inquiry scan, master response, slave response, and inquiry response.

To start, an application program in a Bluetooth device can enter the inquiry state to find out about other devices in the vicinity. To respond to an inquiry, the devices should periodically enter into inquiry scan state. When the inquiry is successfully completed, the device enters the inquiry response state. When a device wants to get connected to another device, it enters the page state to page for another device. In this state, the device would become the master and page for other devices. The command for this paging has to come from an application program running on this Bluetooth device. When the device pages for the device, the other device may respond, and the master enters the master response state. Devices should enter the page scan state periodically to check whether other devices are paging for it. After the device receives the page-scan packet, it enters the slave response state.

After paging of devices is complete, the master and slave establish a connection in active state, during which the packet transmission takes place. The connection can also be put in one of the three modes: hold, sniff, or park. In hold mode, the device will stop receiving the data traffic for a specific amount of time so that other devices in the piconet can use the bandwidth. After the expiration of the specific time, the device will start listening to traffic again. In sniff mode, a slave will be given an instruction such as “listen starting with slot number S every T slots for a period of N slots.” The device doesn’t need to listen to all the packets, but only as specified through the preceding parameters, which are the sniff parameters. The connection can be in park mode, wherein the device only listens to a beacon signal from the master occasionally; it synchronizes with the master but has no data transmission.

A typical procedure for setting up a Bluetooth link is as follows:

1. The device sends an inquiry using a special inquiry-hopping sequence.
2. Inquiry scanning devices respond to the inquiry by sending a packet. This packet contains the information needed to connect to it.
3. The inquiring device requests a connection to the device that responded to the inquiry.
4. Paging is used to initiate the connection with the selected device.
5. The selected device, which has entered the page scan state, responds to the page.
6. If the responding device accesses the connection, it synchronizes with the master's timing and frequency hopping sequence.

Voice and data connections

Because Bluetooth supports both voice and data, it uses a combination of circuit switching and packet switching. For voice communication, Synchronous Connection Oriented (SCO) channel is used, and for data communication Asynchronous Connection Less (ACL) channel is used. Note that SCO links are mainly for applications that are time-critical, such as voice—packet delays are unacceptable (they can cause breaks in the voice). Bluetooth can support an asynchronous channel and up to three synchronous voice channels. Voice channels support 64 Kbps in each direction. After voice connection is established, the voice packets need to be exchanged between the master and the slave continuously. Slots then can be reserved for synchronous packets. An asynchronous channel can support a maximum of 723.2 Kbps asymmetric (and upto 57.6 Kbps in the reverse link) or 433.9 Kbps symmetric data rate.

SCO links are like circuit-switched connections for voice communication. SCO links are for point-to-point communication between master and slave. A master can support up to three SCO links, and a slave can support three SCO links from one master or two SCO links if the links are from different masters in a scatternet environment. SCO packets are never retransmitted.

ACL is a packet-switched connection between the master and all the slaves for data communication. ACL links provide point-to-multipoint communication between the master and all the slaves that participate in the piconet environment. Only one ACL link can exist between master and the slaves. Packet retransmission is done in ACL links in case of packet loss. If a master sends a packet addressed to a slave, the slave can send its packet to the master in the next slot. Packets not addressed to any slave are broadcast packets, which are received by all the slaves.

The packet format is shown in Figure 7-8. The packet consists of an access code (68 or 72 bits), a header (54 bits), and a payload (0 to 2745 bits). Packets can contain only an access code (a shortened access code with 68 bits only), an access code and header, or an access code, header, and payload.

68 or 72 bits	54 bits	0-2745 bits
Access Code	Header	Payload

Figure 7-8: Bluetooth packet format

An access code is used for synchronization and identification of devices in a piconet. All packets in a piconet will have the same access code. An access code is used for paging and inquiry procedures; in such cases, no header or payload is required because only signaling information is carried.

There are three types of access codes:

- ◆ Channel Access Code (CAC): Identifies a piconet; all packets in a piconet contain this code.
- ◆ Device Access Code (DAC): This code is used for paging and response to paging.

- ◆ Inquiry Access Code (IAC): General IAC is used to discover which Bluetooth devices are in range. Dedicated IAC is common for devices with a common characteristic. Only those devices can be discovered.

A packet header is 54 bits long and breaks down as follows: 3 bits for active member address (all zeros for broadcast)

- ◆ 3 bits for active member addresses.
- ◆ 4 bits for type code (SCO link, or ACL link, how many slots packet will occupy), 1 bit for flow control (if buffer is full, 0 for stop and 1 to go), 1 bit for acknowledgement indication (1 indicates that packet is okay, 0 indicates packet in error).
- ◆ 1 bit for sequence number (for each packet, this bit is reversed).
- ◆ 8 bits for Header Error Control for error checking.

These bits total 18 bits; rate 1/3 FEC is used to make it 54 bits by repeating each bit three times. In 1/3 FEC, each bit is repeated three times to help in error correction at the receiving end if there are transmission errors.

Note that three bits are allocated for active member address, which limits the number of addresses in a piconet to eight. Out of these, one address (all zeros) is for broadcasting the packets in a piconet. You're then left with seven addresses, so only seven active devices can be in a piconet.

This field contains the user information, which can be either data or voice.

Bluetooth addressing

Each Bluetooth module (the radio transceiver) is given a 48-bit address containing three fields: LAP (Lower Address Part) with 24 bits, Upper Address Part (UAP) with 8 bits, and Non-Significant Address Part with 16 bits.

The addressing format is shown in Figure 7-9. This address is assigned by the Bluetooth module manufacturer and consists of company ID and company-assigned number. This address is unique to each Bluetooth device. In literature dealing with Bluetooth, this address is referred to as BT_ADDR.

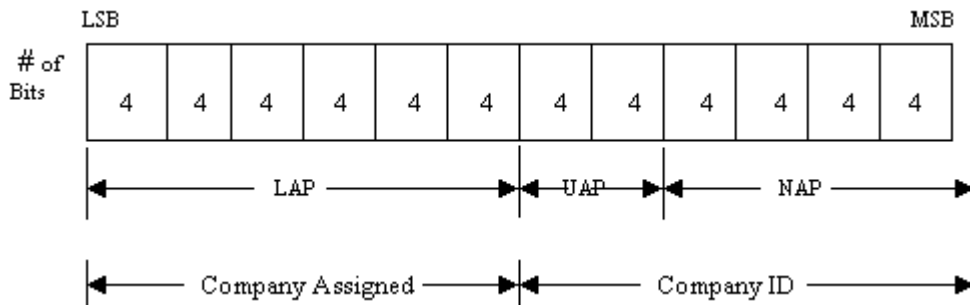


Figure 7-9: Format of Bluetooth address

Each active member in a piconet will have a 3-bit address. In addition to the maximum of seven active members, many more devices can be in parked mode. The parked members also need to have addresses so that, if required, the master can make them active for exchange of packets. Parked member address is either the BT_ADDR of 48 bits or an 8 bit parked member address denoted by PM_ADDR.

To summarize, the link controller does a lot of work — to establish the link based on the type of service required (voice or data), to take care of addressing, and to take care of the device's different states.

Link Manager Protocol

The Link Manager Protocol (LMP) is used to set up and control links. The three layers, RF, a Link Controller, and a Link Manager, will be on the Bluetooth module attached to the device.

This layered architecture is shown in Figure 7-10. The Link Manager on one device exchanges messages with the Link Manager on the other device as indicated in the figure. These messages, known as LMP messages, are not propagated to higher layers. Link messages have higher priority to data. LMP messages are sent as single-slot packets with a header of one byte. The various functions of the LMP are as follows:

- ◆ **Authentication:** Of the two devices that need to communicate, one is a verifier and the other is a claimant. The verifier sends a message, a packet containing a random number, which is called a challenge. The claimant calculates the response, which is a function of challenge, and sends the response along with its Bluetooth address (48-bit address) and secret key. This is known as a challenge response scheme — you throw a challenge and check whether the other device can correctly respond to that challenge.
- ◆ **Encryption:** The master sends a key to all the slaves with which the data is encrypted through an LMP message.
- ◆ **Clock offset request:** To synchronize the clocks between the master and slave is a must for proper data exchange. If the clock has to be offset, the LMP exchanges messages to ensure clock synchronization.
- ◆ **Timing accuracy information request:** To ensure synchronization, the master can ask the slaves for timing accuracy information.
- ◆ **LMP version:** A version of the LMP protocol is exchanged to ensure that both devices use the same set of protocols and understand each other's messages.
- ◆ **Request and response features of LMP:** Request and response packets are used to obtain the LMP features supported by the Bluetooth devices.
- ◆ **Switching master/slave role:** The master and slave in a piconet can switch roles by using the LMP messages. The switching operation can be initiated by the master or slave.
- ◆ **Name request:** Each device can be given a user-friendly name having a maximum of 248 bits in ASCII format. A device can request the name through an LMP message and obtain the response.
- ◆ **Detach:** Messages exchanged to close a connection.
- ◆ **Hold mode:** This is used to place an ACL link in hold for a specified time when there is no data to send, mainly to save power.
- ◆ **Park mode:** This is used to be in synchronization with the master but not participate in data exchange.
- ◆ **Power control:** This requests the transmission of less power. This is useful particularly for Class 1 devices, which are capable of transmitting 100 mW power.
- ◆ **Quality of Service (QoS) parameters exchange:** In applications that require a good quality transmission link, quality of service parameters can be specified. These parameters include the number of repetitions for broadcast packets, delay, and bandwidth allocation.
- ◆ **Request SCO link:** This is used after ACL link is established, to make a request SCO link.
- ◆ **Multi-slot packet control:** When data has to be sent in more than one slot, control information is sent using the LMP messages.
- ◆ **Link supervision:** This is used to monitor a link when the device goes out of range (through a time-out mechanism)
- ◆ **Connection establishment:** When a device responds to the paging message, connection establishment is done through the LMP messages.

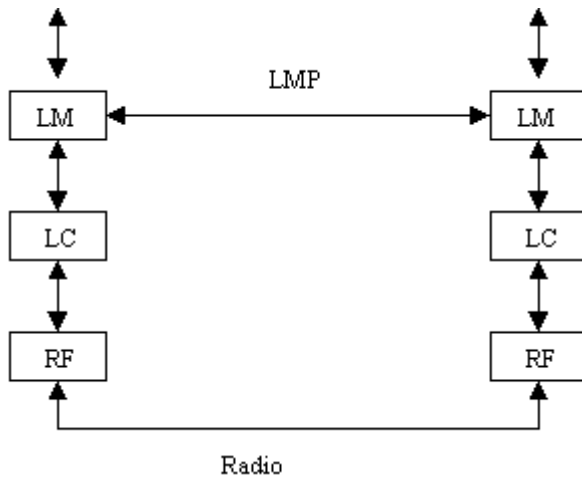


Figure 7-10: Layered architecture for Bluetooth module

As mentioned earlier, the Bluetooth device will implement these three layers in a hardware/firmware combination. These layers ensure establishment of a connection and management of the connection for transfer of voice or data. But to ensure that the whole application runs as per user requirements, you need other protocols. The complete Bluetooth protocol architecture is shown in Figure 7-11.

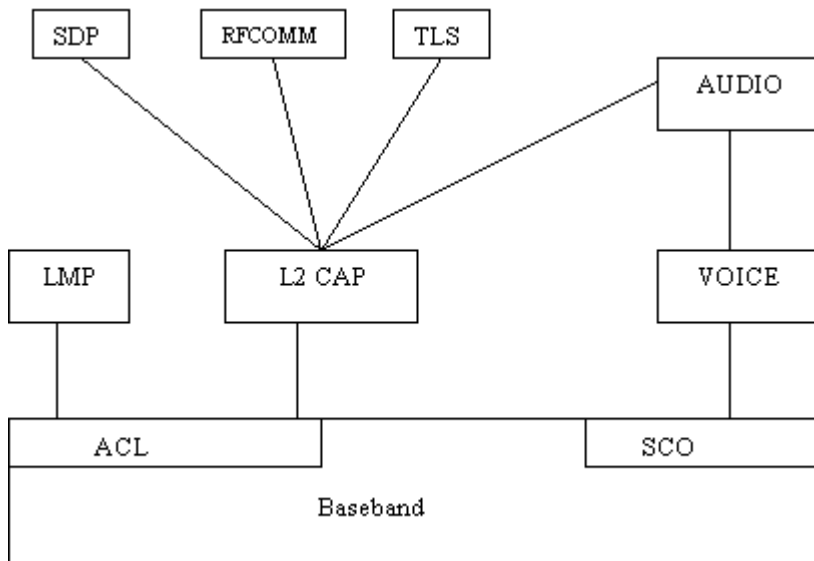


Figure 7-11: Bluetooth protocol architecture

Consider a scenario where two Bluetooth-enabled PCs want to enter a chat session. To start, there should be a chat application program running on each of these PCs. When one PC wants to initiate the chat session, the Bluetooth protocol stack has to discover the other Bluetooth PC, establish an ACL connection, and then transfer the data. Consider the case of establishing a voice connection between two devices. For voice connection, first the signaling information has to be transmitted and then the voice information. The signaling information is exchanged through ACL links and the voice through SCO links. Therefore, you need quite a few protocols to handle the voice and data applications. We study these protocols in detail in the following sections.

Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP runs above the baseband and carries out the data link layer functionality. The L2CAP layer is only for ACL links. L2CAP data packets can be up to 64 Kilobytes long. L2CAP protocol runs on hosts such as laptops, cellular phones, and other wireless devices.

When L2CAP messages are exchanged between two devices, it's assumed that an ACL link is already established between two devices. It's also assumed that ordered delivery of packets is ensured and L2CAP doesn't do any checksum calculation. Note that L2CAP does not support SCO links for voice communication and multicasting.

The functions of L2CAP layer are the following:

- ◆ **Protocol multiplexing:** In the protocol stack shown in Figure 7-11, above L2CAP, a number of other protocols can be running. A packet received by L2CAP indicates which protocol is to be identified so that it's passed to the correct higher layer. This is protocol multiplexing.
- ◆ **Segmentation and reassembly:** Baseband packets are limited in size, as you have seen in the packet format. Large L2CAP packets are segmented into small baseband packets and sent to the baseband. Similarly, the small packets received from the baseband are reassembled and sent to higher layers.
- ◆ **Quality of Service:** To ensure Quality of Service (QoS), constraints are honored.

L2CAP layer sends connection request and QoS request messages from the application programs through the higher layers. It receives the responses for these requests from the lower layers. The responses can be the following: connection indication, connection confirmation, connect confirmation negative, connect confirmation pending, disconnection indication (from remote), disconnect confirmation, time out indication, and quality of service violation indication.

Service Discovery Protocol

The Service Discovery Protocol (SDP) provides the Bluetooth environment with the capability to develop ad hoc networks. This protocol is used for location of services provided by or available through a Bluetooth device. The SDP facilitates the following:

- ◆ The client's ability to search for the services needed by it in the piconet.
- ◆ Services to be discovered based on class of services (such as a print service).
- ◆ The capability to browse services.
- ◆ The discovery of new services when devices enter RF proximity of other devices.
- ◆ The mechanism to find out when a service becomes unavailable because the device goes out of RF range (when there is no RF proximity).
- ◆ The details of services such as classes of services and the attributes of services.
- ◆ The capability to discover services on another device without consulting the third device.

The operation of SDP is depicted in Figure 7-12. When a device wants to discover a service, the application software running on the client machine initiates the request and the SDP client sends an SDP request to the server (the device that can provide the required service). The SDP client and server then exchange SDP messages. Note that the server and the client can be either of the two devices — the server is the device that can provide the service being requested by the client.

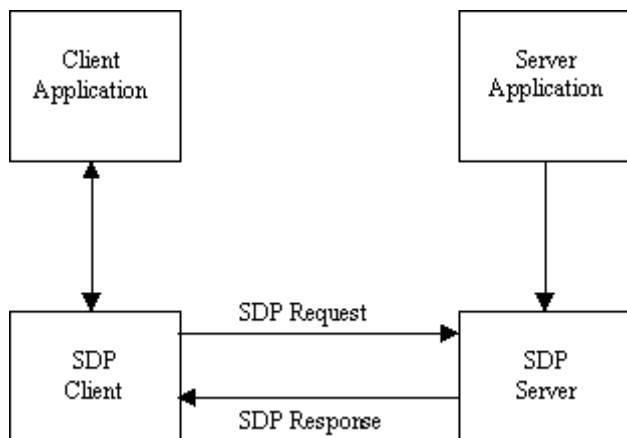


Figure 7-12: Service discovery protocol

The server maintains a list of service records. A 32-bit number identifies each record for unique identification. A service record has a number of attributes. The attributes can be a service class ID list (type of service), a service ID, a protocol description list (protocol used for using the service), a provider name, an icon URL (an iconic representation of the service), a service name, and a service description. Each attribute has two components — an attribute ID and an attribute value.

Suppose that a device, such as a laptop, requires a print service. The laptop is a client looking for a print service in a Bluetooth environment. The process would execute as follows:

1. Client sends a service search request specifying the print service class ID to the Server.
2. Server sends a service search response to the client indicating that two print services are provided.
3. Client sends a service attribute request, protocol descriptor list to the server, asking for the details of the service.
4. Server sends the response to the client indicating that PostScript print service is provided.

The SDP is the heart of the Bluetooth system. It provides the capability to discover availability and details of different services, along with other pertinent service information, such as protocols to access the service.

RFCOMM

RFCOMM is a transport protocol to emulate serial communication (RS232 serial ports) over L2CAP. Two devices can communicate through RFCOMM by using serial communication protocols over Bluetooth radio (see Figure 7-13). To achieve this, RFCOMM emulates the nine connections of RS 232. These signals are:

- ◆ 102 for signal common
- ◆ 103 Transmit Data (TD)
- ◆ 104 Received Data (RD)
- ◆ 105 Request to Send (RTS)
- ◆ 106 Clear to Send (CTS)
- ◆ 107 Data Set Ready (DSR)
- ◆ 108 Data Terminal Ready (DTR)
- ◆ 109 Data Carrier Detect (DTR)
- ◆ 125 Ring Indicator (RI)

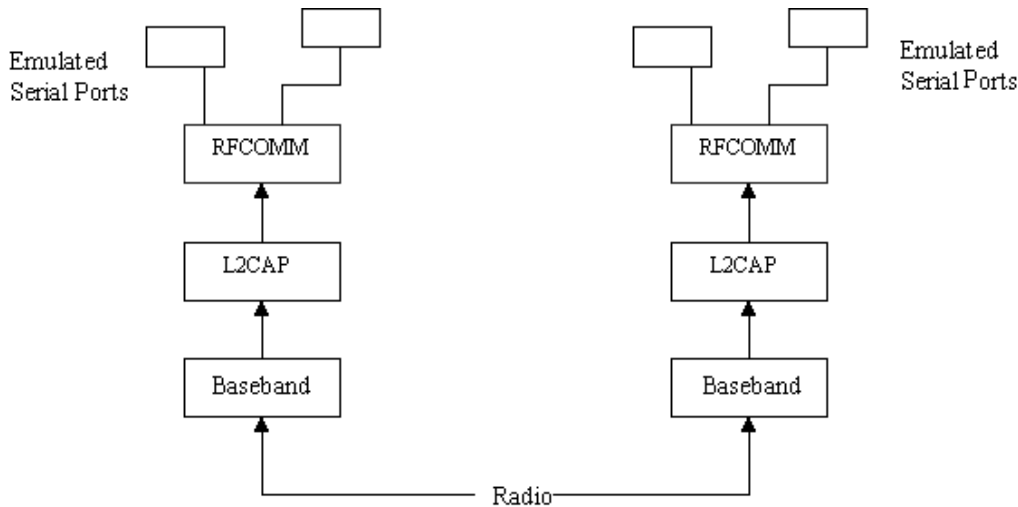
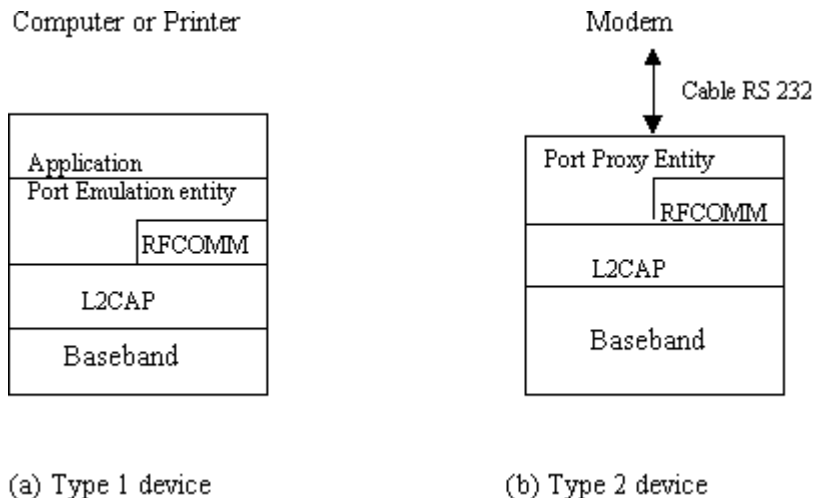


Figure 7-13: Serial port emulation through RFCOMM

RFCOMM is derived from GSM specification TS 07.10 for serial emulation. It supports two types of devices, as shown in Figure 7-14. Type 1 devices are communication end points such as computers and printers. Type 2 devices are part of a communication segment such as a modem.

Telephony Control Protocol Specification (TCS)

- ◆ To establish voice communication between two Bluetooth devices, you need SCO links. SCO links are not handled by the L2CAP protocol. But L2CAP handles the signaling required for establishing voice connections through Telephony Control Protocol Specification (TCS). Note that it's not abbreviated as TCP — this stands for Transmission Control Protocol used in the Internet protocol architecture. TCS defines call control signaling for establishing speech and data calls between Bluetooth devices and mobility management procedures.



(a) Type 1 device

(b) Type 2 device

Figure 7-14: Type 1 and Type 2 RFCOMM devices

This protocol is based on the International Telecommunications Union (ITU) standard Q.931, which is the standard for ISDN. TCS messages are exchanged between devices in the format specified in the protocol to carry out the following functions:

- ◆ Call control signaling to establish and release calls
- ◆ Signaling information not related to a call

Providing supplementary services such as calling line identification

Host Control Interface (HCI)

Suppose that you have a laptop computer that’s going to be Bluetooth-enabled. You can connect a small Bluetooth module to the USB port of the laptop and run the protocol stack on the laptop (called the host).

Figure 7-15 depicts the protocol stacks that need to run on the Bluetooth module and host. A Bluetooth device can have two parts: a module implementing the lower layers (LMP and below) and a software module implementing the higher layers stack (L2CAP and above), which is implemented in a host. The Host Controller Interface (HCI) provides a uniform interface so that the two modules can be from different vendors. HCI uses three types of packets:

- ◆ Commands, which are sent from the host to the module
- ◆ Events, which are sent from the module to the host
- ◆ Data packets, which are exchanged between the host and the module

The functions of HCI are:

- ◆ Setting up, disconnecting, and configuring the links
- ◆ Control of baseband features such as timeouts
- ◆ Retrieving the module’s status information
- ◆ Facilitating local testing of the module by issuing a set of commands from the host. Qualification testing of Bluetooth devices is also done using HCI commands.

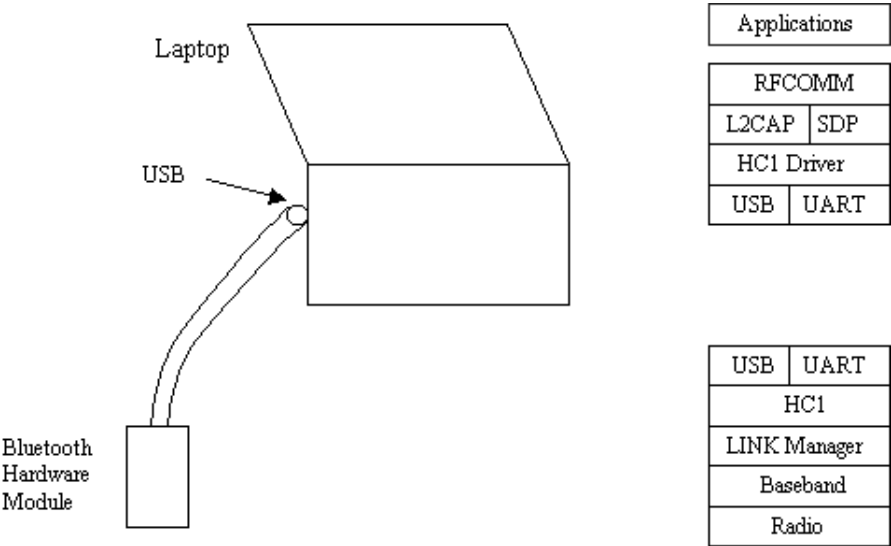


Figure 7-15: Bluetooth protocol stack on a laptop

HCI provides a command interface to the baseband controller and link manager, as well as access to hardware status and control registers. HCI has to reside in the Bluetooth module connected to the laptop as well as the host. In the Bluetooth module firmware, HCI commands are implemented so that the host can access baseband commands, link manager commands, hardware status registers, control registers, and event registers.

Figure 7-16 shows how the Bluetooth protocol stack is implemented. In this figure, the baseband controller is a hardware implementation. The functions of this module are baseband processing, physical layer protocols such as error correction and flow control, voice coding, and encryption. The hardware consists of a Central Processing Unit (CPU) and associated firmware in which the link manager and HCI are implemented. This firmware is responsible for link management to set up and manage links.

The Bluetooth module is connected to the USB port (say, of the laptop). The physical bus is the USB port. Three transport layers are defined to get HCI packets from host to the Bluetooth module: USB, RS 232, and UART (Universal Asynchronous Receive Transmit), a serial interface without error correction.

In the host, the bus driver is implemented as software, above which the HCI driver software and other higher layer protocol software are implemented. The HCI commands can be categorized as:

- ◆ Link control commands to establish piconets and scatternets
- ◆ Link policy commands to put devices in hold mode/sniff mode
- ◆ Commands to get information about the local hardware
- ◆ Commands to get the status parameters
- ◆ Commands to test the local Bluetooth module

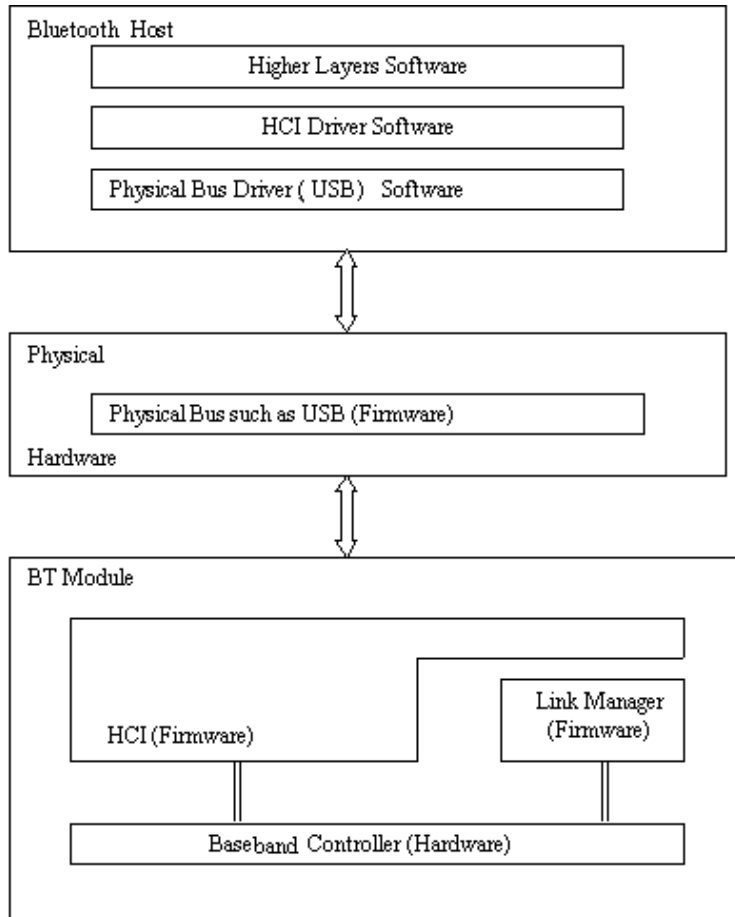


Figure 7-16: Hardware/firmware/software implementation of a Bluetooth system

Bluetooth APIs for Developing Applications

To facilitate development of Bluetooth applications, a number of vendors supply Bluetooth application development kits, which contain the Application Programming Interface (API) calls for different layers. One of these — the Bluetooth development system — is shown in Figure 7-17. The development kit consists of a Bluetooth module (a printed circuit board), which can be connected to the USB port of the desktop or laptop, and software that is run on the host computer.

The APIs are in the form of function calls, written in ANSI C language. For each of the layers — HCI layer, L2CAP layer, SDP, and RFCOMM — APIs are provided so that the developer can access any layer functionality to develop applications. Generally, a rich set of monitoring and debugging tools are also provided to make the development easy and fast. We will study the details of the development kit and show how to develop applications later in the book.

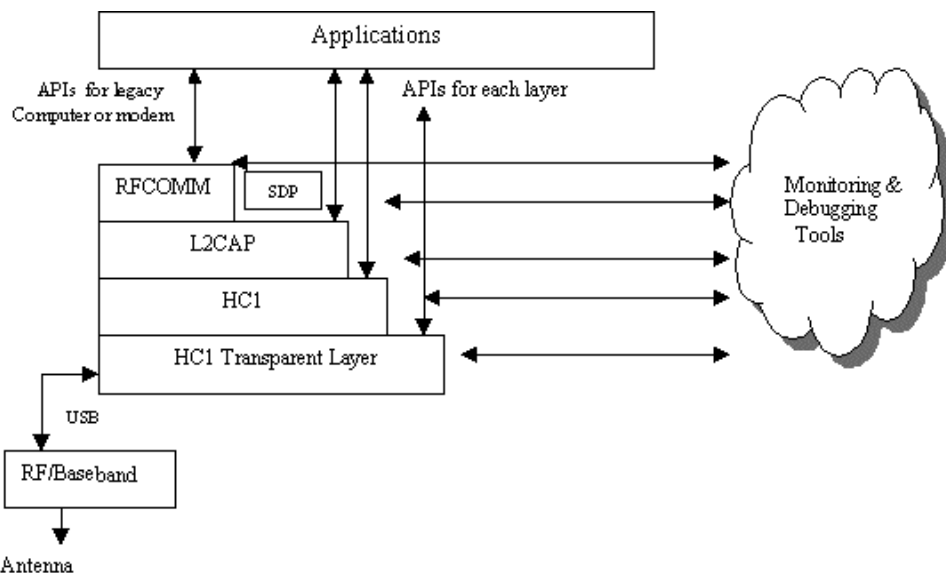


Figure 7-17: Bluetooth development system

Summary

This chapter laid the foundation for Bluetooth technology — the technology that enables devices to be interconnected through the radio transmission medium — which provides a low-cost, reliable, and secure communication between mobile and fixed devices. Bluetooth operates in the 2.4 GHz ISM band using frequency hopping. A number of devices sharing the same frequency channel form a piconet. Every piconet has a master and up to seven active slaves. The communication between the master and the slaves can be point-to-point communication or point-to-multipoint communication. Overlapping piconets form a scatternet. Voice and data services are supported; SCO links are established for voice communication, and ACL links for data communication. The Bluetooth protocol stack consists of baseband, Link Controller, Link Manager, L2CAP, RFCOMM, SDP, and TCS. The Host Controller Interface provides a uniform interface between a Bluetooth module and the host by defining the interface commands. This layered architecture of Bluetooth helps in interoperable Bluetooth devices.

Breathtaking developments are taking place in Bluetooth technology and Bluetooth will become ubiquitous in the near future. Though competing technologies such as HomeRF, IrDA, and IEEE 802.11 are available, Bluetooth is preferred because of its industry support, its capability to form ad hoc Personal Area Networks, and its support for voice and data services using low-cost, low-power radio technology based on open standards.

Chapter 8

Using WAP with Bluetooth

WAP enables mobile devices to access Internet content through a WAP server. Bluetooth enables mobile devices to communicate with other mobile/fixed devices over a short range. If devices are enabled to handle both WAP and Bluetooth protocols, we can develop interesting applications. This aspect is discussed in this chapter. We study the services provided to users by a mobile device that is both WAP-enabled and Bluetooth-enabled. The protocol stacks that must run on the devices to make them both WAP- and Bluetooth-enabled are described. We also discuss the implementation of typical applications using push technology.

Bluetooth as a WAP Bearer

The WAP protocol stack runs on a “bearer” — the bearer carries the data using the necessary protocols over the physical medium to establish and maintain links for communication. The WAP protocol stack has been developed in such a way that it can support different bearers — GSM-, TDMA-, or CDMA-based systems, and so forth. Bluetooth also can be one of the bearers for WAP. Bluetooth provides the physical medium and the link control for communication between a WAP server and a WAP client. So, if a WAP server (or a WAP proxy server) is Bluetooth-enabled and a WAP-enabled mobile phone is also Bluetooth-enabled, the mobile phone can obtain WAP content and present this content to the user. To achieve this, we need to define the necessary protocols. Before that, however, we need to see what kind of applications we can implement by using WAP with Bluetooth.

Application of WAP with Bluetooth

Here are some typical applications of WAP with Bluetooth. These applications are described in the Bluetooth specifications. In these examples, we assume that the devices are Bluetooth-enabled and also WAP-enabled. These applications demonstrate how location-dependent services can be provided to the users.

Briefcase trick

Assume that you are waiting at the airport with your laptop tucked into your briefcase. You would like to browse through your e-mail, which is in your mailbox on the laptop. You don’t have to open the briefcase and take out your laptop. If both your laptop and your mobile phone are Bluetooth-enabled, you can establish a communication between the phone and the laptop and browse through the mailbox using your mobile phone. This scenario is called the “hidden computing scenario.”

Forbidden message

Another case of hidden computing is the “forbidden message.” If you are traveling in an aircraft, you can compose e-mail messages on your laptop, but you can’t send them because you aren’t allowed to use mobile phones in the aircraft. After the plane lands, the laptop automatically checks for the mobile phone, and if the phone is on, a Bluetooth link is established between the mobile phone and the laptop. The e-mail messages are then sent over the wireless network through the mobile phone.

WAP Smart Kiosk

WAP- and Bluetooth-enabled kiosks (servers that provide information) might be located in airports to provide arrival and departure information to travelers. When a Bluetooth-enabled WAP phone comes into the vicinity of the kiosk, the mobile phone can request information from the kiosk, or alternatively, the kiosk can push the information to the mobile phone. For example, a kiosk at an airport can keep sending gate information to the mobile phones in its vicinity; a kiosk in a shopping mall can give information about the location of different shops or new product arrivals, and so forth. The kiosk can be a WAP server in which the WML content resides, or it can be just a WAP proxy server that obtains information from an Origin server on the Internet.

WAP in Bluetooth Piconet

Consider a piconet in which a Bluetooth-enabled WAP server is located, such as in Figure 8-1.

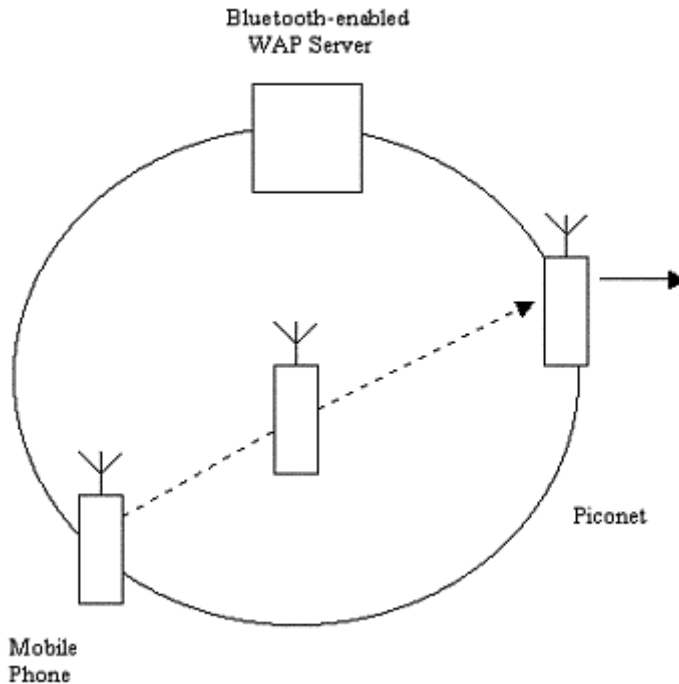


Figure 8-1: Piconet with a WAP server. Mobile phone entering and leaving the Piconet.

A mobile device can enter the piconet, connect to the WAP server, obtain content, and leave the piconet. This is a typical example of an ad-hoc network where the mobile devices get into and out of the piconet. For communication between the server and the mobile phone, there are two possibilities:

- ◆ Initiation by the client
- ◆ Initiation by the server

Initiation by the client

A Bluetooth-enabled mobile device can actively listen for the availability of other Bluetooth devices in the vicinity. When the device enters the RF range of the Bluetooth-enabled WAP server by using Service Discovery Protocol (SDP), the mobile device detects the presence of the WAP server. In other words, it discovers the WAP server. The SDP provides the mobile device the following information:

- ◆ Name of the server (a descriptive name, such as Airport Information Server).

- ◆ URL of the server's home page.
- ◆ Server capabilities (whether the server provides the content or it is just a proxy server that obtains information from some other server).

As the mobile device moves in the piconet, by using the information in the preceding list, the user can get the content from the server by invoking the URL provided. If the WAP server is only a proxy, it fetches the content from the Origin server and sends it over the Bluetooth link to the mobile device. However, because the user is likely to move away from the server, the communication link may be lost as the device goes out of range of the piconet. In such a case, the information obtained through the SDP can be stored in the cache memory of the mobile device. The user can invoke the URL and access the content later through the normal mobile network. So, if the initiation is by the client, the content is obtained from the server by a mobile device using the pull model.

Initiation by the server

In this scenario, the WAP server checks periodically for the availability of clients using the SDP. If the server discovers a WAP-enabled device in the piconet environment, it connects to the client and pushes the message. Note that the actual content is not sent to the client and only a service indication is sent. This is a small descriptive message regarding the content to be pushed (for example, "Would you like to see the flight departure information?" at an airport) and the URL corresponding to the content. The client can discard the message or invoke the URL in the service indication to see the actual content.

When the server discovers the clients using the SDP, it obtains the following information:

- ◆ Client name (in a friendly format)
- ◆ Client capabilities (the Bluetooth capabilities of the mobile device)

So, if the initiation is by the server, the mobile device obtains the content from the server using the push model.

Protocol Stack for Using WAP with Bluetooth

To support WAP services with Bluetooth in the Client/Server environment, the client and the server have to run the protocol stack, as shown in Figure 8-2.

Because the mobile device can be a high-end device, such as a laptop, or a low-end device, such as a cellular phone, the stack shown in Figure 8-2 is very generic in order to accommodate a variety of mobile devices. The following is a brief description of each of the layers:

- ◆ Baseband/Link Manager Protocol (LMP) takes care of link management over the radio. This layer establishes the Bluetooth link, and the messages exchanged between the Bluetooth modules are not passed to the higher layers.
- ◆ Logical Link Control and Adaptation Protocol (L2CAP) assumes that a Bluetooth link is already established between two devices (in this case, the client and the server) and identifies the higher layer to which the messages have to be passed on (for example, SDP or RFCOMM). In other words, this layer carries out protocol multiplexing.
- ◆ Service Discovery Protocol (SDP) locates the services provided by other Bluetooth devices. These devices exchange SDP messages to discover the services of each other. One of the devices sends an SDP request and the other sends an SDP response. The responses are in the form of service records, which contain the details of the service. Either the WAP server or the mobile device can send the request and obtain the response. For push applications, the WAP server sends the SDP request to the mobile device and obtains the response. For pull applications, the mobile device sends the SDP request and obtains the response.

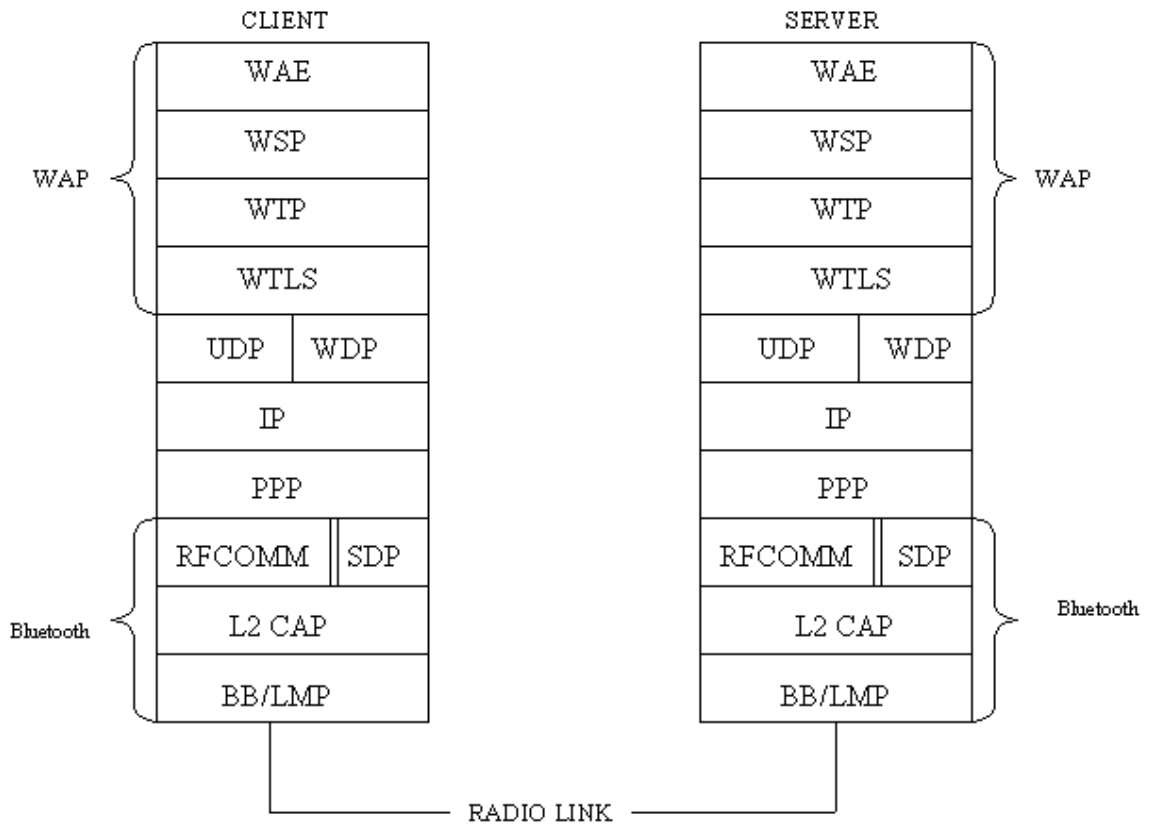


Figure 8-2: Protocol stack on Client and Server for WAP with Bluetooth

- ◆ RFCOMM is the transport protocol that emulates the RS 232 serial port, meaning we can assume that the communication between the mobile device and the server is in the form of serial communication without the cable.
- ◆ Point to Point Protocol (PPP) is the protocol used for dial-up lines to transport packet data from higher layers across the Bluetooth RFCOMM serial port emulator. Because we are emulating the serial communication dial up through RFCOMM, this protocol is required.
- ◆ Internet Protocol (IP) is the protocol that takes care of the addressing and routing on the Internet. Every device connected to the Internet is given an IP address. Every packet contains the source IP address and the destination IP address. The destination IP address is used to route the packets to the correct destination.
- ◆ User Datagram Protocol (UDP) is the transport layer protocol. Unlike Transmission Control Protocol (TCP) that uses connection-oriented service, UDP uses connectionless service. The advantage of UDP is its low protocol overhead as compared to TCP. However, the service is unreliable as packets may be lost.
- ◆ Wireless Datagram Protocol (WDP) is the transport layer equivalent of UDP, which is the transport layer protocol in the WAP stack.
- ◆ Wireless Transport Layer Security (WTLS) is the optional security layer of the WAP stack. This layer provides the optional functionality of authentication and encryption for applications that require secure communication.

- ◆ Wireless Transaction Protocol (WTP) and Wireless Session Protocol (WSP) together provide the HTTP functionality in the WAP environment. These protocols establish a session and communicate with the WAP server/gateway for obtaining the information. They then present the information to the user through the Wireless Application Environment (WAE), which has a micro-browser to interpret the WML content.
- ◆ However, note that the WAP services can also be provided through Short Messaging Service (SMS). In such a case, there is no need to run the IP and UDP protocols. The Wireless Datagram Protocol (WDP) can run on the SMS bearer. If security above WDP is required, the Wireless Transaction Layer Security (WTLS) layer is run (note that this is only an optional layer). Above WTLS, Wireless Transaction Protocol (WTP) and Wireless Session Protocol (WSP) may be run.

Figure 8-3 shows the complete protocol stacks that run on the WAP server/gateway and the Origin server to provide WAP services with Bluetooth.

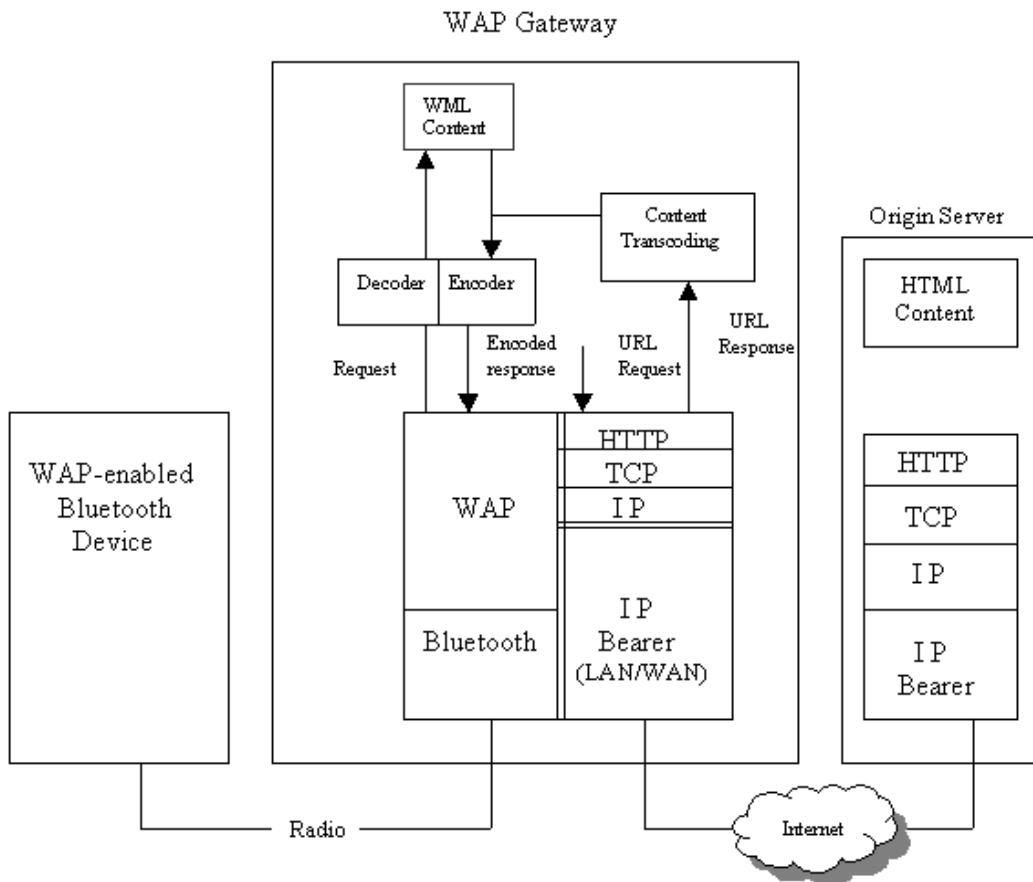


Figure 8-3: Protocol Stack on WAP Gateway and Origin server for WAP services with Bluetooth

The WAP client sends a request in the form of a URL through the Bluetooth bearer (over the radio link) to the WAP gateway. The Bluetooth-enabled WAP gateway receives the request. If the request corresponds to the content that is available locally on the WAP server, it will fetch the WML content, encode it in binary format, and send it to the client. If the content is not available locally, the WAP server contacts the Origin server through HTTP protocol, obtains the content, and sends it to the client. Because the Origin server has the TCP/IP protocol stack, the WAP server has to do the necessary protocol

conversion. If the Origin server sends the content in HTML instead of the WML format, the WAP gateway has to convert the content into WML format, do the encoding of the content, and send it to the mobile device over the Bluetooth bearer. For the WAP gateway to send the URL request to the right Origin server, the URL has to be mapped to the IP address, which is done by a Domain Name Server (DNS). Thus, one of the requirements for the WAP gateway is to have the capability of the DNS address mapping.

Interoperability requirements

WAP's interoperability with Bluetooth is considered one of the "killer applications" of Bluetooth. However, the implementation of WAP services with Bluetooth will happen gradually. The Bluetooth specifications include interoperability requirements for WAP and Bluetooth. The following are the important interoperability requirements to provide full-fledged services:

- ◆ Name of the server and the server capabilities should be sent to the client through the SDP.
- ◆ Name of the client and the client capabilities should be sent to the client through the SDP.
- ◆ When a device enters the RF proximity of another device, one device should automatically be notified about the presence of the other; this is known as asynchronous notification.

Implementation of WAP for Bluetooth

In the WAP protocol stack, the WDP management entity is responsible for managing the services provided by WDP. For implementing WAP with Bluetooth, the management entity should be capable of:

- ◆ **Detecting new clients:** Because mobile devices come within the proximity of the WAP server, make connections, and leave the piconet environment, the network is highly dynamic meaning new clients get connected and disconnected. So, detection of new clients is an important function of the management entity of the WDP.
- ◆ **Detecting new servers:** As the mobile device keeps moving, it comes within the proximity of Bluetooth enabled servers, and the management entity in the mobile devices should be capable of detecting new servers en route.
- ◆ **Detecting that the client node signal is lost:** The server should be able to detect when the mobile device goes out of its RF range and immediately frees the resources allocated to it.
- ◆ **Detecting that the server node signal is lost:** The mobile device should be capable of detecting the loss of an RF signal.
- ◆ **Detection of server push messages:** In the case of a push model being used for WAP over Bluetooth, the mobile device should be able to detect the server push messages. The same procedure as for Service Indication and Service Loading is followed in the case of push messages.

In the Bluetooth protocol stack, the Bluetooth Host Control Interface derives all information except the server push detection.

Addressing in WAP with Bluetooth

Bluetooth, as the bearer, will be transparent to the user to obtain the content. The user uses the normal URL to access the content using pull model. If the URL corresponds to the content present in that WAP gateway itself, the information is sent to the mobile device. If the content is not present in the gateway but has to be fetched by the gateway, the gateway should be capable of using the DNS to obtain the network address, which is a requirement stated in the Bluetooth specifications.

In the following sections, we will study two applications that demonstrate the capability of WAP in a Bluetooth environment. We will assume that the WAP server that provides the WML content and the

mobile device are Bluetooth-enabled and through SDP, the server can discover a mobile device in its vicinity to push the information onto the mobile device. The applications are:

- ◆ A kiosk in an airport, which provides gate information to the users
- ◆ A shopping mall kiosk, which provides information on the location of different shops and information about new shops

Application: Airport Kiosk

This application illustrates the push model usage for sending information on flights to mobile devices that come in the vicinity of a WAP server. When the mobile device enters the RF region of a Bluetooth-enabled WAP server, the WAP server discovers the mobile device and pushes the content.

We need to create a database that stores the flight information in the following format: flight number, destination, gate, and time of departure. You can create the database in MS Access that contains a table named flight. The fields in the table are flightno, destination, gate, and time. These are text fields in which the flight number, destination of the flight, gate number, and time of departure are stored, respectively. Figure 8-4 shows a sample database.

flight : Table				
	flightno	destination	gate	time
	AI112	Delhi	a1	12:35:00 PM
	AI445	Mumbai	b2	12:56:00 AM
	SA443	Qatar	a2	12:45:00 AM
	IA152	Madras	c1	10:09:35 AM
	BT1202	London	a1	10:10:35 AM

Figure 8-4: Database for flight information

Now we need to write a program that generates a WML card that pushes the information stored in the database to the mobile device. The program written in ASP is shown in Listing 8.1.

Listing 8-1: ASP code (flight.asp) for Airport Kiosk

© 2001 Dreamtech Software India Inc
All Rights Reserved

```

1. <% Response.ContentType = "text/vnd.wap.wml" %>
2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
4. <wml>
5. <%
6. set dbConn = Server.CreateObject("ADODB.Connection")
7. dbConn.open("dsn=flight")
8. set flight1 = dbConn.execute("SELECT * from flight")
9. %>
10. <card id="home" title="FLIGHT TIMINGS">
11. <p align="center">
12. <%

```

```

13. While not flight1.eof
14. %>
15. FLIGHT NO:
16. <%=flight1("flightno")%><br/>
17. DESTINATION:
18. <%=flight1("destination")%><br/>
19. GATE:
20. <%=flight1("gate")%><br/>
21. TIME
22. <%=flight1("time")%><br/>
23. <br/>
24. <%
25. flight1.MoveNext
26. Wend
27. %>
28. </p>
29. </card>
30. </wml>

```

Code Description

- ◆ Line 1: This line is to set the content type to WML. As per the MIME setting requirement, we set the content type of the response to "text/vnd.wap.wml"
- ◆ Line 2: Indicates the XML version being used
- ◆ Line 3: Indicates document type definition
- ◆ Line 4: Indicates the start of the WML deck
- ◆ Lines 5–9: This code creates the database object and opens the database using the DSN (Data Source Name). After opening the database, it executes a select statement to extract the details from the database and assigns the result to a variable `flight1`. The DSN used is `flight`.
- ◆ Line 10: Card tag with card id "home" and title of the card "FLIGHT TIMINGS"
- ◆ Line 11: Para tag with attribute to align the text at center
- ◆ Lines 12–14: This code is to create a loop to read all the records from the database until the end of the file. The While loop ends at line 26.
- ◆ Line 15: This is text to be displayed on the WML card: FLIGHT NO
- ◆ Line 16: The script to display the field "flightno" retrieved from the database
- ◆ Line 17: Simple text to display DESTINATION
- ◆ Line 18: The script to display the field "destination" retrieved from the database
- ◆ Line 19: Simple text to display GATE
- ◆ Line 20: The script to display the field "gate" retrieved from the database
- ◆ Line 21: Simple text to display TIME
- ◆ Line 22: The script to display the field "time" retrieved from the database
- ◆ Line 23: Break tag
- ◆ Lines 24–27: The script used for moving to the next record in the database and also for closing the While loop
- ◆ Lines 28–30: Closing tags for para, card and wml deck tags

Create the DSN

To create the DSN, on the Windows desktop go to Start⇒Settings⇒Control Panel. Now look for ODBC Data Sources and double-click the icon. The 32-bit data sources administration window is opened; in user DSN, click the Add button. It asks you to select the driver. Select the appropriate driver (in this case, MS Access) and click the Finish button. The ODBC Microsoft Access Setup window displays: there you have to give the data source name, select the database, and click the OK button. The DSN is created.

Call the Program

Create a folder named wap under inetpub/wwwroot. Place the `flight.asp` file in this folder. In the Nokia tool kit, go to the TOOLKIT⇒SHOW⇒PUSHVIEW menu and click the menu item. The Push Message window appears. Click the Createmsg button. The Push Message Simulator window opens. Pass the URL in the href text box and click the OK button. The push message simulator is shown in Figure 8-5.

Push Message Editor

SI | SL | CO

Service Indication Headers

Content-Type: text/vnd.wap.si

X-Wap-Application-Id:

X-Wap-Content-URI:

X-Wap-Initiator-URI:

Service Indication Content

action: signal-high

href: http://localhost/wap/flight.asp

si-id:

si-expires: 1999-01-02T01:28:39Z

created: 1999-01-01T01:28:39Z

Text Message: This is displayed text.

Class 1:

Info 1:

Class 2:

Info 2:

OK Store Source Store Binary Cancel

Figure 8-5: Push message simulator for Airport Kiosk

In the push message menu window, click the activate msg button. The information in the URL (`http://localhost/wap/flight.asp`) is pushed to the browser. Because the server machine is the same as the client on which we are testing this application, we used localhost as the server name. Otherwise, we must use the name of the server, for example, `http://MyServer/wap/flight.asp`. The tool kit automatically creates a Service Indication (SI). The SI code is shown in Listing 8-2.

Listing 8-2: Service Indication generated by the Tool Kit for Airport Kiosk Push Message

© 2001 Dreamtech Software India Inc
All Rights Reserved

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN"
"http://www.wapforum.org/DTD/si.dtd">
<si>
  <indication
    action="signal-high"
    href="http://localhost/wap/flight.asp"
    si-id="http://localhost/wap/flight.asp"
    created="1999-01-01T01:48:40Z"
    si-expires="1999-01-02T01:48:40Z"
  >
    This is displayed text.
  </indication>
</si>
```

The si tag has action attribute as "signal-high", followed by href, si-id, created and si-expires attributes. The href is the URL, si-id is the unique ID assigned to the service indication message, created specifies the date and time of creation of the message, and si-expires indicates the expiry time of the service indication.

Code Output

The output displayed on the phone emulator is shown in Figure 8-6.



Figure 8-6: Flight information displayed on the mobile phone

Application: Shopping Mall Kiosk

In this example, we will create a kiosk that can be placed in a large shopping mall. This kiosk pushes information about shop locations and new products to the mobile devices. As in the previous example, we need to create a database and an ASP program.

First, create a database in MS Access called shop, which contains one table with the name “item.” See Figure 8-7.

item : Table				
	item	shopno	floor	status
▶	furniture	1001	1	
	sports	2002	2	
	music	3001	3	new
	garments	3002	3	new
*				

Figure 8-7: Database for Shopping Mall Kiosk

The fields in the table are item, shopno, floor, and status. All the fields are text fields. The item field gives the name of the item, shop number gives the number of the shop, floor indicates the floor number on which the shop is located, and the status field gives whether the shop has any new products in stock. If there are no new arrivals, the field can be blank in the records.

In regard to the ASP code for this application, we will provide an option to the user. As soon as the user enters the RF proximity of the WAP server, a small display appears that has a soft key called New. The soft key is a software-generated key — one of the buttons on the keypad can be programmed to do a specific task. The user can go through the shops’ details in order or only press the New button, in which case only those records with “new” in the status field are displayed. Thus we will write two ASP programs.

Shop.asp code is shown in Listing 8-3.

Listing 8-3: ASP code (shop.asp) for Shopping Mall Kiosk

© 2001 Dreamtech Software India Inc
All Rights Reserved

```

1. <% Response.ContentType = "text/vnd.wap.wml" %>
2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml12.dtd">
4. <wml>
5. <%
6. set dbConn = Server.CreateObject("ADODB.Connection")
7. dbConn.open("dsn=shop")
8. set shop = dbConn.execute("SELECT * from item")
9. %>
10 <card id="home" title="SUPERMARKET">
11. <p align="center">
12. <do type="accept" label="New">
```

```

13. <go href="shopnew.asp" />
14. </do>
15. <%
16. While not shop.eof
17. %>
18. ITEM:
19. <%=shop("item")%><br/>
20. SHOPNO:
21. <%=shop("shopno")%><br/>
22. FLOOR:
23. <%=shop("floor")%><br/>
24. <%
25. shop.MoveNext
26. Wend
27. %>
28. </p>
29. </card>
30. </wml>

```

Code Description

- ◆ Line 1: Sets the content type that is being used in the file
- ◆ Line 2: Indicates the XML version being used
- ◆ Line 3: Document type definition that is being used
- ◆ Line 4: wml deck starting
- ◆ Lines 5–9: Code to create the database object and open the database using the DSN. After opening the database, it executes a select statement to extract the details from the database and assigns the results to a variable shop.
- ◆ Line 10: Card tag with card id "home" and title of the card "SUPERMARKET"
- ◆ Line 11: Para tag with attribute to align the text at center
- ◆ Lines 12–14: Do tag which is used to assign a task. The task type is "accept", label is "New". The task is to navigate to a new link given by href (shopnew.asp). So, we are creating a softkey "New" and after the user presses this key on the mobile device, shopnew.asp is invoked.
- ◆ Lines 15–17: The script to create a while loop to read the records in the database one by one, till the end of the file
- ◆ Line 18: Simple text to display ITEM
- ◆ Line 19: Script to display the field "item" retrieved from the database
- ◆ Line 20: Simple text to display SHOPNO
- ◆ Line 21: Script to display the field "shopno" retrieved from the database
- ◆ Line 22: Simple text to display FLOOR
- ◆ Line 23: Script to display the field "floor" retrieved from the database
- ◆ Lines 24–27: Script to move to the next record in the database and end of the while loop on reaching the end of file
- ◆ Lines 28–30: Closing of para, card, and wml deck tags

In the above code, we referred to shopnew.asp (through href attribute in the go tag in Line 13). This code is invoked after the "New" soft key is pressed. Listing 8-4 shows the code for shopnew.asp.

Listing 8-4: ASP code (shopnew.asp) for Shopping Mall Kiosk

© 2001 Dreamtech Software India Inc
All Rights Reserved

```

1. <% Response.ContentType = "text/vnd.wap.wml" %>
2. <?xml version="1.0"?>
3. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml12.dtd">
4. <wml>
5. <%
6. set dbConn = Server.CreateObject("ADODB.Connection")
7. dbConn.open("dsn=shop")
8. set shop = dbConn.execute("SELECT * from item where status = 'new' ")
9. %>
10. <card id="home" title="SUPERMARKET">
11. <p align="center">
12. <do type="accept" label="Home">
13. <go href="shop.asp" />
14. </do>
15. <%
16. While not shop.eof
17. %>
18. ITEM:
19. <%=shop("item")%><br/>
20. SHOPNO:
21. <%=shop("shopno")%><br/>
22. FLOOR:
23. <%=shop("floor")%><br/>
24. <%
25. shop.MoveNext
26. Wend
27. %>
28. </p>
29. </card>
30. </wml>

```

Code Description

This code is the same as the `shop.asp` except in retrieving the data from the database. Instead of retrieving all the records, this code retrieves only those records with category as new.

Lines 5-9: This code creates a database object and opens the database using the DSN. After opening the database, it executes a select statement to extract details from the database. Note the `where` clause in the select statement. This statement retrieves only those records for which the status is 'new'. The result is assigned to a variable `shop`.

Create the DSN

To create the DSN, follow the same procedure as given for the Airport Kiosk application.

Call the Program

Place the asp file in the inetpub wwwroot by creating a folder wap. In the Nokia tool kit, go to the TOOLKIT⇒SHOW⇒PUSHVIEW menu and click the menu item. The push menu window opens. Click the Createmsg button.

The push message simulator window displays, as shown in Figure 8-8.

Push Message Editor

SI | SL | CO

Service Indication Headers

Content-Type: text/vnd.wap.si

X-Wap-Application-Id:

X-Wap-Content-URI:

X-Wap-Initiator-URI:

Service Indication Content

action: signal-high

href: http://localhost/wap/shop.asp

si-id:

si-expires: 1999-01-02T01:28:39Z

created: 1999-01-01T01:28:39Z

Text Message: This is displayed text.

Class 1:

Info 1:

Class 2:

Info 2:

OK Store Source Store Binary Cancel

Figure 8-8: Push Message Simulator for Shopping Mall

Pass the URL in the href box and click OK. In the push menu window, click the activate msg button. The information in the URL (<http://www.localhost/wap/shop.asp>) is pushed on to the browser. The push simulator automatically generates a Service Indication, as shown in Listing 8-5.

Listing 8-5: Service Indication generated by Tool Kit for Shopping Mall Push Message

```
© 2001 Dreamtech Software India Inc
All Rights Reserved<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN"
"http://www.wapforum.org/DTD/si.dtd">

<si>
  <indication
    action="signal-high"
    href="http://localhost/wap/shop.asp"
    si-id="http://localhost/wap/shop.asp"
    created="1999-01-01T01:48:40Z"
    si-expires="1999-01-02T01:48:40Z"
  >
```

```

This is displayed text.
</indication>
</si>

```

Code Output

The display on the WAP phone emulator is shown in Figure 8-9. By default, the entire list is displayed, which can be scrolled, as shown in Figure 8-9(a). Figure 8-9(b) shows the display when the New soft key is pressed, where only those records with status of new are displayed.

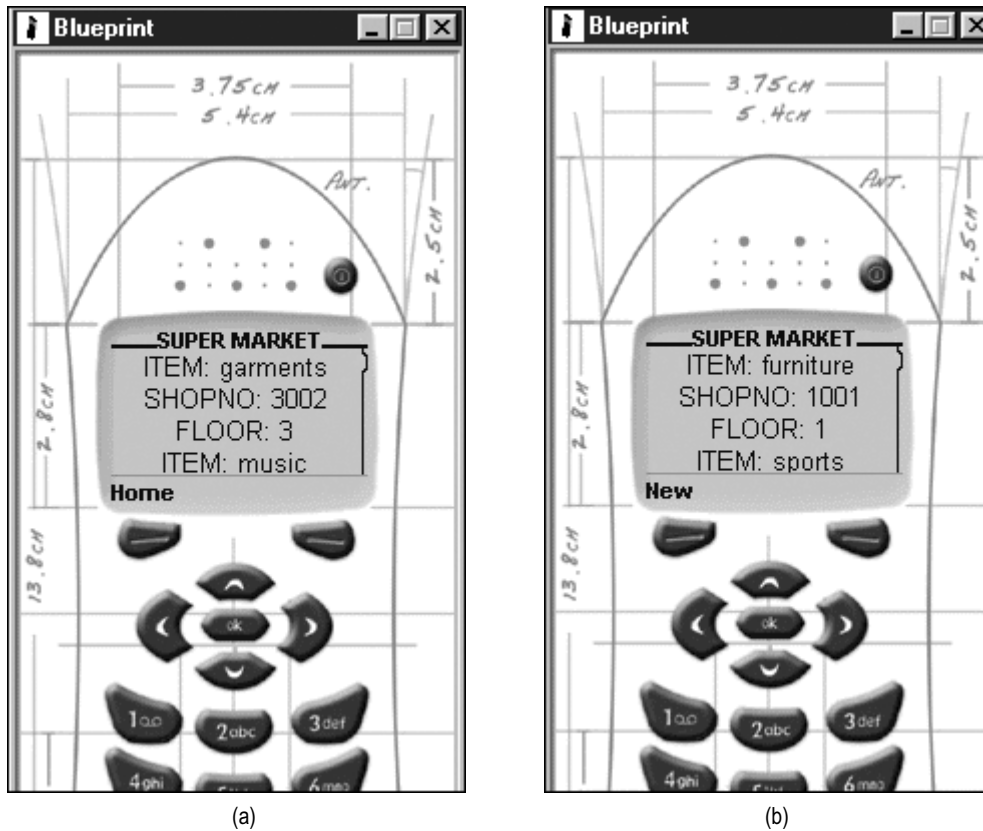


Figure 8-9: Push Message from WAP Server displayed on the mobile phone: (a) display of records one by one and (b) display of records having status of new

Summary

In this chapter, we studied how WAP services can be provided on Bluetooth-enabled devices. Bluetooth acts as a bearer for WAP stack. A WAP server can communicate with a WAP client over a Bluetooth radio link, and through either a pull model or a push model, the WAP content can be presented on the mobile device. Implementing WAP with Bluetooth can provide hidden computing services, such as the “briefcase trick” and “forbidden message.” Kiosks that are Bluetooth-enabled WAP servers/gateways can provide location specific information to the users in various places, such as airports and malls. A kiosk is an interesting example of ad-hoc networks, wherein the mobile devices enter the server’s RF range, discover the services available through the Service Discovery Protocol, and get out of the range in a random fashion. We also studied the implementation of kiosks using the push framework.

Chapter 9

Bluetooth Programming

In this chapter, we focus on the programming aspects of Bluetooth. We see how to access the different layers of the Bluetooth protocol stack and also develop applications in the Windows environment by using Ericsson's Bluetooth PC Reference stack. The complete source code listings for accessing the Host Controller Interface (HCI), Service Discovery Protocol (SDP), and RFCOMM using the Application Programming Interface (API) calls provided with the PC reference stack are presented.

Overview of the Bluetooth Development Kit

The Bluetooth development kit used for illustrating Bluetooth programming is Ericsson's PC reference stack distributed and supported by Teleca Comtec (<http://www.comtec.teleca.se>). The development kit consists of a Bluetooth module (a small Printed Circuit Board) containing the hardware and firmware and a software module that runs on the Windows NT/98/2000 environment. For carrying out development work, we need at least two such modules.

The Bluetooth module works on either a USB interface or a serial interface. The PC reference stack software has to be loaded on the system. The stack provides the API calls for HCI driver, L2CAP, SDP, RFCOMM, and OBEX (Object Exchange) protocol layers. The development is done in VC++ environment. Because the software uses the Windows COM model, familiarity with COM is assumed.

In this chapter, we discuss how the services provided by different layers are accessed through the API calls. The programming aspects give better insight into the protocols and functioning of Bluetooth. The complete source code is given along with a detailed explanation. The explanation covers the description of the API calls.

Installing the Bluetooth Module and PC Reference Stack

Before trying the following applications, you must install the Bluetooth module and software on at least two systems. The Bluetooth module can be connected to the USB interface or serial interface of the PC. The PC reference stack and the driver software need to be installed on the systems (the Bluetooth hosts). The sample chat application given with the PC reference stack can be executed and tested to ensure that the Bluetooth modules are working in a proper manner.

After you have the previous requirements (consisting of two Bluetooth-enabled desktop PCs) set up, you can start working on the following examples.

HCI Programming

The HCI (Host Controller Interface) driver that runs on the host (the PC) is used to carry out a number of functions. These include configuring the port on which the Bluetooth module is placed, obtaining the local Bluetooth module address, obtaining the version number of the Bluetooth device, obtaining the packet sizes supported for ACL (Asynchronous Connection Less) and SCO (Synchronous Connection Oriented) links, sending an inquiry to a remote Bluetooth device, obtaining its address, and doing loopback testing. When the Bluetooth module is connected to the PC, the first step is to establish communication between the protocol stack running on the PC and the Bluetooth module. So, we start

Bluetooth programming with HCI programming. This example illustrates the HCI commands and responses. Note that HCI programming is fundamental to Bluetooth programming because the first step for establishing a connection between two Bluetooth devices is for the stack on the host to communicate with the Bluetooth module.

This application contains the following three modules:

- ◆ GUI: This module provides a good user interface to generate HCI commands. For each HCI command generated by the user, the Bluetooth module fires a HCI event. The user can identify whether the issued command is a success by getting the fired HCI Event information in the message boxes. This module is implemented with `CHCIInformationCommandsDlg` class in a file named `HCI Information CommandsDlg.cpp`. The variables used in this class are declared in `HCI Information CommandsDlg.h` (Listing 9-1).
- ◆ EVENTS: This module connects to the `BT_COMServer` (COM Server supplied by Ericsson along with the BLUETOOTH PC reference stack) and receives all outgoing events from the `BT_COMServer`. It is implemented with `Events` class in `Events.cpp` file. The variables used in this class are declared in `Events.h`.
- ◆ REMOTE DEVICE: This module is to present the Remote BLUETOOTH device address to the user. It is implemented with `CRemoteDevice` class in a file named `RemoteDevice.cpp`. The variables used in this class are declared in `RemoteDevice.h`.

Listing 9-1: HCI Information CommandsDlg.h

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. // HCI Information CommandsDlg.h : header file
2. //
3. #if !defined(AFX_HCIINFORMATIONCOMMANDSDLG_H__30FC7B07_A10F_
        11D2_ B756_0080C805A679__INCLUDED_)
4. #define AFX_HCIINFORMATIONCOMMANDSDLG_H__30FC7B07_A10F_11D2_B756_
        0080C805A679__INCLUDED_
5. #if _MSC_VER > 1000
6. #include "Events.h"
7. #include "RemoteDevice.h"
8. #include <afxtempl.h>
9. #pragma once
10. #endif // _MSC_VER > 1000
11. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
12. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
        WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
13. (AFX_PMSG) (AFX_PMSGW) (void (AFX_MSG_CALL CWnd::*) (void**)) & memberFxn },
14. //Macro to send a Bluetooth Event to the windows message map
15. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
16.     SendMessage( (HWND) this->m_hWnd, WM_BLUETOOTH_EVENT, (WPARAM)
        uiBtEventID, (LPARAM) &pMsg)
17. class CHCIInformationCommandsDlg : public CDialog
18. {
19. // Construction
20. public:
21.
22. CHCIInformationCommandsDlg(CWnd* pParent = NULL);
23. enum { IDD = IDD_HCIINFORMATIONCOMMANDS_DIALOG };
24. CEdit m_add;
25. CListBox m_DeviceList;
26. CEdit m_ScoCount;

```

```

27. CEdit m_ScoSize;
28. CEdit m_AclCount;
29. CEdit m_AclSize;
30. CEdit m_ver;
31. //}}AFX_DATA
32. // ClassWizard generated virtual function overrides
33. //{AFX_VIRTUAL(CHCIInformationCommandsDlg)
34. protected:
35. virtual void DoDataExchange(CDataExchange* pDX);
36. virtual LRESULT WindowProc(UINT message, WPARAM wParam,
                                LPARAM lParam);
37. //}}AFX_VIRTUAL
38. public:
39. void AddDevice(CRemoteDevice device);
40. void ShowAllDevicesFound();
41. // Implementation
42. protected:
43. HICON m_hIcon;
44. CArray <CRemoteDevice,CRemoteDevice&> m_DevicesFound;
45. Events *m_pServerEvents;
46. int m_RemoteNameCounter;
47. int m_ServiceCounter;
48. // Generated message map functions
49. //{AFX_MSG(CHCIInformationCommandsDlg)
50. virtual BOOL OnInitDialog();
51. afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
52. afx_msg void OnPaint();
53. afx_msg HCURSOR OnQueryDragIcon();
54. afx_msg void OnInquiry();
55. afx_msg void OnConnect();
56. afx_msg void OnDestroy();
57. afx_msg void OnCloseapplication();
58.
59. afx_msg void OnComStartCnf(void **ppMsg);
60. afx_msg void OnComStartCnfNeg(void **ppMsg);
61.
62. afx_msg void OnHciConfigurePortConfirm(void **ppMsg);
63. afx_msg void OnHciConfigurePortConfirmNegative(void **ppMsg);
64. afx_msg void OnHciInquiryCnf(void **ppMsg);
65. afx_msg void OnHciInquiryEvt(void **ppMsg);
66. afx_msg void OnHciLocalAddressCnf(void **ppMsg);
67. afx_msg void OnHciLocalAddressCnfNeg(void **ppMsg);
68. afx_msg void OnHciRemoteNameCnf(void **ppMsg);
69. afx_msg void OnHciRemoteNameCnfNeg(void **ppMsg);
70. afx_msg void OnHciStartCnf(void **ppMsg);
71. afx_msg void OnHciReadLocalVersionCnf(void **ppMsg);
72. afx_msg void OnHciReadLocalVersionCnfNeg(void **ppMsg);
73. afx_msg void OnHciWriteLoopbackModeCnf(void **ppMsg);
74. afx_msg void OnHciWriteLoopbackModeCnfNeg(void **ppMsg);
75. afx_msg void OnHciDataInfoCnf(void **ppMsg);
76. afx_msg void OnSilSetDeviceCnf(void **ppMsg);
77. afx_msg void OnSilSetDeviceCnfNeg(void **ppMsg);
78. afx_msg void OnButton1();
79. afx_msg void OnButton2();
80. afx_msg void OnButton3();
81. afx_msg void OnButton5();

```

```

82.  afx_msg void OnButton7();
83.  //}}AFX_MSG
84.  DECLARE_MESSAGE_MAP()
85.  private:
86.
87.  BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
88.  };
89.  //{AFX_INSERT_LOCATION}}
90.  #endif

```

Code Description

Listing 9-1 is a header file for declaration of the necessary constants and declaration of the class and its methods. Note that the MFC application wizard automatically generates lines 1–5, 9, and 10. Lines 6 to 8 are the include files required for our program. `Events.h` and `Remotedevice.h` are the header files, which are created for our application and `afxtempl.h` is a library file. Lines 11–13 and lines 15 and 16 are constant definitions as required by the Bluetooth PC reference stack. Lines 17–90 are for the declaration of the class `CHCIInformationCommandsDialog` for creation of buttons, edit boxes, and message boxes for displaying various messages when events are thrown by the Bluetooth module for various HCI commands issues by the host. The HCI commands and messages are explained in Listing 9-2.

Listing 9-2: HCI Information CommandsDlg.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1.  // HCI Information CommandsDlg.cpp : implementation file
2.  #include "stdafx.h"
3.  #include "HCI Information Commands.h"
4.  #include "HCI Information CommandsDlg.h"
5.  #include <exp\msg.h>
6.  #include <exp\vos2com.h>
7.  #include <exp\hci.h>
8.  #include <exp\hci_drv.h>
9.  #include <exp\sil.h>
10. #include <exp\com.h>
11. #include <exp\sd.h>
12. uint16 SeqNr;
13. #ifdef _DEBUG
14. #define new DEBUG_NEW
15. #undef THIS_FILE
16. static char THIS_FILE[] = __FILE__;
17. #endif
18. union MessageMapFunctions
19. {
20.  AFX_PMSG pfn;    // generic member function pointer
21.  void      (AFX_MSG_CALL CWnd::*pfn_btfn)(void **);
22. };
23. #define PORTSETTINGS (uint8 *)("COM1:Baud=57600 parity=N data=8 stop=1 ")
24. class CAboutDlg : public CDialog
25. {
26. public:
27.  CAboutDlg();
28.
29. // Dialog Data
30.  //{AFX_DATA(CAboutDlg)

```

```

31. enum { IDD = IDD_ABOUTBOX };
32. //}}AFX_DATA
33.
34. // ClassWizard generated virtual function overrides
35. //{AFX_VIRTUAL(CAboutDlg)
36. protected:
37. virtual void DoDataExchange(CDataExchange* pDX);
38. //}}AFX_VIRTUAL
39.
40. // Implementation
41. protected:
42. //{AFX_MSG(CAboutDlg)
43. //}}AFX_MSG
44. DECLARE_MESSAGE_MAP()
45. };
46.
47. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
48. {
49. //{AFX_DATA_INIT(CAboutDlg)
50. //}}AFX_DATA_INIT
51. }
52.
53. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
54. {
55. CDialog::DoDataExchange(pDX);
56. //{AFX_DATA_MAP(CAboutDlg)
57. //}}AFX_DATA_MAP
58. }
59.
60. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
61. //{AFX_MSG_MAP(CAboutDlg)
62. // No message handlers
63. //}}AFX_MSG_MAP
64. END_MESSAGE_MAP()
65. // CHCIInformationCommandsDlg dialog
66.
67. CHCIInformationCommandsDlg::CHCIInformationCommandsDlg(CWnd* pParent)
68. : CDialog(CHCIInformationCommandsDlg::IDD, pParent)
69. {
70. //{AFX_DATA_INIT(CHCIInformationCommandsDlg)
71. //}}AFX_DATA_INIT
72.
73. m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
74. m_pServerEvents = new Events();
75. }
76.
77. void CHCIInformationCommandsDlg::DoDataExchange(CDataExchange* pDX)
78. {
79. CDialog::DoDataExchange(pDX);
80. //{AFX_DATA_MAP(CHCIInformationCommandsDlg)
81. DDX_Control(pDX, IDC_EDIT1, m_add);
82. DDX_Control(pDX, IDC_LIST1, m_DeviceList);
83. DDX_Control(pDX, IDC_EDIT8, m_ScoCount);
84. DDX_Control(pDX, IDC_EDIT7, m_ScoSize);
85. DDX_Control(pDX, IDC_EDIT6, m_AclCount);
86. DDX_Control(pDX, IDC_EDIT5, m_AclSize);

```



```

87. DDX_Control(pDX, IDC_EDIT2, m_ver);
88. //}}AFX_DATA_MAP
89. }
90.
91. BEGIN_MESSAGE_MAP(CHCIIInformationCommandsDlg, CDialog)
92. //{AFX_MSG_MAP(CHCIIInformationCommandsDlg)
93. ON_WM_SYSCOMMAND()
94. ON_WM_PAINT()
95. ON_WM_QUERYDRAGICON()
96. ON_BLUEETOOTH_EVENT(COM_START_CNF, OnComStartCnf)
97. ON_BLUEETOOTH_EVENT(COM_START_CNF_NEG, OnComStartCnfNeg)
98.
99. ON_BLUEETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF, OnHciConfigurePortConfirm)
100. ON_BLUEETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF_NEG,
    OnHciConfigurePortConfirmNegative)
101. ON_BLUEETOOTH_EVENT(HCI_INQUIRY_CNF, OnHciInquiryCnf)
102. ON_BLUEETOOTH_EVENT(HCI_INQUIRY_EVT, OnHciInquiryEvt)
103. ON_BLUEETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF, OnHciLocalAddressCnf)
104. ON_BLUEETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF_NEG, OnHciLocalAddressCnfNeg)
105. ON_BLUEETOOTH_EVENT(HCI_REMOTE_NAME_CNF, OnHciRemoteNameCnf)
106. ON_BLUEETOOTH_EVENT(HCI_REMOTE_NAME_CNF_NEG, OnHciRemoteNameCnfNeg)
107. ON_BLUEETOOTH_EVENT(HCI_START_CNF, OnHciStartCnf)
108. ON_BLUEETOOTH_EVENT(HCI_READ_LOCAL_VERSION_CNF,
    OnHciReadLocalVersionCnf)
109. ON_BLUEETOOTH_EVENT(HCI_READ_LOCAL_VERSION_CNF_NEG,
    OnHciReadLocalVersionCnfNeg)
110. ON_BLUEETOOTH_EVENT(HCI_WRITE_LOOPBACK_MODE_CNF,
    OnHciWriteLoopbackModeCnf)
111. ON_BLUEETOOTH_EVENT(HCI_WRITE_LOOPBACK_MODE_CNF_NEG,
    OnHciWriteLoopbackModeCnfNeg)
112. ON_BLUEETOOTH_EVENT(HCI_DATA_INFO_CNF, OnHciDataInfoCnf)
113. ON_BLUEETOOTH_EVENT(SIL_SET_DEVICE_CNF, OnSilSetDeviceCnf)
114. ON_BLUEETOOTH_EVENT(SIL_SET_DEVICE_CNF_NEG, OnSilSetDeviceCnfNeg)
115. ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
116. ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
117. ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
118. ON_BN_CLICKED(IDC_BUTTON5, OnButton5)
119. ON_BN_CLICKED(IDC_BUTTON7, OnButton7)
120.
121. //}}AFX_MSG_MAP
122. END_MESSAGE_MAP()
123. // CHCIIInformationCommandsDlg message handlers
124. BOOL CHCIIInformationCommandsDlg::OnInitDialog()
125. {
126. CDialog::OnInitDialog();
127. // Add "About..." menu item to system menu.
128. // IDM_ABOUTBOX must be in the system command range.
129. ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
130. ASSERT(IDM_ABOUTBOX < 0xF000);
131. CMenu* pSysMenu = GetSystemMenu(FALSE);
132. if (pSysMenu != NULL)
133. {
134. CString strAboutMenu;
135. strAboutMenu.LoadString(IDS_ABOUTBOX);
136. if (!strAboutMenu.IsEmpty())
137. {

```

```

138.         pSysMenu->AppendMenu(MF_SEPARATOR);
139.         pSysMenu->AppendMenu(MF_STRING,
                                IDM_ABOUTBOX , strAboutMenu );
140.     }
141. }
142. SetIcon(m_hIcon, TRUE);           // Set big icon
143. SetIcon(m_hIcon, FALSE);         // Set small icon
144.     m_pServerEvents->m_pParentDialog = this;
145. return TRUE; // return TRUE unless you set the focus to a control
146. }
147.
148. void CHCIInformationCommandsDlg::OnSysCommand(UINT nID, LPARAM lParam)
149. {
150.     if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
151.     {
152.         CAboutDlg dlgAbout;
153.         dlgAbout.DoModal();
154.     }
155.     else
156.     {
157.         CDialog::OnSysCommand(nID, lParam);
158.     }
159. }
160.
161. void CHCIInformationCommandsDlg::OnPaint()
162. {
163.     if (IsIconic())
164.     {
165.         CPaintDC dc(this); // device context for painting
166.         SendMessage(WM_ICONERASEBKGND, (LPARAM)
                                dc.GetSafeHdc(), 0);
167.         // Center icon in client rectangle
168.         int cxIcon = GetSystemMetrics(SM_CXICON);
169.         int cyIcon = GetSystemMetrics(SM_CYICON);
170.         CRect rect;
171.         GetClientRect(&rect);
172.         int x = (rect.Width() - cxIcon + 1) / 2;
173.         int y = (rect.Height() - cyIcon + 1) / 2;
174.         // Draw the icon
175.         dc.DrawIcon(x, y, m_hIcon);
176.     }
177.     else
178.     {
179.         CDialog::OnPaint();
180.     }
181. }
182.
183. HCURSOR CHCIInformationCommandsDlg::OnQueryDragIcon()
184. {
185.     return (HCURSOR) m_hIcon;
186. }
187. LRESULT CHCIInformationCommandsDlg::WindowProc(UINT message, WPARAM
                                                wParam, LPARAM lParam)
188. {
189.     // TODO: Add your specialized code here and/or call the base class
190.     MSG_TMsg **ptMsg;

```

```

191.
192. if (message == WM_BLUETOOTH_EVENT)
193. {
194.     // It is a Bluetooth event, so call the corresponding handlefunction
195.     OnBluetoothEvent( message, wParam, lParam);
196.     ptMsg = (MSG_TMsg**)lParam;
197.     /* free the message received from the bluetooth server */
198.     if (*ptMsg != NULL)
199.         VOS_Free((void **)lParam);
200. }
201. return CDialog::WindowProc(message, wParam, lParam);
202. }
203. BOOL CHCIInformationCommandsDlg::OnBluetoothEvent(UINT message, WPARAM
                                     wParam, LPARAM lParam)
204. {
205.     const AFX_MSGMAP* pMessageMap;
206.     const AFX_MSGMAP_ENTRY* lpEntry;
207.     // look through message map to see if it applies to us
208. #ifdef _AFXDLL
209.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
210.          pMessageMap = (*pMessageMap->pfnGetBaseMap)())
211. #else
212.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
213.          pMessageMap = pMessageMap->pBaseMap)
214. #endif
215.     {
216. #ifdef _AFXDLL
217.         ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
218. #else
219.         ASSERT(pMessageMap != pMessageMap->pBaseMap);
220. #endif
221.         lpEntry = (AFX_MSGMAP_ENTRY*)&pMessageMap->lpEntries[0];
222.         while (lpEntry->nSig != AfxSig_end)
223.         {
224.             if((lpEntry->nMessage==message)&&(lpEntry->nCode== wParam))
225.             {
226.                 union MessageMapFunctions mmf;
227.                 mmf.pfn = lpEntry->pfn;
228.
229.                 //lets call the function to handle the message
230.                 (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
231.
232.                 return TRUE;
233.             }
234.             lpEntry++;
235.         }
236.         /* unable to find a handler function for this Bluetooth event */
237.         return FALSE;
238.     }
239. return FALSE;
240. }
241. void CHCIInformationCommandsDlg::OnInquiry()
242. {
243.     HCI_TLap tLap = {0x9E,0x8B,0x33};
244.     HCI_TInquiryLength tInquiryLength = 2;
245.     HCI_TNrOfResponses tNrOfResponses = 0;

```

```

246.     AfxMessageBox("Request to send INQUIRY
        \nCommand:HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses)");
247.     HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses);
248. }
249. void CHCIIInformationCommandsDlg::OnDestroy()
250. {
251.     CDialog::OnDestroy();
252. }
253. void CHCIIInformationCommandsDlg::AddDevice(CRemoteDevice device)
254. {
255.     m_DevicesFound.Add(device);
256. }
257. void CHCIIInformationCommandsDlg::ShowAllDevicesFound()
258. {
259.     CRemoteDevice device;
260.     int iFound,i;
261.     CString str;
262.     iFound = m_DevicesFound.GetSize();
263.     for (i=0; i < iFound; i++)
264.     {
265.         device = m_DevicesFound.GetAt(i);
266.         str.Format("Address:0x%s      Name: %s",
            device.GetAddress(),device.GetName());
267.         m_DeviceList.AddString(str);
268.     }
268. }
269. void CHCIIInformationCommandsDlg::OnSilSetDeviceCnf(void **ppMsg)
270. {
271.     ppMsg = ppMsg;
272.     AfxMessageBox("UART INTERFACE was selected
        \nEvent:SIL_SET_DEVICE_CNF");
273.     AfxMessageBox("Request to Configure Port
        \nCommand:HCI_ReqConfigurePort(0,PORTSETTINGS)");
274.     HCI_ReqConfigurePort(0,PORTSETTINGS);
275. }
276. void CHCIIInformationCommandsDlg::OnSilSetDeviceCnfNeg(void **ppMsg)
277. {
278.     AfxMessageBox("INTERFACE was not
        selected\nEvent:SIL_SET_DEVICE_CNF_NEG");
279. }
280. void CHCIIInformationCommandsDlg::OnHciConfigurePortConfirm(void **ppMsg)
281. {
282.     //HCI_TConfigurePortCnf *tConfigurePort = (HCI_TConfigurePortCnf*)
        *ppMsg
283.     tConfigurePort = tConfigurePort;
284.     AfxMessageBox("Serial Port was
        Configured\nEvent:HCI_CONFIGURE_PORT_CNF");
285.     AfxMessageBox("Request to start RFCOMM \nCommand:COM_ReqStart(0)");
286.     COM_ReqStart(0);
287. }
288. void HCIInformationCommandsDlg::OnHciConfigurePortConfirmNegative(void
        **ppMsg)
289. {
290.     // HCI_TConfigurePortCnfNeg *tConfigurePort = (HCI_TConfigurePortCnfNeg
        *)*ppMsg;
291.     tConfigurePort = tConfigurePort;

```

```

292.     MessageBox(_T("Could not open port"));
293. }
294. void CHCIInformationCommandsDlg::OnComStartCnf(void **ppMsg)
295. {
296.     // COM_TStartCnf *tStartCnf = (COM_TStartCnf *)*ppMsg;
297.     tStartCnf = tStartCnf;
298.     AfxMessageBox("RFCOMM was started \nEvent:COM_START_CNF ");
299.     AfxMessageBox("Request to get Local
        BD_ADDRESS\nCommand:HCI_ReqLocalAddress(0)");
300.     HCI_ReqLocalAddress(0);
301. }
302. void CHCIInformationCommandsDlg::OnComStartCnfNeg(void **ppMsg)
303. {
304.     // COM_TStartCnfNeg *tStartCnfNeg = (COM_TStartCnfNeg *)*ppMsg;
305.     tStartCnfNeg = tStartCnfNeg;
306.     MessageBox(_T("Could not start RFCOMM"));
307. }
308. void CHCIInformationCommandsDlg::OnHciLocalAddressCnf(void **ppMsg)
309. {
310.     HCI_TLocalAddressCnf *tLocalAddress = (HCI_TLocalAddressCnf
        *)*ppMsg;
311.     char add[59];
312.     wsprintf(&add[0], "0x%02X%02X%02X%02X%02X%02X\0",
313.         tLocalAddress->tAddress.ucByte0,
314.         tLocalAddress->tAddress.ucByte1,
315.         tLocalAddress->tAddress.ucByte2,
316.         tLocalAddress->tAddress.ucByte3,
317.         tLocalAddress->tAddress.ucByte4,
318.         tLocalAddress->tAddress.ucByte5);
319.     AfxMessageBox("Returned Localaddress\nEvent:HCI_LOCAL_ADDRESS_CNF");
320.     m_add.SetWindowText(_T(add));
321. }
322. }
323. void CHCIInformationCommandsDlg::OnHciLocalAddressCnfNeg(void **ppMsg)
324. {
325.     ppMsg = ppMsg;
326.     m_add.SetWindowText(_T("UNABLE TO CONNECT TO
        DEVICE"));
327. }
328. void CHCIInformationCommandsDlg::OnHciInquiryCnf(void **ppMsg)
329. {
330.     HCI_TInquiryCnf *ptInquiryCnf;
331.     int count;
332.     CRemoteDevice device;
333.     AfxMessageBox("INQUIRY was sent\nEvent:HCI_INQUIRY_CNF");
334.     ptInquiryCnf = (HCI_TInquiryCnf *) *ppMsg;
335.     count = m_DevicesFound.GetSize();
336.     m_RemoteNameCounter = 0;
337.     if (count > 0)
338.     {
339.         device = (CRemoteDevice) _DevicesFound.GetAt(m_RemoteNameCounter);
340.         AfxMessageBox("Request to get remoteName \nCommand:HCI_ReqRemoteName
            (10,device.tAddress,device.tPageScanPeriodMode,
            device.tPageScanMode,device.tClockOffset )");
341.         HCI_ReqRemoteName(10,
342.             device.tAddress,

```

```

343.             device.tPageScanPeriodMode,
344.             device.tPageScanMode,
345.             device.tClockOffset );
346.     }
347.     else
348.     {
349.         m_DeviceList.AddString(_T("No Devices found"));
350.     }
351. }
352. void CHCIInformationCommandsDlg::OnHciRemoteNameCnf(void **ppMsg)
353. {
354.     HCI_TRemoteNameCnf *ptRemoteNameCnf;
355.     CRemoteDevice device;
356.     char sName[248];
357.     int count;
358.     AfxMessageBox("Returned RemoteDevice
                        BD_ADDRESS\nEvent:HCI_REMOTE_NAME_CNF");
359.     ptRemoteNameCnf =(HCI_TRemoteNameCnf *) *ppMsg;
360.     sprintf(sName,"%s",&ptRemoteNameCnf->tName);
361.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)sName);
362.     m_RemoteNameCounter++;
363.     count = m_DevicesFound.GetSize();
364.     if (count > m_RemoteNameCounter)
365.     {
366.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
367.         HCI_ReqRemoteName(0,
368.             device.tAddress,
369.             device.tPageScanPeriodMode,
370.             device.tPageScanMode,
371.             device.tClockOffset );
372.     }
373.     else
374.     {
375.         ShowAllDevicesFound();
376.     }
377. }
378. void CHCIInformationCommandsDlg::OnHciRemoteNameCnfNeg(void **ppMsg)
379. {
380.     HCI_TRemoteNameCnfNeg *ptRemoteNameCnfNeg;
381.     CRemoteDevice device;
382.     int count;
383.     ptRemoteNameCnfNeg =(HCI_TRemoteNameCnfNeg *) *ppMsg;
384.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)_T("UNKNOWN"));
385.     m_RemoteNameCounter++;
386.     count = m_DevicesFound.GetSize();
387.     if (count > m_RemoteNameCounter)
388.     {
389.         device = (CRemoteDevice)m_DevicesFound.GetAt(m_RemoteNameCounter);
390.         HCI_ReqRemoteName(0,
391.             device.tAddress,
392.             device.tPageScanPeriodMode,
393.             device.tPageScanMode,
394.             device.tClockOffset);
395.     }
396.     else
397.     {

```

```

398.         ShowAllDevicesFound();
399.     }
400. }
401. void CHCIIInformationCommandsDlg::OnHciInquiryEvt(void **ppMsg)
402. {
403.     HCI_TInquiryEvt *ptInquiryEvt;
404.     CRemoteDevice device;
405.     ptInquiryEvt = (HCI_TInquiryEvt *) *ppMsg;
406.     device.tAddress = ptInquiryEvt->tAddress;
407.     device.tPageScanMode = ptInquiryEvt->tPageScanMode;
408.     device.tPageScanPeriodMode = ptInquiryEvt->tPageScanPeriodMode;
409.     device.tClockOffset = ptInquiryEvt->tClockOffset;
410.     device.tCod = ptInquiryEvt->tCod;
411.     device.tPageScanRepMode = ptInquiryEvt->tPageScanRepMode;
412.     AddDevice(device);
413. }
414.
415. void CHCIIInformationCommandsDlg::OnHciStartCnf(void **ppMsg)
416. {
417.     HCI_TStartCnf *ptStartCnf = (HCI_TStartCnf *) *ppMsg;
418.     ptStartCnf = ptStartCnf;
419.     HCI_ReqConfigurePort(0, PORTSETTINGS);
420. }
421. void CHCIIInformationCommandsDlg::OnButton1()
422. {
423.     AfxMessageBox("Request to select INTERFACE
        \nCommand:SIL_SetDevice(0,SIL_SERIAL)");
        SIL_SetDevice(0,SIL_SERIAL);
424. }
425.
426. void CHCIIInformationCommandsDlg::OnButton2()
427. {
428.     AfxMessageBox("Request to read local device
        version\nCommand:HCI_ReqReadLocalVersion(0)");
429.     HCI_ReqReadLocalVersion(0);
430. }
431. void CHCIIInformationCommandsDlg::OnHciReadLocalVersionCnf(void **ppMsg)
432. {
433.     HCI_TReadLocalVersionCnf *tver = (HCI_TReadLocalVersionCnf
        *) *ppMsg;
434.     AfxMessageBox("Returned Local
        Version\nEvent:HCI_READ_LOCAL_VERSION_CNF");
435.     CString str;
436.     str.Format("Version: %d",tver->tHciVersion);
437.     m_ver.SetWindowText(str);
438. }
439. void CHCIIInformationCommandsDlg::OnHciReadLocalVersionCnfNeg(void
        **ppMsg)
440. {
441.     AfxMessageBox("Read Local version
        Failed:\nEvent:HCI_READ_LOCAL_VERSION_CNF_NEG");
442. }
443. void CHCIIInformationCommandsDlg::OnButton3()
444. {
445.     AfxMessageBox("HCI_ReqWriteLoopbackMode(1,HCI_LOOPBACK_NONE)");
446.     HCI_ReqWriteLoopbackMode(1, HCI_LOOPBACK_LOCAL);

```

```

447. }
448. void CHCIInformationCommandsDlg::OnHciWriteLoopbackModeCnf(void **ppMsg)
449. {
450.     AfxMessageBox("Local LoopBack Success");
451.     MSG_TControlHdr *thdr = (MSG_TControlHdr *)*ppMsg;
452.     SeqNr = thdr->uiSeqNr;
453. }
454. void CHCIInformationCommandsDlg::OnHciWriteLoopbackModeCnfNeg(void
                                     **ppMsg)
455. {
456.     AfxMessageBox("Failed to set Local LoopBack");
457. }
458. void CHCIInformationCommandsDlg::OnHciDataInfoCnf(void **ppMsg)
459. {
460.     HCI_TDataInfoCnf *info = (HCI_TDataInfoCnf *)*ppMsg;
461.     AfxMessageBox("Returned Packet Details\nEvent:HCI_DATA_INFO_CNf");
462.     CString str;
463.     str.Format("%d",info->tMaxAclPacketSize);
464.     m_AclSize.SetWindowText(str);
465.     str.Format("%d",info->tMaxAclPackets);
466.     m_AclCount.SetWindowText(str);
467.     str.Format("%d",info->tMaxScoPacketSize);
468.     m_ScoSize.SetWindowText(str);
469.     str.Format("%d",info->tMaxScoPackets);
470.     m_ScoCount.SetWindowText(str);
471. }
472. void CHCIInformationCommandsDlg::OnButton5()
473. {
474.     AfxMessageBox("Request to get supported packet details\n
                     Command:HCI_ReqDataInfo(1)");
475.     HCI_ReqDataInfo(1);
476. }
477. void CHCIInformationCommandsDlg::OnButton7()
478. {
479.     OnInquiry();
480. }

```

Code Description

Note that the VC++ MFC application wizard automatically generates some portions of the code, particularly Lines 13–17, 24–73, 124–143, 148–181, and 183–186.

- ◆ Lines 18–22: Declaration of a union containing AEX_MSG_CALL function call and pfn, pointer to the function.
- ◆ Line 23: The PORTSETTINGS is a constant in which the serial port parameters are defined.
 - COM1: The serial port address.
 - Baud=57600: Transmission speed is 57600 bps.
 - parity=N: parity is NONE.
 - data=8: 8 data bits.
 - stop=1: 1 stop bit.
- ◆ Line 74: m_pServerEvents is an instance of Events Class.
- ◆ Lines 77–87: The DDX_Control functions manage data transfer between dialog box controls and CWnd data members of the dialog box. See the following table for more details.

<i>Dialog box control</i>	<i>CWnd Data member</i>
IDC_EDIT1	m_add
IDC_LIST1	m_DeviceList
IDC_EDIT8	m_ScoCount
IDC_EDIT7	m_ScoSize
IDC_EDIT6	m_AclCount
IDC_EDIT5	m_AclSize
IDC_EDIT2	m_ver

- ◆ Lines 91–122: The `ON_BLUETOOTH_EVENT` message map macro indicates which function will handle a specified BLUETOOTH event. See the following table for more details.

<i>BLUETOOTH Event</i>	<i>Handler Function Name</i>
COM_START_CNF	OnComStartCnf
COM_START_CNF_NEG	OnComStartCnfNeg
COM_VERSION_CNF	OnComVersionCnf
HCI_CONFIGURE_PORT_CNF	OnHciConfigurePortConfirm
HCI_CONFIGURE_PORT_CNF_NEG	OnHciConfigurePortConfirm Negative
HCI_INQUIRY_CNF	OnHciInquiryCnf
HCI_INQUIRY_EVT	OnHciInquiryEvt
HCI_LOCAL_ADDRESS_CNF	OnHciLocalAddressCnf
HCI_LOCAL_ADDRESS_CNF_NEG	OnHciLocalAddressCnfNeg
HCI_REMOTE_NAME_CNF	OnHciRemoteNameCnf
HCI_REMOTE_NAME_CNF_NEG	OnHciRemoteNameCnfNeg
HCI_START_CNF	OnHciStartCnf
HCI_READ_LOCAL_VERSION_CNF	OnHciReadLocalVersionCnf
HCI_READ_LOCAL_VERSION_CNF_NEG	OnHciReadLocalVersionCnfNeg
HCI_WRITE_LOOPBACK_MODE_CNF	OnHciWriteLoopbackModeCnf
HCI_WRITE_LOOPBACK_MODE_CNF_NEG	nHciWriteLoopbackModeCnfNeg
HCI_DATA_INFO_CNF	OnHciDataInfoCnf
SIL_SET_DEVICE_CNF	OnSilSetDeviceCnf
SIL_SET_DEVICE_CNF_NEG	OnSilSetDeviceCnfNeg

- ◆ Lines 115–119: The `ON_BN_CLICKED` message map macro indicates which function will handle a specified button click. Each button has a unique ID and function call associated with it (see the following table).

<i>Button ID</i>	<i>Function Name</i>
IDC_BUTTON1	OnButton1
IDC_BUTTON2	OnButton2

IDC_BUTTON3	OnButton3
IDC_BUTTON5	OnButton5
IDC_BUTTON7	OnButton7

- ◆ Line 144: The current `CHCIInformationCommandsDlg` dialog box is initialized to an `Events` class member called `m_pParentDialog` to receive all outgoing events generated by `BT_COMServer`.
- ◆ Lines 187–202: `WindowProc(UINT message, WPARAM wParam, LPARAM lParam)` is a virtual function, which provides a Windows procedure to dispatch BLUETOOTH events to the `CHCIInformationCommandsDlg` Dialog box through the Windows message map. The following are the parameters associated with this function:
 - `UINT message`: Specifies the Windows message to be processed.
 - `WPARAM wParam`: Additional information.
 - `LPARAM lParam`: Additional information.
- ◆ Line 192: Condition to check whether the received message is a BLUETOOTH event.
- ◆ Line 195: If the event is a BLUETOOTH event, it invokes `OnBluetoothEvent (message, wParam, lParam)` function.
- ◆ Line 199: To free the message received from the `BT_COMServer`, the function `VOS_Free((void **)lParam)` is used.
- ◆ Lines 203–240: `OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam)` is a member function of `CHCIInformationCommandsDlg` Class. This function checks whether the received message is a mapped BLUETOOTH event or not. If it is a mapped BLUETOOTH event, it calls the corresponding message-map function. Unmapped BLUETOOTH events are ignored by the application.
- ◆ Lines 241–248: `HCI_ReqInquiry(1, tLap, tInquiryLength, tNrOfResponses)` is a BLUETOOTH command to discover the nearby remote BLUETOOTH devices. The parameters are as follows:
 - Parameter1: sequence number=1 (the response for this request should also have the same sequence number).
 - Parameter2: Lower address part = {0x9E, 0x8B, 0x33}
 - Parameter3: Maximum amount of time specified before the Inquiry is halted = 2 seconds
 - Parameter4: Maximum number of responses from the Inquiry = 0 (0 indicates that it allows unlimited number of responses)

A message box appears to indicate that the command has been issued.

`HCI_ReqInquiry(1, tLap, tInquiryLength, tNrOfResponses)` causes the BLUETOOTH module to send either `HCI_INQUIRY_CNF` or `HCI_INQUIRY_CNF_NEG` event corresponding to success or failure.

- ◆ Lines 269–275: `SIL_SET_DEVICE_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnSilSetDeviceCnf(void **ppMsg)` message-handler function. A message box indicates that the serial interface selection is a success.
- `HCI_ReqConfigurePort(0, PORTSETTINGS)` is a BLUETOOTH SDK API to send a command to configure the serial port with the settings Defined in `PORTSETTINGS`. The parameters are as follows:
 - Parameter1: sequence number = 0
 - Parameter2: port settings = `PORTSETTINGS`

`HCI_ReqConfigurePort(0, PORTSETTINGS)` causes the BLUETOOTH module to fire either `HCI_CONFIGURE_PORT_CNF` or `HCI_CONFIGURE_PORT_CNF_NEG` event corresponding to success or failure. A message box appears to let the user know that the command has been issued.

- ◆ Lines 276–279: `SIL_SET_DEVICE_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnSilSetDeviceCnfNeg(void **ppMsg)` message-handler function. A message box indicates that the serial interface selection has failed.
- ◆ Lines 280–287: `HCI_CONFIGURE_PORT_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciConfigurePortConfirm(void **ppMsg)` message-handler function. A message box displays that the port configuration is a success.
 - `COM_ReqStart(0)` is a BLUETOOTH SDK API to send a command to start RFCOMM service on BLUETOOTH stack.
 - Parameter1: sequence number = 0
 - `COM_ReqStart(0)` causes the BLUETOOTH module to fire either a `COM_START_CNF` or `COM_START_CNF_NEG` event corresponding to success or failure. A message box shows that the command has been issued.
- ◆ Lines 288–292: `HCI_CONFIGURE_PORT_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciConfigurePortConfirmNegative(void **ppMsg)` message handler function. A message box indicates that the port configuration has failed.
- ◆ Lines 294–301: `COM_START_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnComStartCnf(void **ppMsg)` message handler function. A message box indicates that RFCOMM component has been started.
 - `HCI_ReqLocalAddress(0)` is a BLUETOOTH command to request Local BLUETOOTH Device address.
 - Parameter1: sequence number = 0
 - `HCI_ReqLocalAddress(0)` causes the BLUETOOTH module to fire either `HCI_LOCAL_ADDRESS_CNF` or `HCI_LOCAL_ADDRESS_CNF_NEG` event corresponding to success or failure. A message box shows that the command has been issued.
- ◆ Lines 302–307: `COM_START_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnComStartCnfNeg(void **ppMsg)` message-handler function. A message box indicates that the RFCOMM could not start.
- ◆ Lines 308–322: `HCI_LOCAL_ADDRESS_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciLocalAddressCnf(void **ppMsg)` message handler function. A message box indicates that the local BLUETOOTH device address has been retrieved.
 - `tLocalAddress` is a structure of type `HCI_TLocalAddressCnf` that contains the local BLUETOOTH device address `tAddress` as its member. This address is formatted to be displayed in the Edit control.
- ◆ Lines 323–327: `HCI_LOCAL_ADDRESS_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciLocalAddressCnfNeg(void **ppMsg)` message-handler function. A message box indicates that the retrieval of the local BLUETOOTH Device address failed.
- ◆ Lines 328–351: `HCI_INQUIRY_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciInquiryCnf(void **ppMsg)` message-handler function. A message box indicates that the inquiry was sent to nearby BLUETOOTH radios successfully.

- `m_DevicesFound.GetSize()` returns the number of nearby BLUETOOTH devices. If the number of BLUETOOTH devices is greater than zero, it gets one of the remote devices into device structure. The device structure contains members, which are used to request information about remote BLUETOOTH device.
- `HCI_ReqRemoteName (0,`
`device.tAddress,`
`device.tPageScanPeriodMode,`
`device.tPageScanMode,`
`device.tClockOffset);`

is a BLUETOOTH command to request remote BLUETOOTH device address.

- ◆ The parameters are necessary when there is no existing connection between local BLUETOOTH device and the remote BLUETOOTH device.

- Parameter1: sequence number=0
- Parameter2 : address variable to hold the remote BLUETOOTH device address
- Parameter3: page scan period mode
- Parameter4: page scan mode
- Parameter5: estimation of clock offset of the device

It causes the BLUETOOTH module to fire either `HCI_REMOTE_NAME_CNF` or `HCI_REMOTE_NAME_CNF_NEG` event corresponding to success or failure. A message box shows that the command has been issued.

- ◆ Lines 352–377: `HCI_REMOTE_NAME_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciRemoteNameCnf(void **ppMsg)` message-handler function. A message box indicates that the name of a remote BLUETOOTH device has been retrieved successfully.

- The `HCI_ReqRemoteName (0,`
`device.tAddress,`
`device.tPageScanPeriodMode,`
`device.tPageScanMode,`
`device.tClockOffset);`

method is called until all the remote BLUETOOTH device addresses are retrieved. As soon as the nearby BLUETOOTH device addresses are retrieved, the method `ShowAllDevices()` is used to display to the user. Message boxes will appear whenever a remote BLUETOOTH device address is returned.

- ◆ Lines 378–400: `HCI_REMOTE_NAME_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciRemoteNameCnfNeg(void **ppMsg)` message handler function. A message box indicates that the retrieval of the remote BLUETOOTH device name has failed.

- The `HCI_ReqRemoteName (0,`
`device.tAddress,`
`device.tPageScanPeriodMode,`
`device.tPageScanMode,`
`device.tClockOffset);`

method is called to get the next remote BLUETOOTH device address.

- ◆ Lines 401–420: When an inquiry is generated by a Bluetooth device, `ONHCI_INQUIRY_EVENT` is generated and `OnHciInquiryEvt` function is called.
- ◆ Lines 421–424: When `IDC_BUTTON1` is clicked, the corresponding message map function `OnButton1` will be called.

- `SIL_SetDevice(0, SIL_SERIAL)` is a BLUETOOTH SDK API to send a command to select the serial interface on the BLUETOOTH module.
 - Parameter1: sequence number of the interface = 0
 - Parameter2: type of interface = `SIL_SERIAL`
 - `SIL_SetDevice(0, SIL_SERIAL)` causes the BLUETOOTH module to fire either `SIL_SET_DEVICE_CNF` or `SIL_SET_DEVICE_CNF_NEG` event corresponding to success or failure. A message box appears to let the user know that the command has been issued.
- ◆ Lines 426–430: When `IDC_BUTTON2` is clicked, the corresponding message-map function `OnButton2` will be called.
- `HCI_ReqReadLocalVersion(0)` is a BLUETOOTH command to get the version of the BLUETOOTH module.
 - Parameter1: sequence number = 0
 - `HCI_ReqReadLocalVersion(0)` causes the BLUETOOTH module to fire either `HCI_READ_LOCAL_VERSION_CNF` or `HCI_READ_LOCAL_VERSION_CNF_NEG` event corresponding to success or failure. A message box will be displayed to indicate that the command has been issued.
 - `tver` is a structure of type `HCI_TReadLocalVersionCnf` which contains local BLUETOOTH device version `tHciVersion` as its member. This version number is formatted to be displayed in the Edit control.
- ◆ Lines 431–438: `HCI_READ_LOCAL_VERSION_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciReadLocalVersionCnf(void **ppMsg)` message handler function. A message box indicates that the local BLUETOOTH device version has been retrieved.
- ◆ Lines 439–442: `HCI_READ_LOCAL_VERSION_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciReadLocalVersionCnfNeg(void **ppMsg)` message handler function. A message box indicates that the local BLUETOOTH device version retrieval failed.
- ◆ Lines 443–447: When `IDC_BUTTON3` is clicked, the corresponding message map function `OnButton3` is called.
- `HCI_ReqWriteLoopbackMode(1, HCI_LOOPBACK_LOCAL)` is a BLUETOOTH command to set the BLUETOOTH module in local loop back mode.
 - Parameter1: sequence number = 1
 - Parameter2: Local loop back mode
 - `HCI_ReqWriteLoopbackMode(1, HCI_LOOPBACK_LOCAL)` causes the BLUETOOTH module to fire either an `HCI_LOOPBACK_LOCAL_CNF` or `HCI_LOOPBACK_LOCAL_CNF_NEG` event corresponding to success or failure. A message box appears to show the user that the command has been issued.
- ◆ Lines 448–453: `HCI_WRITE_LOOPBACK_MODE_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciWriteLoopbackModeCnf(void **ppMsg)` message-handler function. A message box indicates that local loop back has been set.
- ◆ Lines 454–457: `HCI_WRITE_LOOPBACK_MODE_CNF_NEG` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciWriteLoopbackModeCnfNeg(void **ppMsg)` message-handler function. A message box indicates that local loop back setting has failed.
- ◆ Lines 458–471: `HCI_DATA_INFO_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnHciDataInfoCnf(void **ppMsg)` message-handler function. A message box indicates that the local BLUETOOTH device supported packet information has been retrieved.

Info is a structure of type `HCI_TdataInfoCnf`. Info contains maximum size of ACL packet, maximum number of ACL packets, maximum size of SCO packet and maximum number of SCO packets as its members. The packet information is formatted to be displayed in Edit controls.

- ◆ Lines 472–476: When `IDC_BUTTON5` is clicked, the corresponding message-handler function `OnButton5` is called.
 - `HCI_ReqDataInfo(1)` is a BLUETOOTH command to get the supported packet information of the BLUETOOTH module. The packet information contains ACL packet details and as well as SCO packet details.
 - Parameter1: sequence number = 1
 - `HCI_ReqDataInfo(1)` causes the BLUETOOTH module to fire either `HCI_DATA_INFO_CNF` or `HCI_DATA_INFO_CNF_NEG` event corresponding to success or failure. A message box appears to show the user that the command has been issued.
- ◆ Lines 477–480: After the `IDC_BUTTON7` is clicked, the corresponding message handler function `OnButton7` is called. This `OnButton7()` in turn calls the function `OnInquiry()`.

Listing 9-3 is a header file for constant definitions and class declaration.

Listing 9-3: RemoteDevice.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. // RemoteDevice.h: interface for the CRemoteDevice class.
2. //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3. #if !defined(AFX_DEVICE_H__E7DA5337_C36C_11D3_B6C6_0060082D5EF1
   __INCLUDED_)
4. #define AFX_DEVICE_H__E7DA5337_C36C_11D3_B6C6_0060082D5EF1__
   INCLUDED_
5. #if _MSC_VER > 1000
6. #pragma once
7. #endif // _MSC_VER > 1000
8. #include <exp/bt.h>
9. #include <exp/hci.h>
10. class CRemoteDevice
11. {
12. public:
13.     CRemoteDevice();
14.     CRemoteDevice(CString sAddress,CString sName);
15.     virtual ~CRemoteDevice();
16.         void SetAddress (CString sAddress);
17.         void SetName(CString sName);
18.         CString GetAddress ();
19.         CString GetName();
20.         BT_TAddress          tAddress;
21.         uint8                tPageScanRepMode;
22.         uint8                tPageScanPeriodMode;
23.         uint8                tPageScanMode;
24.         HCI_TCod             tCod;
25.         uint16               tClockOffset;
26. private:
27.         CString m_Address;
28.         CString m_Name;
29. };
30. #endif

```

Code Description

Lines 1 to 7 are automatically generated; Lines 8 and 9 are for including the necessary Bluetooth header files, and Lines 10 to 30 are for CRemoteDevice class declaration with public and private members. The various methods and variables used are explained in the Listing 9-4, which gives the source code for RemoteDevice.cpp.

Listing 9-4: RemoteDevice.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. //RemoteDevice.cpp: implementation of the CRemoteDevice class.
2. // ////////////////////////////////////// 3.
   #include "stdafx.h"
4. #include "HCI Information Commands.h"
5. #include "RemoteDevice.h"
6. #ifdef _DEBUG
7. #undef THIS_FILE
8. static char THIS_FILE[]=__FILE__;
9. #define new DEBUG_NEW
10. #endif
11.
12. CRemoteDevice::CRemoteDevice()
13. {
14.     m_Address.Empty();
15.     m_Name.Empty();
16. }
17. CRemoteDevice::CRemoteDevice(CString sAddress,CString sName)
18. {
19.     m_Address = sAddress;
20.     m_Name = sName;
21. }
22. CRemoteDevice::~CRemoteDevice()
23. {
24. }
25. void CRemoteDevice::SetAddress (CString sAddress)
26. {
27.     m_Address = sAddress;
28. }
29. void CRemoteDevice::SetName(CString sName)
30. {
31.     m_Name = sName;
32. }
33. CString CRemoteDevice::GetAddress ()
34. {
35.     char s[80];
36.     CString sAddress;
37.     sprintf(s,"%02X%02X%02X%02X%02X",
38.         tAddress.ucByte0,tAddress.ucByte1,tAddress.ucByte2,
39.         tAddress.ucByte3, tAddress.ucByte4, tAddress.ucByte5);
40.     return (CString) s;//sAddress;
41. }
42. CString CRemoteDevice::GetName()
43. {
44.     return m_Name;
45. }

```

Code Description

In Listing 9-4, lines 1–10 are automatically generated by the MFC application wizard. Explanations for the rest of the code are as follows:

- ◆ Lines 12–16: `CRemoteDevice` constructor is defined and used to free the content stored in variables `m_Address` and `m_Name` by applying the `Empty()` method.
- ◆ Lines 17–21: `CRemoteDevice` constructor is overloaded to initialize the remote BLUETOOTH device address and its name.
- ◆ Lines 22–23: `CRemoteDevice` destructor is defined to destruct the class.
- ◆ Lines 25–28: `SetAddress (CString sAddress)` member function is defined to initialize BLUETOOTH device address to a data member of `CRemoteDevice` class.
- ◆ Lines 29–32: `SetName(CString sName)` member function is defined to initialize BLUETOOTH device name to a data member of `CRemoteDevice` class.
- ◆ Lines 33–41: `GetAddress()` member function is defined to return the BLUETOOTH device address as formatted string.
- ◆ Lines 42–45: `GetName()` member function is defined to return the BLUETOOTH device name in the form of a data member of `CRemoteDevice` class.

Listing 9-5 is the header file for the necessary declarations for the COM components.

Listing 9-5: Events.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. class CHCIInformationCommandsDlg;
2. class Events : public CCmdTarget
3. {
4.     DECLARE_DYNCREATE(Events)
5. public:
6.     Events();
7.     virtual ~Events();
8. public:
9.     CHCIInformationCommandsDlg *m_pParentDialog;
10. public:
11.     virtual void OnFinalRelease();
12.     protected:
13.         IConnectionPointContainer *m_pConnectionPointContainer;
14.         IConnectionPoint *m_pConnectionPoint;
15.         IVOSProcess *m_pVOSProcess;
16.         IVOSProcess *m_pVOSProcessInterface;
17.         DWORD m_ConnectionPointID;
18.     DECLARE_MESSAGE_MAP()
19.     afx_msg SCODE MsgReceived(VARIANT FAR *pvMsg, LONG SenderID);
20.     //}}AFX_DISPATCH
21.     DECLARE_DISPATCH_MAP()
22.     DECLARE_INTERFACE_MAP()
23. private:
24. };
25. #endif

```


Code Description

For the Bluetooth PC reference stack to access the COM component, the necessary COM server component variables are declared in Listing 9-5. The use of the variables is evident in the Listing 9-6, which gives the source code for `Events.cpp`.

Listing 9-6: Events.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. // Events.cpp : implementation file
2. #include "stdafx.h"
3. #include "HCI Information Commands.h"
4. #include "HCI Information CommandsDlg.h"
5. #include "events.h"
6. #include "WinError.h"
7. #include <exp\vos2com.h>
8. #include <exp\SafeArray.h>
9. #ifdef _DEBUG
10. #define new DEBUG_NEW
11. #undef THIS_FILE
12. static char THIS_FILE[] = __FILE__;
13. #endif
14. CLSID clsid;
15. extern IVOSProcess* Server; /* variable declared in vos2com.h */
16.
17. IMPLEMENT_DYNCREATE(Events, CCmdTarget)
18.
19. Events::Events()
20. {
21.     // make a connection to the BT_COMServer
22.     EnableAutomation();
23.     m_pVOSProcess = NULL;
24.     clsid = CLSID_VOSProcess;
25.     CoInitialize(NULL);
26.     // first make contact to the BT_COMServer
27.     HRESULT hr=CoCreateInstance
        (CLSID_VOSProcess, NULL, CLSCTX_LOCAL_SERVER, IID_IVOSProcess,
        (void **)&m_pVOSProcess);
28.     // Get interface form BT_COMServer
29.     if (SUCCEEDED(hr))
30.         hr=m_pVOSProcess->QueryInterface(IID_IVOSProcess,
        (void **)&m_pVOSProcessInterface);
31.     else
32.     {
33.         MessageBox(NULL, _T("No Instance
        Created"), NULL, MB_OK);
34.         exit(0);
35.     }
36.     if (FAILED(hr))
37.     {
38.         MessageBox(NULL, _T("No VosProcess
        Supported"), NULL, MB_OK);
39.         exit(0);
40.     }
41.     //Get Connection point container for BT_COMServer

```

```

42. hr=m_pVOSProcess->
    QueryInterface(IID_IConnectionPointContainer, (void
    **) &m_pConnectionPointContainer);
43. if (SUCCEEDED(hr))
44. {
45.     // then get connection point
46.     hr=m_pConnectionPointContainer->
        FindConnectionPoint(DIID__IVOSProcessEvents, &m_pConnectionPoint);
47.     m_pConnectionPointContainer->Release();
48.     if (SUCCEEDED(hr))
49.     { // Pointer to the client's advise sink ,
        m_ServerEvents will receive all
50.         // outgoing events from the BT_COMServer
51.         LPDISPATCH lpServerDispatch = GetIDispatch(TRUE);
52.         hr=m_pConnectionPoint->Advise(lpServerDispatch,
        &m_ConnectionPointID);
53.         if ((hr!=S_OK) || (m_ConnectionPointID==0))
54.         {
55.             switch (hr)
56.             {
57.                 case E_POINTER :
58.                     MessageBox(NULL, _T("The value in pUnk or pdwCookie is
                        not valid. For example, either pointer may be
                        NULL."), NULL, MB_OK);
59.                     break;
60.                 case CONNECT_E_ADVISELIMIT :
61.                     MessageBox(NULL, _T("The connection point has already
                        reached its limit of connections and cannot accept any
                        more."), NULL, MB_OK);
62.                     break;
63.                 case CONNECT_E_CANNOTCONNECT :
64.                     MessageBox(NULL, _T("The sink does not support the
                        interface required by this connection point."), NULL, MB_OK);
65.                     break;
66.                 default:
67.                     MessageBox(NULL, _T("error not defined"), NULL, MB_OK);
68.                     break;
69.             }
70.             MessageBox(NULL, _T("Advise
                failed"), NULL, MB_OK);
71.             exit(0);
72.         }
73.     }
74.     else
75.     {
76.         MessageBox(NULL, _T("No connection
                point"), NULL, MB_OK);
77.         exit(0);
78.     }
79. }
80. else
81. {
82.     MessageBox(NULL, _T("No IConnectionPointContainer
                supported"), NULL, MB_OK);
83.     exit(0);
84. }

```

```

85.
86. if (m_pVOSProcess)
87. {
88.     //Access to Server functions
89.     Server = m_pVOSProcess;
90.     InitVOSProcesses();
91.
92. }
93. }
94. Events::~Events()
95. {
96.     if (m_pConnectionPoint)
97.         m_pConnectionPoint->Unadvise(m_ConnectionPointID);
98.     m_pVOSProcessInterface->Release();
99.     m_pConnectionPointContainer->Release();
100.    m_pConnectionPoint->Release();
101. }
102. void Events::OnFinalRelease()
103. {
104.     // When the last reference for an automation object is
        released
105.     // OnFinalRelease is called. The base class will
        automatically
106.     // deletes the object. Add additional cleanup required for your
107.     // object before calling the base class.
108.
109.     CCmdTarget::OnFinalRelease();
110. }
111. BEGIN_MESSAGE_MAP(Events, CCmdTarget)
112.     //{AFX_MSG_MAP(Events)
113.     // NOTE - the ClassWizard will add and remove mapping
        macros here.
114.     //}}AFX_MSG_MAP
115. END_MESSAGE_MAP()
116. BEGIN_DISPATCH_MAP(Events, CCmdTarget)
117.     //{AFX_DISPATCH_MAP(Events)
118.     DISP_FUNCTION(Events, "method MsgReceived", MsgReceived,
        VT_ERROR, VTS_VARIANT VTS_I4 )
119.     //}}AFX_DISPATCH_MAP
120. END_DISPATCH_MAP()
121.
122.
123.
124.
125. static const IID IID_IServerEvents =
126. { 0x425E3D83, 0x7714, 0x11D3, { 0x98, 0xB0, 0x00, 0x90, 0x27,
        0x1C, 0x90, 0x35 } };
127. BEGIN_INTERFACE_MAP(Events, CCmdTarget)
128.     INTERFACE_PART(Events, IID_IServerEvents, Dispatch)
129. END_INTERFACE_MAP()
130. SCODE Events::MsgReceived(VARIANT FAR *pvMsg, LONG SenderID)
131. {
132.     // TODO: Add your dispatch handler code here
133.     MSG_TMsg *ptMsg;
134.     tMemHeader *ptDummy;
135.     SAFEARRAY FAR* psaMsg;

```

```

136. //Retrieve SafeArray from the variant type
137. psaMsg = (SAFEARRAY FAR*) V_ARRAY(pvMsg);
138. // convert the SafeArray to Bluetooth Stack message
139. ptMsg = (MSG_TMsg *)SafeArrayToMsg(psaMsg);
140. // Gerrit: Sender filling added.
141. // Calculate beginning of memory header
142. ptDummy = (tMemHeader*)((char*)ptMsg-(sizeof(tMemHeader)-
    sizeof(void *)));
143. ptDummy->tSender = (VOS_TProcess) SenderID;
144. SendMessage(m_pParentDialog->m_hWnd,
    WM_BLUETOOTH_EVENT, (WPARAM) ptMsg->tHdr.tID, (LPARAM)
    &ptMsg);
145. return S_OK;
146. }

```

Code Description

In Listing 9-6, the lines 9–13, 17, and 111–120 are automatically generated by the class wizard. Explanations for the remaining code are as follows.

- ◆ Lines 14–16: Variables declaration. `clsid` is the ID of the COM component.
- ◆ Line 22: `EnableAutomation()` function is called to enable OLE automation for the `BT_COMSERVER` component.
- ◆ Line 25: the `CoInitialize(NULL);` function is used to initialize the COM library.
- ◆ Line 27: This function is called to get an instance of `BT_COMSERVER`.
- ◆ Lines 29 and 30: This function is called to get the interface associated with the `BT_COMSERVER`.
- ◆ Lines 31–45: If creation of COM component instance creation fails, a series of messages are displayed; a switch statement is used to take care of different cases.
- ◆ Line 46: Gets connection point for `BT_COMSERVER`.
- ◆ Lines 47–88: This code generates diagnostic messages in the form of message boxes.
- ◆ Lines 89 and 90: This code is to access all the outgoing events from `BT_COMSERVER`.
- ◆ Lines 94–101: The destructor is defined to release the memory for all pointers.
- ◆ Lines 102–110: `OnFinalRelease` is called when the last reference for an automation Object is released to clean up the objects.
- ◆ Lines 111–120: This code is generated by MFC application wizard.
- ◆ Lines 130–146: `SendMessage(m_pParentDialog->m_hWnd, WM_BLUETOOTH_EVENT, (WPARAM) ptMsg->tHdr.tID, (LPARAM) &ptMsg);` function is used to send events to `CHCIInformationDlg`.

Code Output

After you build the project in the VC++ environment and execute its application, the main window appears, as shown in Figure 9-1.

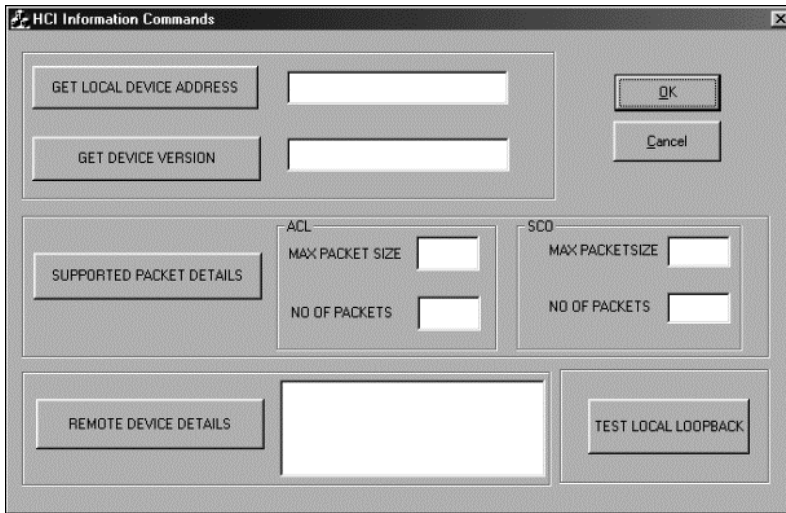


Figure 9-1: Output of HCI Application

The screen in Figure 9-1 contains the following buttons:

- ◆ button1: GET LOCAL DEVICE ADDRESS
- ◆ button2: GET DEVICE VERSION
- ◆ button3: SUPPORTED PACKET DETAILS
- ◆ button4: REMOTE DEVICE DETAILS
- ◆ button5: TEST LOCAL LOOPBACK

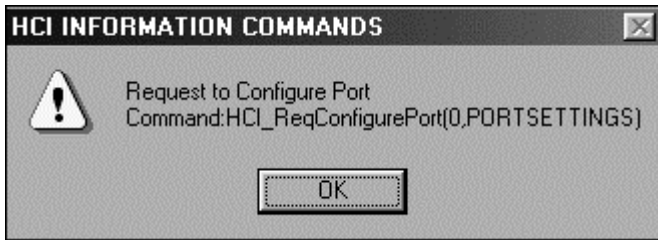
After button1 (GET LOCAL DEVICE ADDRESS) is clicked, the following message box appears. It shows that the command `SIL_SetDevice(0, SIL_SERIAL)` has been issued.



After the OK button in the previous message box is clicked, the following message box appears. It shows that the event `SIL_SET_DEVICE_CNF` has been generated.



After the OK button is clicked on the previous message box, the following message box appears. It shows that the command `HCI_ReqConfigurePort(0, PORTSETTINGS)` has been issued.



After the OK button is clicked, the following message box appears. It shows that the command `HCI_CONFIGURE_PORT_CNF` has been issued.



After you click the OK button, the following message box appears. It shows that the command `COM_ReqStart(0)` has been issued.



After clicking the OK button, the following message box appears. It shows that the command `COM_START_CNF` has been issued.



After clicking the OK button, the following message box appears. It shows that the command `HCI_ReqLocalAddress(0)` has been issued.



After clicking the OK button, the following message box appears. It shows that the command `HCI_LOCAL_ADDRESS_CNF` has been issued.



After you click the OK button, the local device address is displayed in the edit box, as shown in Figure 9-2.

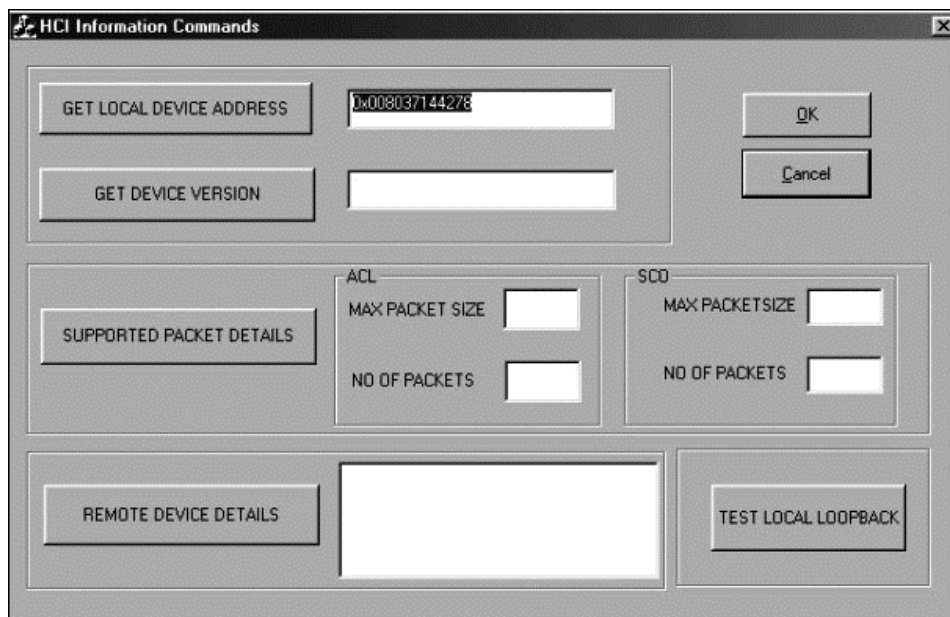


Figure 9-2: Window showing the Local Device Address

After button2 (GET DEVICE VERSION) is clicked, the following message box appears. It shows that the command `HCI_ReqReadLocalVersion(0)` has been issued.



After clicking the OK button, the following message box appears. It shows that the command `HCI_READ_LOCAL_VERSION_CNF` has been issued.



After you click the OK button, local device VERSION is displayed in the edit box, as shown in Figure 9-3.

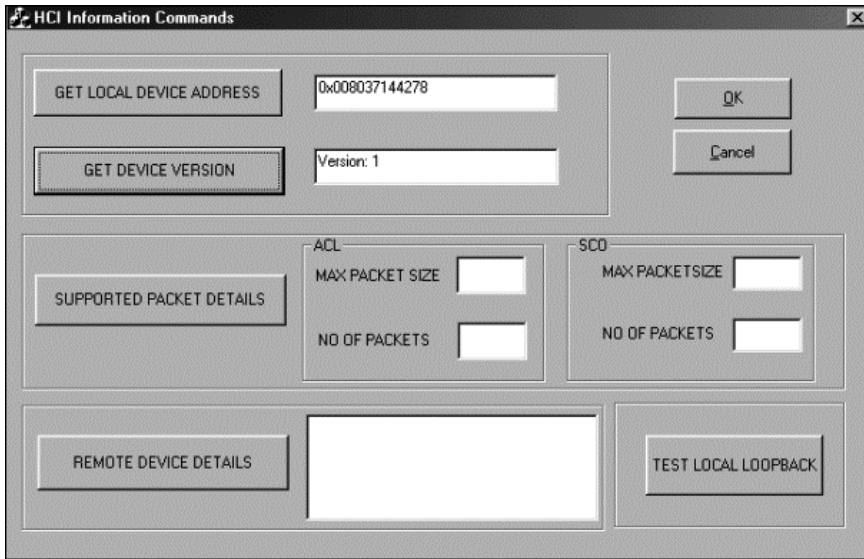


Figure 9-3: Window showing the Device Version

After button3 (SUPPORTED PACKET DETAILS) is clicked, the following message box appears. It shows that the command `HCI_ReqDataInfo(1)` has been issued.



After clicking the OK button, the following message box appears. It shows that the event `HCI_DATA_INFO_CNF` has been returned.



After the OK button is clicked, the supported packet details displays in the corresponding edit boxes, as shown in Figure 9-4.

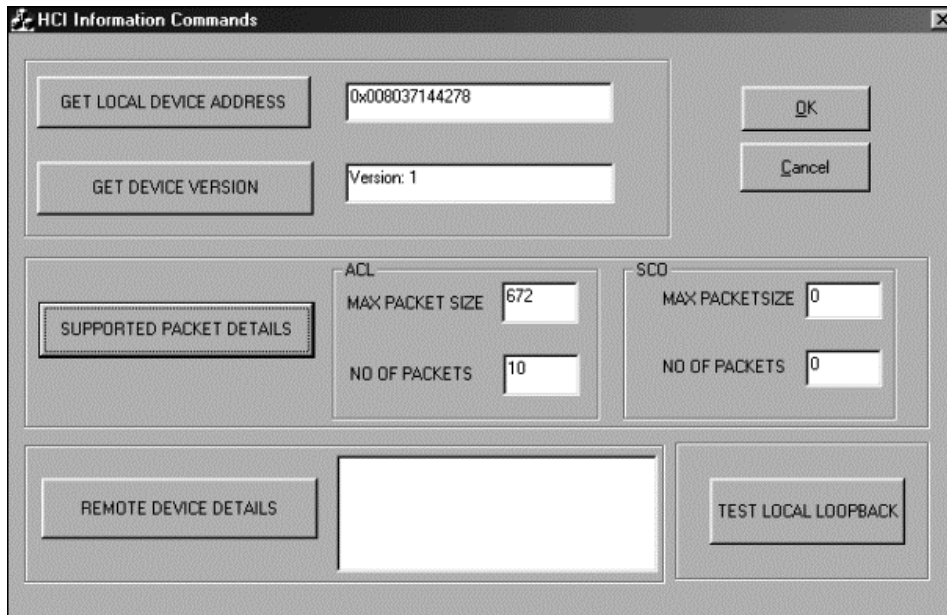


Figure 9-4: Window showing the Supported Packet Details

After button4 (REMOTE DEVICE DETAILS) is clicked, the following message box appears. It shows that the command `HCI_ReqInquiry(1, tLap, tInquiryLength, tNrOfResponses)` has been issued.



After clicking the OK button, the following message box appears. It shows that the event `HCI_INQUIRY_CNF` has been returned.



After clicking the OK button, the following message box appears. It shows that the command `HCI_ReqRemoteName(10, device.tAddress, device.tPageScanPeriodMode, device.tPageScanMode, device.tClockOffset)` has been issued.



After clicking the OK button, the following message box appears. It shows that the event HCI_REMOTE_NAME_CNF has been returned.



After the OK button is clicked, the Remote device address displays in the corresponding edit box, as shown in Figure 9-5.

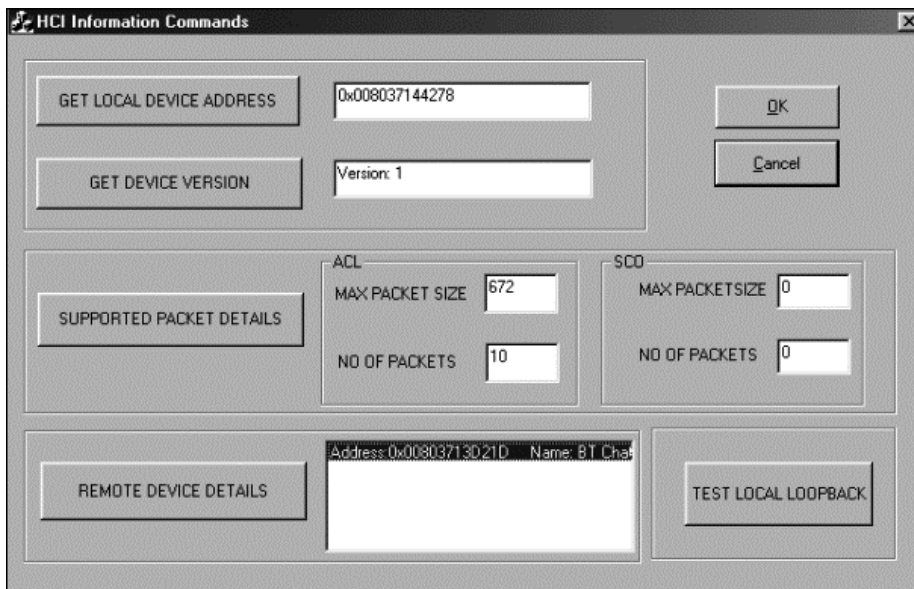


Figure 9-5: Window showing the Remote Device Details

After button5 (TEST LOCAL LOOPBACK) is clicked, the following message box appears on the screen. It indicates that the command HCI_ReqWriteLoopbackMode(1, HCI_LOOPBACK_NONE) has been issued.



After you click the OK button, the following message box appears.



Click OK to return to the main window. Click Cancel on the main window to quit from the application.

Now that we have completed the HCI programming to make the stack talk to the Bluetooth module and also to obtain the information about the remote Bluetooth devices, the next step is to discover the services available on other Bluetooth devices. The Service Discovery Protocol (SDP) is used to achieve this.

Registering and Discovering Services: SDP Programming

This example illustrates how to register a service in a Bluetooth device and how to access the service from other Bluetooth devices. Of the two Bluetooth-enabled PCs, one can provide a print service. This service has to be registered so that the other PC can first discover the service and then use it. We will see how to register a print service. For the service registration, Database Manager, which is a part of the SDP, is used. The software consists of the following five modules:

- ◆ GUI: This module provides the user interface to register a Bluetooth service. For each command generated by the user, the Bluetooth module fires the corresponding Bluetooth event. The user can identify whether the issued command is successful or not by getting the fired event information in the message boxes. This module is implemented with `CSDPInformationCommandsDlg` class in the file named `SDP Information CommandsDlg.cpp`. The corresponding header file is `CommandsDlg.h`.
- ◆ PRINTSERVICE: This module defines a print profile to implement print service. The print profile provides the necessary information about print service to remote Bluetooth devices. This module is implemented with `CPrintProfile` class in the file named `PrintProfile.cpp`. The corresponding header file is `PrintProfile.h`.
- ◆ RS232: This module registers the defined print service in the Database of Bluetooth stack by calling DBM (Database Manager). This module is implemented with `CRS232` class in the file named `RS232.CPP`. The corresponding header file is `RS232.h`.
- ◆ EVENTS: Same function as in HCI.
- ◆ REMOTEDEVICE: Same function as in HCI.

Listing 9-7 gives the source code for the header file `CommandsDlg.h`, in which necessary library files are imported and variables and classes are declared.

Listing 9-7: CommandsDlg.h

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. // SDP Information CommandsDlg.h : header file
2. //
3.
4. #if
    !defined(AFX_SDPINFORMATIONCOMMANDSDLG_H__93104A63_A111_11D2_B76C_0080C805A67
9 INCLUDED_)
5. #define
    AFX_SDPINFORMATIONCOMMANDSDLG_H__93104A63_A111_11D2_B76C_0080C805A679__INCLUD
ED_
6.
7. #if _MSC_VER > 1000
8. #pragma once
9. #endif // _MSC_VER > 1000
10.
11.
12. #include "Events.h"
13. #include "RemoteDevice.h"
14. #include "RS232.h"
15. #include <afxtempl.h>
16.
17.
18.
19. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
20.
21. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
22.     { WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
23.         (AFX_PMSG)(AFX_PMSGW)(void (AFX_MSG_CALL CWnd::*)(void
**))&memberFxn },
24.
25. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
26.     SendMessage((HWND)this_hWnd, WM_BLUETOOTH_EVENT, (WPARAM)uiBtEventID, (LPARAM)
&pMsg)
27.
28.
29.
30. // CSDPInformationCommandsDlg dialog
31.
32. class CSDPInformationCommandsDlg : public CDialog
33. {
34. // Construction
35. public:
36.     void SetDevice();
37.     CSDPInformationCommandsDlg(CWnd* pParent = NULL);
        // standard constructor
38.
39. // Dialog Data
40.     //{AFX_DATA(CSDPInformationCommandsDlg)
41.     enum { IDD = IDD_SDPINFORMATIONCOMMANDS_DIALOG };
42.     // NOTE: the ClassWizard will add data members here
43.     //}AFX_DATA
44.
45. // ClassWizard generated virtual function overrides

```

```

46. //{AFX_VIRTUAL(CSDPInformationCommandsDlg)
47. protected:
48. virtual void DoDataExchange(CDataExchange* pDX);
49. virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);
50. //}AFX_VIRTUAL
51.
52. // Implementation
53. protected:
54. HICON m_hIcon;
55. CArray <CRemoteDevice,CRemoteDevice> m_DevicesFound;
56. Events *m_pServerEvents;
57. CRS232 *m_pSerialPort;
58. int m_RemoteNameCounter;
59. int m_ServiceCounter;
60.
61. uint16 m_RFCommHandle;
62. uint8 m_RFServerChannel;
63.
64. // Generated message map functions
65. //{AFX_MSG(CSDPInformationCommandsDlg)
66. virtual BOOL OnInitDialog();
67. afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
68. afx_msg void OnPaint();
69. afx_msg HCURSOR OnQueryDragIcon();
70. afx_msg void OnStart();
71. afx_msg void OnRegisterService();
72.
73. afx_msg void OnDbmStartCnf(void **ppMsg);
74. afx_msg void OnDbmRegisterCnf(void **ppMsg);
75. afx_msg void OnDbmRegisterCnfNeg(void **ppMsg);
76.
77. //}AFX_MSG
78. DECLARE_MESSAGE_MAP()
79. private:
80. BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
81. };
82.
83. //{AFX_INSERT_LOCATION}}
84. // Microsoft Visual C++ will insert additional declarations immediately
   before the previous line.
85.
86. #endif //
   !defined(AFX_SDPINFORMATIONCOMMANDSDLG_H__93104A63_A111_11D2_B76C_
   0080C805A679_INCLUDED_)

```

Code Description

In Listing 9-7, lines 4–11 are automatically added by the VC++ application wizard. Lines 12–15 are include files to import the necessary header files. Lines 19–29 define the constants and members for the prototypes. Lines 30–86 give the class definition with the necessary variables and methods required in the CommandDlg.cpp file. These variables and methods are explained in the detailed explanation provided for Listing 9-8.

Listing 9-8: CommandDlg.cpp

```

1. // SDP Information CommandsDlg.cpp : implementation file
2. //
3.
4. #include "stdafx.h"
5. #include "SDP Information Commands.h"
6. #include "SDP Information CommandsDlg.h"
7.
8.
9.
10. #include <exp\msg.h>
11. #include <exp\vos2com.h>
12. #include <exp\hci.h>
13. #include <exp\hci_drv.h>
14. #include <exp\sil.h>
15. #include <exp\com.h>
16. #include <exp\sd.h>
17. #include <exp\dbm.h>
18. #include <exp\scm.h>
19. #include <exp\sds.h>
20.
21.
22.
23. #ifdef _DEBUG
24. #define new DEBUG_NEW
25. #undef THIS_FILE
26. static char THIS_FILE[] = __FILE__;
27. #endif
28.
29.
30.
31. union MessageMapFunctions
32. {
33.     AFX_PMSG pfn;    // generic member function pointer
34.     void      (AFX_MSG_CALL CWnd::*pfn_btfn)(void **);
35. };
36.
37.
38. //////////////////////////////////////
39. // CAboutDlg dialog used for App About
40.
41. class CAboutDlg : public CDialog
42. {
43. public:
44.     CAboutDlg();
45.
46.     // Dialog Data
47.     //{{AFX_DATA(CAboutDlg)
48.     enum { IDD = IDD_ABOUTBOX };
49.     //}}AFX_DATA
50.
51.     // ClassWizard generated virtual function overrides
52.     //{{AFX_VIRTUAL(CAboutDlg)
53. protected:
54.     virtual void DoDataExchange(CDataExchange* pDX);
55.     //DDX/DDV support
56.     //}}AFX_VIRTUAL

```

```

55.
56. // Implementation
57. protected:
58. //{AFX_MSG(CAboutDlg)
59. //}AFX_MSG
60. DECLARE_MESSAGE_MAP()
61. };
62.
63. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
64. {
65. //{AFX_DATA_INIT(CAboutDlg)
66. //}AFX_DATA_INIT
67. }
68.
69. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
70. {
71. CDialog::DoDataExchange(pDX);
72. //{AFX_DATA_MAP(CAboutDlg)
73. //}AFX_DATA_MAP
74. }
75.
76. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
77. //{AFX_MSG_MAP(CAboutDlg)
78. // No message handlers
79. //}AFX_MSG_MAP
80. END_MESSAGE_MAP()
81.
82. ///////////////////////////////////////////////////////////////////
83. // CSDPInformationCommandsDlg dialog
84.
85. CSDPInformationCommandsDlg::CSDPInformationCommandsDlg(CWnd* pParent
    /*=NULL*/)
86. : CDialog(CSDPInformationCommandsDlg::IDD, pParent)
87. {
88. //{AFX_DATA_INIT(CSDPInformationCommandsDlg)
89. // NOTE: the ClassWizard will add member initialization here
90. //}AFX_DATA_INIT
91. // Note that LoadIcon does not require a subsequent DestroyIcon in
    Win32
92. m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
93. m_pServerEvents = new Events();
94. }
95.
96. void CSDPInformationCommandsDlg::DoDataExchange(CDataExchange* pDX)
97. {
98. CDialog::DoDataExchange(pDX);
99. //{AFX_DATA_MAP(CSDPInformationCommandsDlg)
100. // NOTE: the ClassWizard will add DDX and DDV calls here
101. //}AFX_DATA_MAP
102. }
103.
104. BEGIN_MESSAGE_MAP(CSDPInformationCommandsDlg, CDialog)
105. //{AFX_MSG_MAP(CSDPInformationCommandsDlg)
106. ON_WM_SYSCOMMAND()
107. ON_WM_PAINT()
108. ON_WM_QUERYDRAGICON()

```

```

109.  ON_BN_CLICKED(IDC_BUTTON1, OnStart)
110.  ON_BLUETOOTH_EVENT(DBM_START_CNF, OnDbmStartCnf)
111.  ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF, OnDbmRegisterCnf)
112.
    ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF_NEG, OnDbmRegisterCnfNeg)
113.  //}}AFX_MSG_MAP
114.  END_MESSAGE_MAP()
115.
116.
////////////////////////////////////
117. // CSDPInformationCommandsDlg message handlers
118.
119. BOOL CSDPInformationCommandsDlg::OnInitDialog()
120. {
121.  CDialog::OnInitDialog();
122.
123.  // Add "About..." menu item to system menu.
124.
125.  // IDM_ABOUTBOX must be in the system command range.
126.  ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
127.  ASSERT(IDM_ABOUTBOX < 0xF000);
128.
129.  CMenu* pSysMenu = GetSystemMenu(FALSE);
130.  if (pSysMenu != NULL)
131.  {
132.      CString strAboutMenu;
133.      strAboutMenu.LoadString(IDS_ABOUTBOX);
134.      if (!strAboutMenu.IsEmpty())
135.      {
136.          pSysMenu->AppendMenu(MF_SEPARATOR);
137.          pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
138.      }
139.  }
140.
141.  // Set the icon for this dialog. The framework does this automatically
142.  // when the application's main window is not a dialog
143.  SetIcon(m_hIcon, TRUE);           // Set big icon
144.  SetIcon(m_hIcon, FALSE);        // Set small icon
145.
146.  // TODO: Add extra initialization here
147.  m_pServerEvents->m_pParentDialog = this;
148.  return TRUE; // return TRUE unless you set the focus to a control
149. }
150.
151. void CSDPInformationCommandsDlg::OnSysCommand(UINT nID, LPARAM lParam)
152. {
153.  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
154.  {
155.      CAboutDlg dlgAbout;
156.      dlgAbout.DoModal();
157.  }
158.  else
159.  {
160.      CDialog::OnSysCommand(nID, lParam);
161.  }
162. }

```



```

163.
164. // If you add a minimize button to your dialog, you will need the code below
165. // to draw the icon. For MFC applications using the document/view model,
166. // this is automatically done for you by the framework.
167.
168. void CSDPInformationCommandsDlg::OnPaint()
169. {
170.     if (IsIconic())
171.     {
172.         CPaintDC dc(this); // device context for painting
173.
174.         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
175.
176.         // Center icon in client rectangle
177.         int cxIcon = GetSystemMetrics(SM_CXICON);
178.         int cyIcon = GetSystemMetrics(SM_CYICON);
179.         CRect rect;
180.         GetClientRect(&rect);
181.         int x = (rect.Width() - cxIcon + 1) / 2;
182.         int y = (rect.Height() - cyIcon + 1) / 2;
183.
184.         // Draw the icon
185.         dc.DrawIcon(x, y, m_hIcon);
186.     }
187.     else
188.     {
189.         CDialog::OnPaint();
190.     }
191. }
192.
193. // The system calls this to obtain the cursor to display while the user
194. // drags
195. // the minimized window.
196. HCURSOR CSDPInformationCommandsDlg::OnQueryDragIcon()
197. {
198.     return (HCURSOR) m_hIcon;
199. }
200.
201.
202. LRESULT CSDPInformationCommandsDlg::WindowProc(UINT message, WPARAM wParam,
203. LPARAM lParam)
204. {
205.     // TODO: Add your specialized code here and/or call the base class
206.     MSG_TMsg **ptMsg;
207.
208.     if (message == WM_BLUETOOTH_EVENT)
209.     {
210.         // it is a Bluetooth event so call the corresponding handlefunction
211.         OnBluetoothEvent( message, wParam, lParam);
212.
213.         ptMsg = (MSG_TMsg**)lParam;
214.         /* free the message received from the bluetooth server */
215.         if (*ptMsg != NULL)
216.             VOS_Free((void **)lParam);
217.     }

```

```

217. return CDialog::WindowProc(message, wParam, lParam);
218. }
219.
220. BOOL CSDPInformationCommandsDlg::OnBluetoothEvent(UINT message, WPARAM
    wParam, LPARAM lParam)
221. {
222.     const AFX_MSGMAP* pMessageMap;
223.     const AFX_MSGMAP_ENTRY* lpEntry;
224.
225.     // look through message map to see if it applies to us
226. #ifdef _AFXDLL
227.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
228.         pMessageMap = (*pMessageMap->pfnGetBaseMap)())
229. #else
230.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
231.         pMessageMap = pMessageMap->pBaseMap)
232. #endif
233.     {
234.         // Note: catches BEGIN_MESSAGE_MAP(CMyClass, CMyClass)!
235. #ifdef _AFXDLL
236.         ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
237. #else
238.         ASSERT(pMessageMap != pMessageMap->pBaseMap);
239. #endif
240.
241.         // C version of search routine
242.         lpEntry = (AFX_MSGMAP_ENTRY*)(&pMessageMap->lpEntries[0]);
243.         while (lpEntry->nSig != AfxSig_end)
244.         {
245.             if ((lpEntry->nMessage == message) && (lpEntry->nCode == wParam))
246.             {
247.                 // found it
248.                 union MessageMapFunctions mmf;
249.                 mmf.pfn = lpEntry->pfn;
250.
251.                 //lets call the function to handle the message
252.                 (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
253.
254.                 return TRUE;
255.             }
256.             lpEntry++;
257.         }
258.         /* unable to find a handler function for this Bluetooth event */
259.         return FALSE;
260.     }
261.     return FALSE;
262. }
263.
264.
265. void CSDPInformationCommandsDlg::OnStart()
266. {
267.     // TODO: Add your control notification handler code here
268.     AfxMessageBox("DBM_ReqStart(0)");
269.     DBM_ReqStart(0);
270. }
271.

```

```

272. void CSDPInformationCommandsDlg::OnDbmStartCnf(void **ppMsg)
273. {
274.
275.     CString sAddress;
276.     DBM_TStartCnf *ptStartCnf;
277.
278.     AfxMessageBox("DBM Start Confirmation");
279.
280.     ptStartCnf =(DBM_TStartCnf *) *ppMsg;
281.
282.     // Create Profile for SerialPort
283.
284.     m_pSerialPort = new CRS232(PROFILE_SERIAL);
285. }
286.
287. void CSDPInformationCommandsDlg::OnDbmRegisterCnf(void **ppMsg)
288. {
289.
290.     AfxMessageBox("DBM Register Service Confirmation");
291.
292.     DBM_TRegisterServiceCnf *ptRegisterCnf = (DBM_TRegisterServiceCnf *)
        *ppMsg;
293.
294.     // write the profile in DBM
295.     m_pSerialPort->WriteProfile(ptRegisterCnf->ulDbmHandle);
296.
297.     AfxMessageBox("New Service Was Registered");
298.
299.
300. }
301.
302. void CSDPInformationCommandsDlg::OnDbmRegisterCnfNeg(void **ppMsg)
303. {
304.     ppMsg = ppMsg;
305.     // let the user know
306.     MessageBox(_T("Could not register to Data Base Manager"));
307.
308.
309.     DestroyWindow();
310. }

```

Code Description

- ◆ Lines 1–19: All the necessary function prototypes and variables are included in this class. Some of these included files exports Bluetooth APIs, constants, and variables.
- ◆ Lines 23–27: Visual C++ editor generates these lines to give a common framework for all MFC applications through the application wizard.
- ◆ Lines 31–35: The union named `MessageMapFunctions` is defined.
- ◆ Lines 38–82: These lines indicate the MFC Application Wizard framework for About dialog.
- ◆ Lines 83–94: Visual C++ editor adds this default class initialization code.
- ◆ Line 93: The `m_pServerEvents` is an instance of the `Events` class.
- ◆ Lines 96–102: Visual C++ class wizard adds the lines for DDX and DDV.

- ◆ Lines 104–114: The following message map macros are default macros provided by VC++ MFC Application Wizard.
 - `ON_WM_SYSCOMMAND` ()
 - `ON_WM_PAINT` ()
 - `ON_WM_QUERYDRAGICON` ()
- ◆ The `ON_BLUETOOTH_EVENT` message map macro indicates which function handles a specified BLUETOOTH event. The following table provides more details.

BLUETOOTH Event	Handler Function Name
<code>BM_START_CNF</code>	<code>OnDbmStartCnf</code>
<code>DBM_REGISTER_SERVICE_CNF</code>	<code>OnDbmRegisterCnf</code>
<code>DBM_REGISTER_SERVICE_CNF</code>	<code>OnDbmRegisterCnf</code>

- ◆ The `ON_BN_CLICKED` message map macro indicates which button click handles a specified Bluetooth event (see the following table).

Button ID	Function Name
<code>IDC_BUTTON1</code>	<code>OnStart</code>

- ◆ Lines 117–149: `OnInitDialog()` method is provided by Visual C++ MFC Application wizard to initialize necessary information.
 - `m_pServerEvents->m_pParentDialog = this;` — This statement is used to associate current dialog with the `Events` class member to get all the outgoing events from `BT_COMSERVER`.
- ◆ Lines 151–162: `OnSysCommand(UINT nID, LPARAM lParam)` method is provided by Visual C++ MFC application wizard to model the dialog.
- ◆ Lines 168–191: `OnPaint()` method is defined by Visual C++ MFC application wizard to Paint controls on the current dialog.
- ◆ Lines 195–198: `OnQueryDragIcon()` method is defined by Visual C++ MFC application wizard to create an ICON for the current dialog.
- ◆ Lines 202 to 218: The method `WindowProc(UINT message, WPARAM wParam, LPARAM lParam)` is discussed in the HCI Information Commands programming application.
- ◆ Lines 220–262: The method `OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam)` is also discussed in the HCI Information Commands programming application.
- ◆ Lines 265–270: When the button `IDC_BUTTON1` is clicked, the corresponding message-handler function `OnStart()` is called.
 - `DBM_ReqStart(0)` is a BLUETOOTH SDK API, which is used to send a command to start the Database Manager on BLUETOOTH module.
 - Parameter1: sequence number of the interface=0
 - `DBM_ReqStart(0)` causes the BLUETOOTH module to fire either a `DBM_START_CNF` or `DBM_START_CNF_NEG` event corresponding to success or failure.
- ◆ Lines 272–285: `DBM_START_CNF` event makes the `ON_BLUETOOTH_EVENT` macro to call `OnDbmStartCnf(void **ppMsg)` message handler function. A Message Box indicates that the start of Database Manager is a success.
 - `m_pSerialPort = new CRS232(PROFILE_SERIAL)` — After the Database Manager has been started, the profile for serial port can be created to implement BLUETOOTH print service.

- ◆ Lines 287 to 300: DBM_REGISTER_SERVICE_CNF event makes the ON_BLUETOOTH_EVENT macro to call OnDbmRegisterCnf(void **ppMsg) message-handler function. A Message Box indicates that the registration of a service with Database Manager is a success.
 - The WriteProfile(ptRegisterCnf->ulDbmHandle) is a member function of class CRS232 which is written to add descriptors and attributes for the service
- ◆ Lines 302–310: DBM_REGISTER_SERVICE_CNF_NEG event makes the ON_BLUETOOTH_EVENT macro to call OnDbmRegisterCnfNeg(void **ppMsg) message handler function. A message box indicates that the registration of a service with Database Manager has failed. The current dialog box will be destroyed.

Listing 9-9 gives the source code for the header file PrintProfile.h. This file contains the declarations of variables implemented in PrintProfile.cpp.

Listing 9-9: PrintProfile.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
//PrintProfile.h declarations of variables implemented in PrintProfile.cpp
1. #if
   !defined(AFX_PRINTPROFILE_H__B020C543_F4C6_11D3_B6C4_00105A680F8F__INCLUDE)
2. #define AFX_PRINTPROFILE_H__B020C543_F4C6_11D3_B6C4_00105A680F8F__INCLUDED_
3.
4. #if _MSC_VER > 1000
5. #pragma once
6. #endif // _MSC_VER > 1000
7.
8.
9. #include <exp\standard.h>
10.
11. #define PROFILE_SERIAL      ((uint16) 6)
12. #define SRP_SERIAL_GENERIC_SERIALPORT_UUID      ((uint16)
    0x1101)
13. #define SRP_SERIAL_SERVICENAME_OFFSET      ((uint16)
    0x0000)
14.
15.
16. class CPrintProfile
17. {
18. public:
19.   CPrintProfile();
20.   CPrintProfile(uint16 uiId);
21.
22.   virtual ~CPrintProfile();
23.
24.
25.   virtual void WriteProfile(uint32 ulSRPHandle);
26.
27.   uint32 GetSRPHandle();
28.   void   GetProfileName(char **pcName);
29.
30. protected:
31.   uint32 m_ulSRPHandle;
32.
33.   uint16 m_uiProfileId;
```

```

34.
35. char    *m_pcName;
36.
37. void    AddDesc_ServiceClassIDList(uint16 uiSeqnr,
38.                                     uint16 uiServiceClassValue);
39. void    AddAttr_ProtocolDescriptorList(uint16 uiSeqnr);
40.
41. void    AddAttr_ServiceName(uint16 uiSeqnr,
42.                              uint16 uiOffset,
43.                              char    *pcTextName,
44.                              uint16 uiSize);
45.
46. };
47.
48. #endif

```

Code Description

In Listing 9-9, the lines 1 to 6 are automatically generated by the VC++ application wizard. Line 9 is for inclusion of the header file. Lines 11 to 13 are define statements as required by the Bluetooth SDK. Lines 16 to 48 are for class declaration with the necessary variables and methods, which are described in Listing 9-10.

Listing 9-10: PrintProfile.cpp

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. //PrintProfile.cpp: implementation of the CProfile class.
2.
3. #include "stdafx.h"
4.
5. #include "PrintProfile.h"
6.
7. #include <exp\vos2com.h>
8. #include <exp\dbm.h>
9.
10. #ifdef _DEBUG
11. #undef THIS_FILE
12. static char THIS_FILE[]=__FILE__;
13. #define new DEBUG_NEW
14. #endif
15.
16.
17. CPrintProfile::CPrintProfile()
18. {
19.     m_ulSRPHandle = 0;
20.     m_uiProfileId = 0;
21.
22. }
23.
24. CPrintProfile::CPrintProfile(uint16 uiId)
25. {
26.     m_ulSRPHandle = 0;
27.     m_uiProfileId = uiId;
28.
29.     DBM_ReqRegisterService(uiId,DBM_ServiceDiscoveryDB);

```

```

30.
31. }
32.
33. CPrintProfile::~CPrintProfile()
34. {
35.
36. }
37.
38. void CPrintProfile::WriteProfile(uint32 ulSRPHandle)
39. {
40.     m_ulSRPHandle = ulSRPHandle;
41. }
42.
43. uint32 CPrintProfile::GetSRPHandle()
44. {
45.     return m_ulSRPHandle;
46. }
47. void CPrintProfile::GetProfileName(char **pcName)
48. {
49.     *pcName = m_pcName;
50. }
51.
52.
53. void CPrintProfile::AddDesc_ServiceClassIDList(uint16 uiSeqnr, uint16
    uiServiceClassValue)
54. {
55.     DBM_TDescriptorUuid      tDescriptorUuid;
56.     DBM_TDescriptorValue     tDescriptor;
57.
58.     tDescriptorUuid.tType      = DBM_DET_UUID16;
59.     tDescriptorUuid.pucDescriptorUuidValue = (uint8*)
        VOS_Alloc(sizeof(uint16));
60.     memcpy(tDescriptorUuid.pucDescriptorUuidValue, &uiServiceClassValue,
        sizeof(uint16));
61.
62.     tDescriptor.uiNrOfParams      = 0;
63.     tDescriptor.uiSizeOfValueInBytes = 0;
64.     tDescriptor.pucValue         = NULL;
65.
66.     /* Add Descriptor */
67.     AfxMessageBox("DBM_ReqAddDescriptor (uiSeqnr, m_ulSRPHandle,
        BT_SERVICE_CLASS_ID_LIST,&tDescriptorUuid, &tDescriptor)");
68.     DBM_ReqAddDescriptor (uiSeqnr, m_ulSRPHandle,
        BT_SERVICE_CLASS_ID_LIST,&tDescriptorUuid, &tDescriptor);
69.
70.     VOS_Free((void*)&tDescriptorUuid.pucDescriptorUuidValue);
71. }
72.
73.
74. void CPrintProfile::AddAttr_ProtocolDescriptorList(uint16 uiSeqnr)
75. {
76.     DBM_TAttributeValue      tAttribute;
77.
78.     tAttribute.tType          = DBM_DET_NULL;
79.     tAttribute.pucValue       = NULL;
80.

```

```

81. /* Add Attribute */
82. AfxMessageBox("DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle,
    BT_PROTOCOL_DESCRIPTOR_LIST,      &tAttribute)");
83. DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle, BT_PROTOCOL_DESCRIPTOR_LIST,
    &tAttribute);
84. }
85.
86. void CPrintProfile::AddAttr_ServiceName(uint16 uiSeqnr,  uint16 uiOffset,
    char *pcTextName, uint16 uiSize)
87. {
88.     DBM_TAttributeValue  tAttribute;
89.     uint8                *pucTemp;
90.
91.     if (uiSize < 256)
92.     {
93.         tAttribute.tType      = DBM_DET_STRING8;
94.         tAttribute.pucValue   = (uint8 *)VOS_Alloc((uint16)(uiSize +
            sizeof(uint8)));
95.         memcpy(tAttribute.pucValue, &uiSize, sizeof(uint8));
96.         pucTemp = tAttribute.pucValue + sizeof(uint8);
97.         memcpy(pucTemp, pcTextName, (uiSize));
98.     }
99.     else if (uiSize < 65536)
100.    {
101.        tAttribute.tType      = DBM_DET_STRING16;
102.        tAttribute.pucValue   = (uint8 *)VOS_Alloc((uint16)(uiSize +
            sizeof(uint16)));
103.        memcpy(tAttribute.pucValue, &uiSize, sizeof(uint16));
104.        pucTemp = tAttribute.pucValue + sizeof(uint16);
105.        memcpy(pucTemp, pcTextName, (uiSize));
106.    }
107.     else
108.    {
109.        tAttribute.tType      = DBM_DET_STRING8;
110.        tAttribute.pucValue   = (uint8 *)VOS_Alloc((uint16)(uiSize +
            sizeof(uint32)));
111.        memcpy(tAttribute.pucValue, &uiSize, sizeof(uint32));
112.        pucTemp = tAttribute.pucValue + sizeof(uint32);
113.        memcpy(pucTemp, pcTextName, (uiSize));
114.    }
115.
116. /* Add Attribute */
117. AfxMessageBox("DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle,
    (uint16)(BT_SERVICE_NAME(uiOffset)), &tAttribute)");
118. DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle,
    (uint16)(BT_SERVICE_NAME(uiOffset)),&tAttribute);
119. VOS_Free((void*)&tAttribute.pucValue);
120. }

```

Code Description

- ◆ Lines 3–8: The header files required for this application are included.
- ◆ Lines 10–14: The Visual C++ editor provided these lines to give a common framework for all applications using the MFC Application Wizard.

- ◆ Lines 17–22: The default constructor is defined to initialize service handle and Database Manager handles to zero.
- ◆ Lines 24–31: The constructor is overridden to accept user-defined input for profileID.
 - DBM_ReqRegisterService (uiId,DBM_ServiceDiscoveryDB) is a Bluetooth SDK API used to send a command to register a service with the Database Manager on a Bluetooth module.
 - Parameter1: user-defined Database manager handle.
 - Parameter2: service discovery database.
 - DBM_ReqRegisterService (uiId,DBM_ServiceDiscoveryDB) causes the Bluetooth module to fire either DBM_REGISTER_SERVICE_CNF or DBM_REGISTER_SERVICE_CNF_NEG event corresponding to success or failure.
- ◆ Lines 33–36: The destructor is used to destruct the class.
- ◆ Lines 38–41: m_ulSRPHandle = ulSRPHandle; indicates that the service record handle is stored in the member variable of the class.
- ◆ Lines 43–46: This method is declared to get the handle of service record.
- ◆ Lines 47–50: This method is used to return the profile name.
- ◆ Lines 53–71: DBM_ReqAddDescriptor (uiSeqnr, m_ulSRPHandle, BT_SERVICE_CLASS_ID_LIST,&tDescriptorUuid, &tDescriptor); is used to add a descriptor for service class ID list.
- ◆ Lines 74–84: DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle, BT_PROTOCOL_DESCRIPTOR_LIST, &tAttribute); is used to add a descriptor for protocol list.
- ◆ Lines 86–120: DBM_ReqAddAttribute (uiSeqnr, m_ulSRPHandle, (uint16) (BT_SERVICE_NAME(uiOffset)),&tAttribute); is used to add service name.

Listing 9-11 gives the source code for the header file RS232.h, in which variables required for RS232.cpp are declared.

Listing 9-11: RS232.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. //RS232.H
2. #if !defined(AFX_RS232_H__F8A72756_F4EB_11D3_B6C4_00105A680F8F__INCLUDED_)
3. #define AFX_RS232_H__F8A72756_F4EB_11D3_B6C4_00105A680F8F__INCLUDED_
4.
5. #if _MSC_VER > 1000
6. #pragma once
7. #endif // _MSC_VER > 1000
8.
9. #include "PrintProfile.h"
10.
11. class CRS232 : public CPrintProfile
12. {
13. public:
14.   CRS232();
15.   CRS232(uint16 uiId);
16.
17.
18.   virtual ~CRS232();

```

```

19.
20.
21.
22.   void WriteProfile(uint32 ulSRPHandle);
23.
24.
25. };
26.
27. #endif

```

Code Description

In Listing 9-11, the lines 2–7 are generated by VC++ application wizard, line 9 is an include file, and Lines 11–27 are for class declaration. The variables and methods used in the class are described in Listing 9-12.

Listing 9-12: RS232.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

//RS232.CPP
1. #include "stdafx.h"
2. #include "RS232.h"
3.
4. #include <exp\dbm.h>
5. #ifdef _DEBUG
6. #undef THIS_FILE
7. static char THIS_FILE[]=__FILE__;
8. #define new DEBUG_NEW
9. #endif
10.
11. CRS232::CRS232()
12. {
13.     m_pcName = NULL;
14.
15. }
16.
17. CRS232::~CRS232()
18. {
19.
20. }
21.
22. CRS232::CRS232(uint16 uiId)
23. {
24.     m_ulSRPHandle = 0;
25.     m_uiProfileId = uiId;
26.
27. m_pcName = "Print Service\0";
28. AfxMessageBox("DBM_ReqRegisterService(m_uiProfileId,
    DBM_ServiceDiscoveryDB )");
29. DBM_ReqRegisterService(m_uiProfileId,DBM_ServiceDiscoveryDB);
30.
31. }
32.
33. void CRS232::WriteProfile(uint32 ulSRPHandle)
34. {

```

```

35. m_ulSRPHandle = ulSRPHandle;
36.
37. AfxMessageBox("Next 3 Commands For Reg. Print Service");
38.
39. AddDesc_ServiceClassIDList(0,SRP_SERIAL_GENERIC_SERIALPORT_UUID);
40.
41.
42. AddAttr_ProtocolDescriptorList(1);
43.
44.
45.
46. AddAttr_ServiceName(2, SRP_SERIAL_SERVICENAME_OFFSET,
47.                      m_pcName,
48.                      (uint16) strlen(m_pcName));
49.
50.
51. }

```

Code Description

- ◆ Lines 1–9: Lines 1–4 are for inclusion of header files. The remaining lines are generated by Visual C++ editor to give a common framework for all applications using the MFC application wizard.
- ◆ Lines 11–15: Constructor for the class.
- ◆ Lines 17–20: Destructor for the class.
- ◆ Lines 22–31: The constructor is overridden to call `DBM_ReqRegisterService(m_uiProfileId,DBM_ServiceDiscoveryDB);`
- ◆ Lines 33–51: `WriteProfile (uint32 ulSRPHandle)` is defined to call the following functions:
 - `AddDesc_ServiceClassIDList(0,SRP_SERIAL_GENERIC_SERIALPORT_UUID);`
 - `AddAttr_ProtocolDescriptorList(1);`
 - `AddAttr_ServiceName(2, SRP_SERIAL_SERVICENAME_OFFSET,`
 - `m_pcName,(uint16) strlen(m_pcName));`

Code Output

When the application is built in the VC++ environment and executed, the main window is displayed (see Figure 9-6). This window has one button labeled Register New Service.



Figure 9-6: Register New Service window

After this button is clicked, the following message box appears to show that the command `DBM_ReqStart(0)` has been issued.



After clicking the OK button, the following message box appears to show the confirmation for the previous command.



After clicking the OK button, the following message box appears to show that the command `DBM_ReqRegisterService(m_uiProfileId,DBM_ServiceDiscoveryDB)` has been issued.



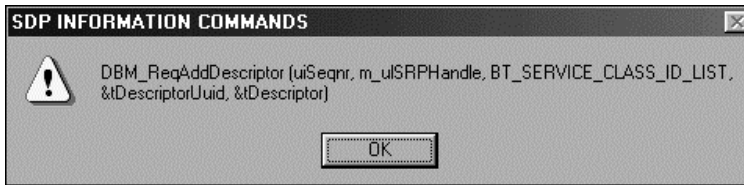
After the OK button is clicked, the following message box appears indicating the confirmation of the previous command.



After the OK button is clicked, the following message box appears.



Click OK to issue the following commands.



The following message box shows the service has been registered with the Database Manager.



The sample chat client program can be used to see the service from the remote Bluetooth device, and when the Get Services button is clicked in the main window, the print service is displayed.

File Transfer Application

This full-fledged application demonstrates a file transfer utility. Using HCI, SDP, and RFCOMM API calls of the PC Reference Stack, we develop an application to transfer a file from one Bluetooth-enabled PC to another Bluetooth-enabled PC over a Bluetooth radio link. The file is transferred from a client Bluetooth device to a server Bluetooth device. The server provides the file service, which is to be accessed by the client.

The software contains the following three modules:

- ◆ Common module (common to both the server and the client): The common module provides the mechanism to set up an ACL link and to access the service. This module is implemented with the classes `Cservice` and `CconnectionInfo` in the files named `Service.cpp` and `ConnectionInfo.cpp` respectively.
- ◆ Client module: The client module gets the service from the server and transfers a file over RFCOMM. The client module is implemented with the class `CRadioFileClientDlg` in the file named `RadioFileClientDlg.cpp`.

- ◆ Server module: The server module registers a service with the Database Manager and the file service is made available to all nearby clients. The server module is implemented with the class `CRadioFileServerDlg` in the file named `RadioFileServerDlg.cpp`.

Common Module

The source code for `Service.h`, `Service.cpp`, `ConnectionInfo.h`, and `ConnectionInfo.cpp` is given in Listings 9-13, 9-14, 9-15 and 9-16, respectively. The header files contain the statements for inclusion of other header files including the library files and declarations of variables and methods for the classes used in the CPP files. The detailed explanation for the variables and methods used along with Bluetooth function calls are explained with reference to the CPP files.

Listing 9-13: Service.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
// Service.h: interface for the CService class.
1.
2. #include <exp/standard.h>
3. class CService
4. {
5. public:
6. CService();
7. CService(CString sService);
8. virtual ~CService();
9.
10. CString GetService();
11. void SetService(CString sService);
12.
13. uint16   m_SDCHandle;
14. uint32   m_ServiceRecordHandle;
15. uint16   m_CurrentNumber;
16. uint8     m_pAttributeData[100];
17. uint16   m_AttributeListByteCount;
18. char     m_pServiceName[100];
19.
20.
21. private:
22. CString m_sService;
23. };
```

Code Description

Listing 9-13 is the header file in which variables used in `Service.cpp` are declared. The functionality of these variables will be evident in `Service.cpp`. The explanation of the variables is given in the code description of `Service.cpp`.

Listing 9-14:Service.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. // Service.cpp: implementation of the CService class.
2. #include "stdafx.h"
3. #include "RadioChat.h"
4. #include "Service.h"
5. #include <exp/vos.h>
```

```

6. #ifdef _DEBUG
7. #undef THIS_FILE
8. static char THIS_FILE[]=__FILE__;
9. #define new DEBUG_NEW
10. #endif
11.
12. CService::CService()
13. {
14.
15.     m_sService.Empty();
16. }
17.
18. CService::CService(CString sService)
19. {
20.     m_sService = sService;
21. }
22.
23. CService::~CService()
24. {
25. }
26. void CService::SetService (CString sService)
27. {
28.     m_sService = sService;
29. }
30.
31. CString CService::GetService ()
32. {
33.     CString sService;
34.
35.     sService.Format(_T("%d, Service Name %s"),m_SDCHandle,m_pServiceName);
36.
37.     return sService;
38. }

```

Code Description

- ◆ Line 1–5: The files required to implement service are included.
- ◆ Line 6–10: These lines are included by VC++ to provide a common framework for all MFC applications.
- ◆ Line 12–16: The default constructor is defined. The `m_sService` is initialized to contain an empty string.
- ◆ Line 18–21: The default constructor is overridden to initialize the user service to member variable of `CService` class.
- ◆ Line 23–25: The default destructor is defined.
- ◆ Line 26–29: The method is defined to initialize the specified service to the member variable `m_sService`.
- ◆ Line 31–38: The method returns the service name and service handle in the form of a formatted string.

Listing 9-15: ConnectionInfo.h

```

//ConnectionInfo.h
1. #include <exp/bt.h>
2. class CConnectionInfo
3. {
4. public:
5.   CConnectionInfo();
6.   virtual ~CConnectionInfo();
7.   BT_TAddress          tAddress;
8.   uint16               uiServiceClass;
9.   uint8                tPacketType;
10.  uint8                tPageScanRepMode;
11.  uint8                tPageScanMode;.
12.  uint16               tClockOffset;
13.  uint16               uiMaxFrameSize;
14.  uint32               ulDbmHandle;

15.  BT_THandle           tAclHandle;
16.  uint16               uiSdcHandle;
17.  uint8                pucAttributeData[100];
18.  uint16               uiAttributeListByteCount;
19.  uint32               ulServiceRecordHandle;
20.  uint16               uiRFCCommHandle;
21.  int8                pcServiceName[100];
22. };

```

Code Description

The variables declared in Listing 9-15 are used in the program ConnectionInfo.cpp. The functionality of these variables will be evident in the code description given for ConnectionInfo.cpp.

Listing 9-16: ConnectionInfo.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1.  //ConnectionInfo.cpp
2.  #include "stdafx.h"
3.  #include "RadioChat.h"
4.  #include "ConnectionInfo.h"
5.  #include <exp/vos.h>
6.  #include <exp/com.h>
7.
8.  #ifdef _DEBUG
9.  #undef THIS_FILE
10. static char THIS_FILE[]=__FILE__;
11. #define new DEBUG_NEW
12. #endif
13.
14. CConnectionInfo::CConnectionInfo()
15. {
16.   uiMaxFrameSize = COM_DEFAULT_MFS;
17.   tAclHandle = 0;
18.   ulDbmHandle = 0;
19. }
20. CConnectionInfo::~CConnectionInfo()
21. {
22. }

```


Code Description

- ◆ Line 1–6: The files required to implement CConnectionInfo class are included.
- ◆ Line 8–12: These lines are included by VC++ to provide a common framework for all MFC applications.
- ◆ Line 14–19: The default constructor is defined to initialize the ACL link, Database Manager (DBM), and RFCOMM parameters.
- ◆ Line 20–22: The default destructor is defined.

Client Module

The source code for RadioFileClientDlg.h and RadioFileClientDlg.cpp are given in Listings 9-17 and 9-18, respectively.

Listing 9-17: RadioFileClientDlg.h

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. // RadioFileClientDlg.h : header file
2.
3. #include "Events.h"
4. #include "ConnectionInfo.h"
5. #include "Remotedevice.h"
6. #include "service.h"
7. #include <exp\sd.h>
8. #include <exp\BT_COMServer.h>
9. #include <afxtempl.h>
10.
11. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
12. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
13.     { WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
14.         (AFX_PMSG) (AFX_PMSGW) (void (AFX_MSG_CALL CWnd::*) (void
15.         **)) &memberFxn },
16. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
17.     SendMessage( (HWND) this->m_hWnd, WM_BLUETOOTH_EVENT, (LPARAM) uiBtEventID, (LPARAM) &pMsg)
18.
19. class CRadioFileClientDlg : public CDialog
20. {
21. public:
22.     CRadioFileClientDlg(CWnd* pParent = NULL); // standard constructor
23.     ~CRadioFileClientDlg();
24.     CConnectionInfo m_ConnectionInfo;
25. private:
26.     void InitSecurityClient();
27.     BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
28.     //{{AFX_DATA(CRadioFileClientDlg)
29.     enum { IDD = IDD_RADIOCHAT_DIALOG };
30.     CListBox m_ChatchArea;
31.     CEdit m_InputChat;
32.     CTreeCtrl m_tree;
33.     //}}AFX_DATA
34.     //{{AFX_VIRTUAL(CRadioFileClientDlg)
35. public:
36.     virtual BOOL DestroyWindow();

```

```

36. protected:
37. virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
38. virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);
39. //}}AFX_VIRTUAL
40. public:
41. void HandleReturn();
42. void AddDevice(CString sAddress, CString sName);
43. void AddDevice(CDevice device);
44. void ShowAllDevicesFound();
45. void AddService(CString sService);
46. void AddService(CService service);
47. void ShowAllServicesFound();
48. void AskForServiceName();
49. void ReceiveServiceName(SD_TServiceAttributeCnf *tServiceAttributeCnf);
50. void AskForServiceRecordHandle();
51. void ReceiveServiceRecordHandle(SD_TServiceAttributeCnf
*tServiceAttributeCnf);
52. protected:
53. HICON m_hIcon;
54. int index;
55. Events *m_pServerEvents;
56. CArray <CDevice,CDevice&> m_DevicesFound;
57. CArray <CService,CService&> m_ServicesFound;
58. int m_RemoteNameCounter;
59. int m_ServiceCounter;
60. //{AFX_MSG(CRadioFileClientDlg)
61. virtual BOOL OnInitDialog();
62. afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
63. afx_msg void OnPaint();
64. afx_msg HCURSOR OnQueryDragIcon();
65. afx_msg void OnInquiry();
66. afx_msg void OnSelDevice();
67. afx_msg void OnConnect();
68. afx_msg void OnGetservices();
69. afx_msg void OnSelservices();
70. afx_msg void OnDestroy();
71. afx_msg void OnSerialport();
72. afx_msg void OnCloseapplication();
73. afx_msg void OnHCISerial();
74. afx_msg void OnHCIUsb();
75. afx_msg void OnComStartCnf(void **ppMsg);
76. afx_msg void OnComStartCnfNeg(void **ppMsg);
77. afx_msg void OnComVersionCnf(void **ppMsg);
78. afx_msg void OnScmRegisterCnf(void **ppMsg);
79. afx_msg void OnScmRegisterCnfNeg(void **ppMsg);
80. afx_msg void OnConnectAcceptInd(void **ppMsg);
81. afx_msg void OnScmPincodeInd(void **ppMsg);
82. afx_msg void OnScmConnectCnf(void **ppMsg);
83. afx_msg void OnScmConnectCnfNeg(void **ppMsg);
84. afx_msg void OnScmConnectEvt(void **ppMsg);
85. afx_msg void OnScmDisconnectEvt(void **ppMsg);
86. afx_msg void OnScmDisconnectCnf(void **ppMsg);
87. afx_msg void OnScmDisconnectCnfNeg(void **ppMsg);
88. afx_msg void OnScmDeRegisterCnf(void **ppMsg);
89. afx_msg void OnScmDeRegisterCnfNeg(void **ppMsg);
90. afx_msg void OnSdStartCnf(void **ppMsg);

```

```

91.  afx_msg void OnSdConnectCnf(void **ppMsg);
92.  afx_msg void OnSdConnectCnfNeg(void **ppMsg);
93.  afx_msg void OnSdServiceSearchCnf(void **ppMsg);
94.  afx_msg void OnSdServiceSearchCnfNeg(void **ppMsg);
95.  afx_msg void OnSdServiceAttributeCnf(void **ppMsg);
96.  afx_msg void OnSdServiceAttributeCnfNeg(void **ppMsg);
97.  afx_msg void OnSdDisconnectCnf(void **ppMsg);
98.  afx_msg void OnDbmRegisterServiceCnf(void **ppMsg);
99.  afx_msg void OnDbmRegisterServiceCnfNeg(void **ppMsg);
100. afx_msg void OnDbmUnRegisterServiceCnf(void **ppMsg);
101. afx_msg void OnDbmUnRegisterServiceCnfNeg(void **ppMsg);
102. afx_msg void OnDbmAddDescriptorCnf(void **ppMsg);
103. afx_msg void OnDbmAddDescriptorCnfNeg(void **ppMsg);
104. afx_msg void OnHciConfigurePortConfirm(void **ppMsg);
105. afx_msg void OnHciConfigurePortConfirmNegative(void **ppMsg);
106. afx_msg void OnHciInquiryCnf(void **ppMsg);
107. afx_msg void OnHciInquiryEvt(void **ppMsg);
108. afx_msg void OnHciLocalAddressCnf(void **ppMsg);
109. afx_msg void OnHciLocalAddressCnfNeg(void **ppMsg);
110. afx_msg void OnHciRemoteNameCnf(void **ppMsg);
111. afx_msg void OnHciRemoteNameCnfNeg(void **ppMsg);
112. afx_msg void OnHciStartCnf(void **ppMsg);
113. afx_msg void OnHciWriteScanEnableCnf(void **ppMsg);
114. afx_msg void OnHciWriteScanEnableCnfNeg(void **ppMsg);
115.
116.
117. afx_msg void OnHciWriteAuthenticationModeCnf(void **ppMsg);
118. afx_msg void OnHciWriteAuthenticationModeCnfNeg(void **ppMsg);
119. afx_msg void OnHciWriteEncryptionModeCnf(void **ppMsg);
120. afx_msg void OnHciWriteEncryptionModeCnfNeg(void **ppMsg);
121. afx_msg void OnHciWriteCodCnf(void **ppMsg);
122. afx_msg void OnHciWriteCodCnfNeg(void **ppMsg);
123. afx_msg void OnHciWriteNameCnf(void **ppMsg);
124. afx_msg void OnHciWriteNameCnfNeg(void **ppMsg);
125. afx_msg void OnHciWriteConnectTimeoutCnf(void **ppMsg);
126. afx_msg void OnHciWriteConnectTimeoutCnfNeg(void **ppMsg);
127. afx_msg void OnHciWritePageTimeoutCnf(void **ppMsg);
128. afx_msg void OnHciWritePageTimeoutCnfNeg(void **ppMsg);
129. afx_msg void OnSilSetDeviceCnf(void **ppMsg);
130. afx_msg void OnSilSetDeviceCnfNeg(void **ppMsg);
131. afx_msg void OnSilReqDeviceCnf(void **ppMsg);
132. afx_msg void OnSilReqDeviceCnfNeg(void **ppMsg);
133. afx_msg void OnComConnectCnf(void **ppMsg);
134. afx_msg void OnComConnectCnfNeg(void **ppMsg);
135. afx_msg void OnComDataInd(void **ppMsg);
136. afx_msg void OnComDataCnf(void **ppMsg);
137. afx_msg void OnComDataCnfNeg(void **ppMsg);
138. afx_msg void OnComDisconnectEvt(void **ppMsg);
139. afx_msg void OnComDisconnectCnf(void **ppMsg);
140. afx_msg void OnComDisconnectCnfNeg(void **ppMsg);
141. afx_msg void OnBrowse();
142. //}}AFX_MSG
143. DECLARE_MESSAGE_MAP()
144. };
145.
146.

```



```

47. protected:
48.  //{AFX_MSG(CAboutDlg)
49.  //}AFX_MSG
50.  DECLARE_MESSAGE_MAP()
51. };
52. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
53. {
54.  //{AFX_DATA_INIT(CAboutDlg)
55.  //}AFX_DATA_INIT
56. }
57. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
58. {
59.  CDialog::DoDataExchange(pDX);
60.  //{AFX_DATA_MAP(CAboutDlg)
61.  //}AFX_DATA_MAP
62. }
63. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
64.  //{AFX_MSG_MAP(CAboutDlg)
65.  //}AFX_MSG_MAP
66. END_MESSAGE_MAP()
67. CRadioFileClientDlg::CRadioFileClientDlg(CWnd* pParent /*=NULL*/)
68. : CDialog(CRadioFileClientDlg::IDD, pParent)
69. {
70.  //{AFX_DATA_INIT(CRadioFileClientDlg)
71.  //}AFX_DATA_INIT
72.  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
73.  m_pServerEvents = new Events();
74. }
75. CRadioFileClientDlg::~CRadioFileClientDlg()
76. {
77.  m_DevicesFound.RemoveAll();
78.  m_ServicesFound.RemoveAll();
79.  delete m_pServerEvents;
80. }
81. void CRadioFileClientDlg::DoDataExchange(CDataExchange* pDX)
82. {
83.  CDialog::DoDataExchange(pDX);
84.  //{AFX_DATA_MAP(CRadioFileClientDlg)
85.  DDX_Control(pDX, IDC_LIST1, m_ChatArea);
86.  DDX_Control(pDX, IDC_EDIT1, m_InputChat);
87.  DDX_Control(pDX, IDC_TREE1, m_tree);
88.  //}AFX_DATA_MAP
89. }
90. BEGIN_MESSAGE_MAP(CRadioFileClientDlg, CDialog)
91.  //{AFX_MSG_MAP(CRadioFileClientDlg)
92.  ON_WM_SYSCOMMAND()
93.  ON_WM_PAINT()
94.  ON_WM_QUERYDRAGICON()
95.  ON_BLUETOOTH_EVENT(COM_START_CNF, OnComStartCnf)
96.  ON_BLUETOOTH_EVENT(COM_START_CNF_NEG, OnComStartCnfNeg)
97.  ON_BLUETOOTH_EVENT(COM_VERSION_CNF, OnComVersionCnf)
98.  ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF, OnScmRegisterCnf)
99.  ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF_NEG, OnScmRegisterCnfNeg)
100. ON_BLUETOOTH _EVENT(SCM_CONNECT_ACCEPT_IND, OnConnectAcceptInd)
101. ON_BLUETOOTH_EVENT(SCM_PINCODE_IND, OnScmPincodeInd)
102. ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF, OnScmConnectCnf)

```

```

103. ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF_NEG, OnScmConnectCnfNeg)
104. ON_BLUETOOTH_EVENT(SCM_CONNECT_EVT, OnScmConnectEvt)
105. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_EVT, OnScmDisconnectEvt)
106. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF, OnScmDisconnectCnf)
107. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF_NEG, OnScmDisconnectCnfNeg)
108. ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF, OnScmDeRegisterCnf)
109. ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF_NEG, OnScmDeRegisterCnfNeg)
110. ON_BLUETOOTH_EVENT(SD_START_CNF, OnSdStartCnf)
111. ON_BLUETOOTH_EVENT(SD_CONNECT_CNF, OnSdConnectCnf)
112. ON_BLUETOOTH_EVENT(SD_CONNECT_CNF_NEG, OnSdConnectCnfNeg)
113. ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF, OnSdServiceSearchCnf)
114. ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF_NEG, OnSdServiceSearchCnfNeg)
115. ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF, OnSdServiceAttributeCnf)
116. ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF_NEG,
    OnSdServiceAttributeCnfNeg)
117. ON_BLUETOOTH_EVENT(SD_DISCONNECT_CNF, OnSdDisconnectCnf)
118. ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF, OnDbmRegisterServiceCnf)
119. ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF_NEG,
    OnDbmRegisterServiceCnfNeg)
120. ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF, OnDbmUnRegisterServiceCnf)
121. ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF_NEG,
    OnDbmUnRegisterServiceCnfNeg)
122. ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF, OnDbmAddDescriptorCnf)
123. ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF_NEG, OnDbmAddDescriptorCnfNeg)
124. ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF, OnHciConfigurePortConfirm)
125. ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF_NEG,
    OnHciConfigurePortConfirmNegative)
126. ON_BLUETOOTH_EVENT(HCI_INQUIRY_CNF, OnHciInquiryCnf)
127. ON_BLUETOOTH_EVENT(HCI_INQUIRY_EVT, OnHciInquiryEvt)
128. ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF, OnHciLocalAddressCnf)
129. ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF_NEG, OnHciLocalAddressCnfNeg)
130. ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF, OnHciRemoteNameCnf)
131. ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF_NEG, OnHciRemoteNameCnfNeg)
132. ON_BLUETOOTH_EVENT(HCI_START_CNF, OnHciStartCnf)
133. ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF, OnHciWriteScanEnableCnf)
134. ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF_NEG,
    OnHciWriteScanEnableCnfNeg)
135.
136.
137. ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF,
    OnHciWriteAuthenticationModeCnf)
138. ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG,
    OnHciWriteAuthenticationModeCnfNeg)
139. ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF,
    OnHciWriteEncryptionModeCnf)
140. ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF_NEG,
    OnHciWriteEncryptionModeCnfNeg)
141. ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF, OnHciWriteCodCnf)
142. ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF_NEG, OnHciWriteCodCnfNeg)
143. ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF, OnHciWriteNameCnf)
144. ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF_NEG, OnHciWriteNameCnfNeg)
145. ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF,
    OnHciWriteConnectTimeoutCnf)
146. ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG,
    OnHciWriteConnectTimeoutCnfNeg)
147. ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF, OnHciWritePageTimeoutCnf)

```

```

148. ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF_NEG,
                       OnHciWritePageTimeoutCnfNeg)
149. ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF, OnSilSetDeviceCnf)
150. ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF_NEG, OnSilSetDeviceCnfNeg)
151. ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF, OnSilReqDeviceCnf)
152. ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF_NEG, OnSilReqDeviceCnfNeg)
153. ON_BLUETOOTH_EVENT(COM_CONNECT_CNF, OnComConnectCnf )
154. ON_BLUETOOTH_EVENT(COM_CONNECT_CNF_NEG, OnComConnectCnfNeg )
155. ON_BLUETOOTH_EVENT(COM_DATA_IND, OnComDataInd )
156. ON_BLUETOOTH_EVENT(COM_DATA_CNF, OnComDataCnf )
157. ON_BLUETOOTH_EVENT(COM_DATA_CNF_NEG, OnComDataCnfNeg )
158. ON_BLUETOOTH_EVENT(COM_DISCONNECT_EVT, OnComDisconnectEvt )
159. ON_BLUETOOTH_EVENT(COM_DISCONNECT_CNF, OnComDisconnectCnf )
160. ON_BLUETOOTH_EVENT(COM_DISCONNECT_CNF_NEG, OnComDisconnectCnfNeg )
161. ON_BN_CLICKED(IDC_BUTTON1, OnBrowse)
162. //}}AFX_MSG_MAP
163. END_MESSAGE_MAP()
164. BOOL CRadioFileClientDlg::OnInitDialog()
165. {
166. CDialog::OnInitDialog();
167. ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
168. ASSERT(IDM_ABOUTBOX < 0xF000);
169. CMenu* pSysMenu = GetSystemMenu(FALSE);
170. if (pSysMenu != NULL)
171. {
172. CString strAboutMenu;
173. strAboutMenu.LoadString(IDS_ABOUTBOX);
174. if (!strAboutMenu.IsEmpty())
175. {
176. pSysMenu->AppendMenu(MF_SEPARATOR);
177. pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
178. }
179. }
180. SetIcon(m_hIcon, TRUE); // Set big icon
181. SetIcon(m_hIcon, FALSE); // Set small icon
182. m_pServerEvents->m_pParentDialog = this;
183. TVINSERTSTRUCT tvInsert;
184. tvInsert.hParent = NULL;
185. tvInsert.hInsertAfter = NULL;
186. tvInsert.item.mask = TVIF_TEXT;
187. tvInsert.item.pszText = _T("RemoteRadios");
188. hPA = m_tree.InsertItem(&tvInsert);
189. index = 0;
190. InitSecurityClient();
191. return TRUE;
192. }
193. LRESULT CRadioFileClientDlg::WindowProc(UINT message, WPARAM wParam,
                                           LPARAM lParam)
194. {
195. MSG_TMsg **ptMsg;
196. if (message == WM_BLUETOOTH_EVENT)
197. {
198. OnBluetoothEvent( message, wParam, lParam);
199. ptMsg = (MSG_TMsg**)lParam;
200. if (*ptMsg != NULL)
201. VOS_Free((void **)lParam);

```

```

202.     }
203.     return CDialog::WindowProc(message, wParam, lParam);
204. }
205. BOOL CRadioFileClientDlg::OnBluetoothEvent(UINT message, WPARAM wParam,
        LPARAM lParam)
206. {
207.     const AFX_MSGMAP* pMessageMap;
208.     const AFX_MSGMAP_ENTRY* lpEntry;
209.     #ifdef _AFXDLL
210.         for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
211.             pMessageMap = (*pMessageMap->pfnGetBaseMap)())
212.             #else
213.             for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
214.                 pMessageMap = pMessageMap->pBaseMap)
215.                 #endif
216.         {
217.             #ifdef _AFXDLL
218.                 ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
219.             #else
220.                 ASSERT(pMessageMap != pMessageMap->pBaseMap);
221.             #endif
222.             lpEntry = (AFX_MSGMAP_ENTRY*)(&pMessageMap->lpEntries[0]);
223.             while (lpEntry->nSig != AfxSig_end)
224.             {
225.                 if ((lpEntry->nMessage == message) && (lpEntry->nCode == wParam))
226.                 {
227.                     union MessageMapFunctions mmf;
228.                     mmf.pfn = lpEntry->pfn;
229.                     (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
230.                     return TRUE;
231.                 }
232.                 lpEntry++;
233.             }
234.             return FALSE;
235.         }
236.     return FALSE;
237. }
238. void CRadioFileClientDlg::OnSysCommand(UINT nID, LPARAM lParam)
239. {
240.     if ((nID & 0xFFFF) == IDM_ABOUTBOX)
241.     {
242.         CAboutDlg dlgAbout;
243.         dlgAbout.DoModal();
244.     }
245.     else
246.     {
247.         CDialog::OnSysCommand(nID, lParam);
248.     }
249. }
250. void CRadioFileClientDlg::OnPaint()
251. {
252.     if (IsIconic())
253.     {
254.         CPaintDC dc(this);
255.         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
256.         int cxIcon = GetSystemMetrics(SM_CXICON);

```



```

257.     int cyIcon = GetSystemMetrics(SM_CYICON);
258.     CRect rect;
259.     GetClientRect(&rect);
260.     int x = (rect.Width() - cxIcon + 1) / 2;
261.     int y = (rect.Height() - cyIcon + 1) / 2;
262.     dc.DrawIcon(x, y, m_hIcon);
263. }
264. else
265. {
266.     CDialog::OnPaint();
267. }
268. }
269. HCURSOR CRadioFileClientDlg::OnQueryDragIcon()
270. {
271.     return (HCURSOR) m_hIcon;
272. }
273. void CRadioFileClientDlg::InitSecurityClient()
274. {
275.     SIL_SetDevice(0, SIL_SERIAL);
276. }
277. void CRadioFileClientDlg::OnSilSetDeviceCnf(void **ppMsg)
278. {
279.     ppMsg = ppMsg;
280.     HCI_ReqConfigurePort(0, PORTSETTINGS);
281. }
282. void CRadioFileClientDlg::OnSilSetDeviceCnfNeg(void **ppMsg)
283. {
284.     SIL_TSetDevice* ptSetDevice;
285.     ptSetDevice = (SIL_TSetDevice*) *ppMsg;
286.     if(ptSetDevice->tHdr.iResult == SIL_ERR_DEVICE)
287.     SIL_ReqDevice(0);
288. }
289. void CRadioFileClientDlg::OnSilReqDeviceCnf(void **ppMsg)
290. {
291.     SIL_TReqDevice* ptReq;
292.     ptReq = (SIL_TReqDevice*) *ppMsg;
293.     if(ptReq->uiDevice == SIL_SERIAL)
294.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
                                Interface is SERIAL"));
295.     if(ptReq->uiDevice == SIL_USB)
296.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
                                Interface is USB"));
297. }
298. void CRadioFileClientDlg::OnSilReqDeviceCnfNeg(void **ppMsg)
299. {
300.     ppMsg = ppMsg;
301.     MessageBox(_T("Device Request FAILED!"));
302. }
303. void CRadioFileClientDlg::OnHciConfigurePortConfirm(void **ppMsg)
304. {
305.     HCI_TConfigurePortCnf *tConfigurePort = (HCI_TConfigurePortCnf
                                                *) *ppMsg;
306.     tConfigurePort = tConfigurePort;
307.     COM_ReqStart(0);
308. }
309. void CRadioFileClientDlg::OnHciConfigurePortConfirmNegative(void **ppMsg)

```

```

310. {
311.     HCI_TConfigurePortCnfNeg *tConfigurePort = (HCI_TConfigurePortCnfNeg
                                                    *) *ppMsg;
312.     tConfigurePort = tConfigurePort;
313.     MessageBox(_T("Could not open port"));
314. }
315. void CRadioFileClientDlg::OnComStartCnf(void **ppMsg)
316. {
317.     COM_TStartCnf *tStartCnf = (COM_TStartCnf *) *ppMsg;
318.     tStartCnf = tStartCnf;
319.     HCI_ReqLocalAddress(0);
320. }
321. void CRadioFileClientDlg::OnComStartCnfNeg(void **ppMsg)
322. {
323.     COM_TStartCnfNeg *tStartCnfNeg = (COM_TStartCnfNeg *) *ppMsg;
324.     tStartCnfNeg = tStartCnfNeg;
325.     MessageBox(_T("Could not start RFCOMM"));
326. }
327. void CRadioFileClientDlg::OnHciLocalAddressCnf(void **ppMsg)
328. {
329.     HCI_TLocalAddressCnf *tLocalAddress = (HCI_TLocalAddressCnf *) *ppMsg;
330.     char lpStr[59];
331.     wsprintf(&lpStr[0], "BD_ADDRESS: 0x%02X%02X%02X%02X%02X%02X\0",
332.             tLocalAddress->tAddress.ucByte0,
333.             tLocalAddress->tAddress.ucByte1,
334.             tLocalAddress->tAddress.ucByte2,
335.             tLocalAddress->tAddress.ucByte3,
336.             tLocalAddress->tAddress.ucByte4,
337.             tLocalAddress->tAddress.ucByte5);
338.     SetWindowText(_T(lpStr));
339.     SD_ReqStart(0);
340. }
341. void CRadioFileClientDlg::OnHciLocalAddressCnfNeg(void **ppMsg)
342. {
343.     ppMsg = ppMsg;
344.     SetWindowText(_T("DEVICE NOT FOUND"));
345.     SD_ReqStart(0);
346. }
347. void CRadioFileClientDlg::OnSdStartCnf(void **ppMsg)
348. {
349.     ppMsg = ppMsg;
350.     HCI_ReqWriteEncryptionMode(0, HCI_ENCRYPTION_OFF);
351. }
352. void CRadioFileClientDlg::OnHciWriteEncryptionModeCnf(void **ppMsg)
353. {
354.     ppMsg = ppMsg;
355.     HCI_ReqWriteAuthenticationMode(0, HCI_AUTH_DISABLE);
356. }
357. void CRadioFileClientDlg::OnHciWriteEncryptionModeCnfNeg(void **ppMsg)
358. {
359.     ppMsg = ppMsg;
360. }
361. void CRadioFileClientDlg::OnHciWriteAuthenticationModeCnf(void **ppMsg)
362. {
363.     ppMsg = ppMsg;
364.     HCI_ReqWriteConnectTimeout(0, 0x1FA0);

```

```

365. }
366. void CRadioFileClientDlg::OnHciWriteAuthenticationModeCnfNeg(void
                                     **ppMsg)
367. {
368.     ppMsg = ppMsg;
369. }
370. void CRadioFileClientDlg::OnHciWriteConnectTimeoutCnf(void **ppMsg)
371. {
372.     ppMsg = ppMsg;
373.     HCI_ReqWritePageTimeout(0,8000);
374. }
375. void CRadioFileClientDlg::OnHciWriteConnectTimeoutCnfNeg(void **ppMsg)
376. {
377.     ppMsg = ppMsg;
378. }
379. void CRadioFileClientDlg::OnHciWritePageTimeoutCnf(void **ppMsg)
380. {
381.     ppMsg = ppMsg;
382.
383.     HCI_ReqWriteCod(0,_tCod);
384.
385.
386.
387. }
388. void CRadioFileClientDlg::OnHciWritePageTimeoutCnfNeg(void **ppMsg)
389. {
390.     ppMsg = ppMsg;
391. }
392.
393.
394.
395.
396.
397.
398.
399.
400.
401. void CRadioFileClientDlg::OnHciWriteCodCnf(void **ppMsg)
402. {
403.     ppMsg = ppMsg;
404.     HCI_ReqWriteName (0,(HCI_TName*) "BT File");
405. }
406. void CRadioFileClientDlg::OnHciWriteCodCnfNeg(void **ppMsg)
407. {
408.     ppMsg = ppMsg;
409. }
410. void CRadioFileClientDlg::OnHciWriteNameCnf(void **ppMsg)
411. {
412.     ppMsg = ppMsg;
413.     HCI_ReqWriteScanEnable(0,HCI_PAGE_SCAN_ENABLED |
                                     HCI_INQUIRY_SCAN_ENABLED);
414. }
415. void CRadioFileClientDlg::OnHciWriteNameCnfNeg(void **ppMsg)
416. {
417.     ppMsg = ppMsg;
418. }

```

```

419. void CRadioFileClientDlg::OnHciWriteScanEnableCnf(void **ppMsg)
420. {
421.     ppMsg = ppMsg;
422.     SCM_ReqRegister(0, SCM_SECURITY_HANDLER);
423. }
424. void CRadioFileClientDlg::OnHciWriteScanEnableCnfNeg(void **ppMsg)
425. {
426.     ppMsg = ppMsg;
427. }
428. void CRadioFileClientDlg::OnScmRegisterCnf(void **ppMsg)
429. {
430.     SCM_TRegisterCnf *tRegisterCnf = (SCM_TRegisterCnf *)*ppMsg;
431.     tRegisterCnf = tRegisterCnf;
432.     if (tRegisterCnf->tHdr.uiSeqNr == 0)
433.     {
434.         SCM_ReqRegister(1, SCM_MONITOR_GROUP);
435.     }
436.     else
437.     {
438.         OnInquiry();
439.     }
440. }
441. void CRadioFileClientDlg::OnScmRegisterCnfNeg(void **ppMsg)
442. {
443.     SCM_TRegisterCnfNeg *tRegisterCnfNeg = (SCM_TRegisterCnfNeg *)*ppMsg;
444.     tRegisterCnfNeg = tRegisterCnfNeg;
445.     MessageBox(_T("Could not register to SCM"));
446. }
447. void CRadioFileClientDlg::OnHciInquiryCnf(void **ppMsg)
448. {
449.     HCI_TInquiryCnf      *ptInquiryCnf;
450.     int count;
451.     CDevice device;
452.     ptInquiryCnf = (HCI_TInquiryCnf *) *ppMsg;
453.     count = m_DevicesFound.GetSize();
454.     m_RemoteNameCounter = 0;
455.     if (count > 0)
456.     {
457.         device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
458.         HCI_ReqRemoteName(10,
459.             device.tAddress,
460.             device.tPageScanPeriodMode,
461.             device.tPageScanMode,
462.             device.tClockOffset );
463.     }
464.     else
465.     {
466.         AfxMessageBox("No device found");
467.     }
468. }
469. void CRadioFileClientDlg::OnHciRemoteNameCnf(void **ppMsg)
470. {
471.     HCI_TRemoteNameCnf    *ptRemoteNameCnf;
472.     CDevice device;
473.     char sName[248];
474.     int count;

```

```

475.     ptRemoteNameCnf =(HCI_TRemoteNameCnf *) *ppMsg;
476.     sprintf(sName, "%s", &ptRemoteNameCnf->tName);
477.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)sName);
478.     m_RemoteNameCounter++;
479.     count = m_DevicesFound.GetSize();
480.     if (count > m_RemoteNameCounter)
481.     {
482.         device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
483.         HCI_ReqRemoteName(10,
484.             device.tAddress,
485.             device.tPageScanPeriodMode,
486.             device.tPageScanMode,
487.             device.tClockOffset );
488.     }
489.     else
490.     {
491.         ShowAllDevicesFound();
492.     }
493. }
494. void CRadioFileClientDlg::OnHciRemoteNameCnfNeg(void **ppMsg)
495. {
496.     HCI_TRemoteNameCnfNeg *ptRemoteNameCnfNeg;
497.     CDevice device;
498.     int count;
499.     ptRemoteNameCnfNeg =(HCI_TRemoteNameCnfNeg *) *ppMsg;
500.     m_DevicesFound[m_RemoteNameCounter].SetName ((CString)_T("UNKNOWN"));
501.     m_RemoteNameCounter++;
502.     count = m_DevicesFound.GetSize();
503.     if (count > m_RemoteNameCounter)
504.     {
505.         device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
506.         HCI_ReqRemoteName(10,
507.             device.tAddress,
508.             device.tPageScanPeriodMode,
509.             device.tPageScanMode,
510.             device.tClockOffset );
511.     }
512.     else
513.     {
514.         ShowAllDevicesFound();
515.     }
516. }
517. void CRadioFileClientDlg::OnScmConnectCnf(void **ppMsg)
518. {
519.     SCM_TConnectCnf *tConnectCnf = (SCM_TConnectCnf *)*ppMsg;
520.     AfxMessageBox("connected");
521.     tConnectCnf = tConnectCnf;
522.     m_ConnectionInfo.tAclHandle = tConnectCnf->tHandle;
523.     m_ConnectionInfo.tAddress = tConnectCnf->tAddress;
524.     OnGetServices();
525. }
526. void CRadioFileClientDlg::OnScmConnectCnfNeg(void **ppMsg)
527. {
528.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
529.     tConnectCnfNeg = tConnectCnfNeg;
530.     AfxMessageBox("No Connection made");

```

```

531. }
532. void CRadioFileClientDlg::OnSdConnectCnf(void **ppMsg)
533. {
534.     SD_TConnectCnf *tConnectCnf = (SD_TConnectCnf *)*ppMsg;
535.     SD_TUuid          *ptSearchPatternList;
536.     uint16             uiMaxRecords;
537.     uint8              ucNrOfUuids;
538.     m_ConnectionInfo.uiSdcHandle = tConnectCnf->uiSdcHandle;
539.     uiMaxRecords = 6;
540.     ucNrOfUuids = 1;
541.     ptSearchPatternList = (SD_TUuid*)VOS_Alloc((uint16)(ucNrOfUuids *
        sizeof(SD_TUuid)));
542.     ptSearchPatternList[0].eUuidType = SD_DET_UUID16;
543.     ptSearchPatternList[0].TUuid.uiUuid16 =
        SRP_SERIAL_GENERIC_SERIALPORT_UUID ;
544.     SD_ReqServiceSearch (0, m_ConnectionInfo.uiSdcHandle, uiMaxRecords,
        ucNrOfUuids, ptSearchPatternList);
545. }
546. void CRadioFileClientDlg::OnSdConnectCnfNeg(void **ppMsg)
547. {
548.     SD_TConnectCnfNeg *tConnectCnfNeg = (SD_TConnectCnfNeg *)*ppMsg;
549.     CString str;
550.     str.Format("Could not connect to SD , Error %d",tConnectCnfNeg-
        >tHdr.iResult);
551.     MessageBox(str);
552. }
553. void CRadioFileClientDlg::OnSdServiceSearchCnf(void **ppMsg)
554. {
555.     SD_TServiceSearchCnf *tServiceSearchCnf = (SD_TServiceSearchCnf
        *)*ppMsg;
556.     uint16             uiCurrentServiceRecordCount;
557.     uint32             *pulSRHandles;
558.     uint16             *puiAttributeIDList;
559.     uint8              ucNrOfAttr;
560.     CService          service;
561.     uiCurrentServiceRecordCount = tServiceSearchCnf-
        >uiCurrentServiceRecordCount;
562.     pulSRHandles = (uint32*)VOS_Alloc(((uint16)
        (uiCurrentServiceRecordCount*sizeof(uint32))));
563.     (void*)memcpy(pulSRHandles,
        &tServiceSearchCnf->ulServiceRecordHandleList,
564.         (uiCurrentServiceRecordCount*sizeof(uint32)));
565.     m_ConnectionInfo.ulServiceRecordHandle = tServiceSearchCnf-
        >ulServiceRecordHandleList;
566.     ucNrOfAttr = 1;
567.     puiAttributeIDList = (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof
        (uint16)));
568.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
569.     service.m_SDCHandle = m_ConnectionInfo.uiSdcHandle;
570.     service.m_ServiceRecordHandle = tServiceSearchCnf-
        >ulServiceRecordHandleList;
571.     m_ServicesFound.SetAtGrow(m_ServiceCounter,service);
572.     SD_ReqServiceAttribute(1, m_ConnectionInfo.uiSdcHandle,
        pulSRHandles[0], ucNrOfAttr, puiAttributeIDList);
573.     VOS_Free((void*)&puiAttributeIDList);
574.     VOS_Free((void*)&pulSRHandles);

```

```

576. }
577. void CRadioFileClientDlg::OnSdServiceSearchCnfNeg(void **ppMsg)
578. {
579.     SD_TServiceSearchCnfNeg *tConnectCnfNeg = (SD_TServiceSearchCnfNeg
                                                *)*ppMsg;
580.     CString str;
581.     tConnectCnfNeg = tConnectCnfNeg;
582.     str.Format("Service Search Confirm Negative, Error %d",tConnectCnfNeg-
                >tHdr.iResult);
583.     MessageBox(str);
584. }
585. void CRadioFileClientDlg::OnSdServiceAttributeCnf(void **ppMsg)
586. {
587.     SD_TServiceAttributeCnf *tServiceAttributeCnf =
                (SD_TServiceAttributeCnf *)*ppMsg;
588.     CService service;
589.     switch (tServiceAttributeCnf->tHdr.uiSeqNr)
590.     {
591.     case 1:
592.         ReceiveServiceName(tServiceAttributeCnf);
593.         AskForServiceRecordHandle();
594.         AfxMessageBox("Service Attribute Cnf");
595.         break;
596.     case 2:
597.         ReceiveServiceRecordHandle(tServiceAttributeCnf);
598.         AfxMessageBox("SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle)");
599.         SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle);
600.         break;
601.     default:
602.         break;
603.     }
604. }
605. void CRadioFileClientDlg::OnSdServiceAttributeCnfNeg(void **ppMsg)
606. {
607.     SD_TServiceAttributeCnfNeg *tConnectCnfNeg =
                (SD_TServiceAttributeCnfNeg *)*ppMsg;
608.     CString str;
609.     tConnectCnfNeg = tConnectCnfNeg;
610.     str.Format("Service Attribute Confirm negative, error
                %d",tConnectCnfNeg->tHdr.iResult);
611.     MessageBox(str);
612. }
613. void CRadioFileClientDlg::OnSdDisconnectCnf(void **ppMsg)
614. {
615.     ppMsg = ppMsg;
616.     Beep (1000,200);
617.     OnSelservices();
618. }
619. void CRadioFileClientDlg::OnDbmRegisterServiceCnf(void **ppMsg)
620. {
621.     uint16                uiDescriptorUuidValue;
622.     DBM_TDescriptorValue  tDescriptorValue;
623.     DBM_TDescriptorUuid   tDescriptor;
624.     DBM_TRegisterServiceCnf *tRegisterCnf = (DBM_TRegisterServiceCnf
                                                *)*ppMsg;
625.     m_ConnectionInfo.ulDbmHandle = tRegisterCnf->ulDbmHandle;

```

```

626.     uiDescriptorUuidValue           = BT_PSM_COM;
627.     tDescriptor.tType                = DBM_DET_UUID16;
628.     tDescriptor.pucDescriptorUuidValue = (uint8*) &uiDescriptorUuidValue;
629.     tDescriptorValue.uiNrOfParams    = 1;
630.     tDescriptorValue.uiSizeOfValueInBytes = 2;
631.     tDescriptorValue.pucValue        = (uint8 *) VOS_Alloc( (sizeof(uint16)) );
632.     *tDescriptorValue.pucValue       = DBM_DET_UINT8;
633.     tDescriptorValue.pucValue++;
634.     *tDescriptorValue.pucValue      = m_ConnectionInfo.pucAttributeData
                                     [m_ConnectionInfo.uiAttributeListByteCount - 1];
635.     tDescriptorValue.pucValue--;
636.     DBM_ReqAddDescriptor(0,
637.                          m_ConnectionInfo.ulDbmHandle,
638.                          BT_PROTOCOL_DESCRIPTOR_LIST,
639.                          &tDescriptor,
640.                          &tDescriptorValue);
641.     VOS_Free((void**) &tDescriptorValue.pucValue);
642. }
643. void CRadioFileClientDlg::OnDbmRegisterServiceCnfNeg(void **ppMsg)
644. {
645.     ppMsg = ppMsg;
646.     MessageBox(_T("Not possible to Register to DBM"));
647.     SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle);
648. }
649. void CRadioFileClientDlg::OnDbmAddDescriptorCnf(void **ppMsg)
650. {
651.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
652.     tConnectCnfNeg = tConnectCnfNeg;
653.     OnConnect();
654.     Beep (1000,200);
655. }
656. void CRadioFileClientDlg::OnDbmAddDescriptorCnfNeg(void **ppMsg)
657. {
658.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
659.     tConnectCnfNeg = tConnectCnfNeg;
660.     MessageBox(_T("Could not register the service to DBM"));
661. }
662. void CRadioFileClientDlg::OnConnectAcceptInd(void **ppMsg)
663. {
664.     SCM_TConnectAcceptInd *ptConnectAcceptInd;
665.     ptConnectAcceptInd =(SCM_TConnectAcceptInd *) *ppMsg;
666.     SCM_RspConnectAccept( (MSG_TMsg **)ppMsg,
667.                          SCM_POS_RESULT,
668.                          ptConnectAcceptInd->tAddress,
669.                          SCM_SLAVE);
670.     *ppMsg = NULL;
671. }
672. void CRadioFileClientDlg::OnHciInquiryEvt(void **ppMsg)
673. {
674.     HCI_TInquiryEvt      *ptInquiryEvt;
675.     CDevice device;
676.     ptInquiryEvt =(HCI_TInquiryEvt *) *ppMsg;
677.     device.tAddress = ptInquiryEvt->tAddress;
678.     device.tPageScanMode = ptInquiryEvt->tPageScanMode;
679.     device.tPageScanPeriodMode = ptInquiryEvt->tPageScanPeriodMode;
680.     device.tClockOffset = ptInquiryEvt->tClockOffset;

```



```

681.     device.tCod = ptInquiryEvt->tCod;
682.     device.tPageScanRepMode = ptInquiryEvt->tPageScanRepMode;
683.     AddDevice(device);
684. }
685. void CRadioFileClientDlg::OnScmPincodeInd(void **ppMsg)
686. {
687.     SCM_TPINcodeInd      *ptPincodeInd;
688.     ptPincodeInd = (SCM_TPINcodeInd *) *ppMsg;
689.     SCM_RspPincode( (MSG_TMsg **)ppMsg,
690.                     SCM_POS_RESULT,
691.                     ptPincodeInd->tAddress,
692.                     _tPincode,
693.                     PINCODE_LENGTH);
694. }
695. void CRadioFileClientDlg::OnScmConnectEvt(void **ppMsg)
696. {
697.     SCM_TConnectEvt *tConnectEvt = (SCM_TConnectEvt *)*ppMsg;
698.     tConnectEvt = tConnectEvt;
699.     m_ConnectionInfo.tAclHandle = tConnectEvt->tHandle;
700.     m_ConnectionInfo.tAddress = tConnectEvt->tAddress;
701. }
702. void CRadioFileClientDlg::OnScmDisconnectEvt(void **ppMsg)
703. {
704.     ppMsg = ppMsg;
705.     m_ConnectionInfo.tAclHandle = 0;
706.     OnCloseapplication();
707. }
708. void CRadioFileClientDlg::OnHciStartCnf(void **ppMsg)
709. {
710.     HCI_TStartCnf *ptStartCnf = (HCI_TStartCnf *)*ppMsg;
711.     ptStartCnf = ptStartCnf;
712.     HCI_ReqConfigurePort(0,PORTSETTINGS);
713. }
714. void CRadioFileClientDlg::OnComVersionCnf(void **ppMsg)
715. {
716.     CAboutDlg Abodlg;
717.     COM_TVersionCnf* ptVersionCnf;
718.     char* cpVerStr = NULL;
719.     int8 iCharCount = 9;
720.     char cpStr[3];
721.     ptVersionCnf = (COM_TVersionCnf *) *ppMsg;
722.     cpVerStr = &ptVersionCnf->cVersion;
723.     do
724.     {
725.         iCharCount++;
726.         cpStr[iCharCount-10] = cpVerStr[iCharCount];
727.     }while(iCharCount <= 11);
728.     cpStr[3] = ((char)0);
729.     Abodlg.DoModal();
730. }
731. void CRadioFileClientDlg::AskForServiceName()
732. {
733.     uint16          *puiAttributeIDList;
734.     uint8           ucNrOfAttr;
735.     ucNrOfAttr = 1;

```

```

736.     puiAttributeIDList = (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof
                                         (uint16)));
737.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
738.     SD_ReqServiceAttribute(0, m_ConnectionInfo.uiSdcHandle,
        m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr, puiAttributeIDList);
739.     VOS_Free((void*)&puiAttributeIDList);
740. }
741. void CRadioFileClientDlg::ReceiveServiceName(SD_TServiceAttributeCnf
        *tServiceAttributeCnf)
742. {
743.     CService service;
744.     service = m_ServicesFound.GetAt(m_ServiceCounter);
745.     service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
746.     service.m_AttributeListByteCount = tServiceAttributeCnf-
        >uiAttributeListByteCount;
747.     (void*)memcpy(service.m_pAttributeData,
748.         &tServiceAttributeCnf->ucAttributeData,
749.         service.m_AttributeListByteCount);
750.     (void*)memcpy(service.m_pServiceName,
751.         &service.m_pAttributeData[7],
752.         service.m_pAttributeData[6]);
753.     service.m_pServiceName[service.m_pAttributeData[6]] = NULL;
754.     m_ServicesFound.SetAt(m_ServiceCounter, service);
755.     m_ServiceCounter++;
756.     m_ConnectionInfo.uiAttributeListByteCount =
        tServiceAttributeCnf->uiAttributeListByteCount;
757.     (void*)memcpy(m_ConnectionInfo.pucAttributeData,
758.         &tServiceAttributeCnf->ucAttributeData,
759.         m_ConnectionInfo.uiAttributeListByteCount);
760.     (void*)memcpy(m_ConnectionInfo.pcServiceName,
761.         &service.m_pAttributeData[7],
762.         service.m_pAttributeData[6]);
763.     m_ConnectionInfo.pcServiceName[service.m_pAttributeData[6]] = NULL;
764.     ShowAllServicesFound();
765. }
766. void CRadioFileClientDlg::AskForServiceRecordHandle()
767. {
768.     uint16          *puiAttributeIDList;
769.     uint8           ucNrOfAttr;
770.     ucNrOfAttr = 2;
771.     puiAttributeIDList = (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof
                                         (uint16)));
772.     puiAttributeIDList[0] = BT_SERVICE_RECORD_HANDLE;
773.     puiAttributeIDList[1] = BT_PROTOCOL_DESCRIPTOR_LIST;
774.     SD_ReqServiceAttribute(2, m_ConnectionInfo.uiSdcHandle,
        m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr,
        puiAttributeIDList);
775.     VOS_Free((void*)&puiAttributeIDList);
776. }
777. void CRadioFileClientDlg::ReceiveServiceRecordHandle
        (SD_TServiceAttributeCnf *tServiceAttributeCnf)
778. {
779.     CService service;
780.     service = m_ServicesFound.GetAt(m_ServiceCounter-1);
781.     service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;

```

```

782.     service.m_AttributeListByteCount = tServiceAttributeCnf-
                                           >uiAttributeListByteCount;
783.     (void*)memcpy(service.m_pAttributeData,
784.                   &tServiceAttributeCnf->ucAttributeData,
785.                   service.m_AttributeListByteCount);
786.     m_ServicesFound.SetAt(m_ServiceCounter-1,service);
787.     m_ServiceCounter++;
788.     m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf -
                                           >uiAttributeListByteCount;
789.     (void*)memcpy(m_ConnectionInfo.pucAttributeData,
790.                   &tServiceAttributeCnf->ucAttributeData,
791.                   m_ConnectionInfo.uiAttributeListByteCount);
792. }
793. void CRadioFileClientDlg::OnCloseapplication()
794. {
795.     SCM_ReqDeRegister(1,SCM_SECURITY_HANDLER);
796. }
797. void CRadioFileClientDlg::OnScmDeRegisterCnf(void **ppMsg)
798. {
799.     SCM_TDeRegisterCnf *ptDeRegisterCnf = (SCM_TDeRegisterCnf *) *ppMsg;
800.     switch (ptDeRegisterCnf->tHdr.uiSeqNr)
801.     {
802.     case 1:
803.         SCM_ReqDeRegister(2,SCM_MONITOR_GROUP);
804.         break;
805.     case 2:
806.         if (m_ConnectionInfo.ulDbmHandle > 0)
807.         {
808.             DBM_ReqUnRegisterService(3,m_ConnectionInfo.ulDbmHandle);
809.         }
810.         else
811.         {
812.             if (m_ConnectionInfo.tAclHandle>0)
813.             {
814.                 SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
815.             }
816.             else
817.             {
818.                 DestroyWindow();
819.             }
820.         }
821.         break;
822.     default:
823.         break;
824.     }
825. }
826. void CRadioFileClientDlg::OnScmDeRegisterCnfNeg(void **ppMsg)
827. {
828.     ppMsg = ppMsg;
829.     MessageBox(_T("Could not unregister from SCM"));
830.     DestroyWindow();
831. }
832. void CRadioFileClientDlg::OnDbmUnRegisterServiceCnf(void **ppMsg)
833. {
834.     ppMsg = ppMsg;
835.     if (m_ConnectionInfo.tAclHandle>0)

```

```

836.     {
837.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
838.     }
839.     else
840.     {
841.         DestroyWindow();
842.     }
843. }
844. void CRadioFileClientDlg::OnDbmUnRegisterServiceCnfNeg(void **ppMsg)
845. {
846.     ppMsg = ppMsg;
847.     MessageBox(_T("Not possible to UnRegister from DBM"));
848.     DestroyWindow();
849. }
850. void CRadioFileClientDlg::OnScmDisconnectCnf(void **ppMsg)
851. {
852.     ppMsg = ppMsg;
853.     m_ConnectionInfo.tAclHandle = 0;
854.     DestroyWindow();
855. }
856. void CRadioFileClientDlg::OnScmDisconnectCnfNeg(void **ppMsg)
857. {
858.     ppMsg = ppMsg;
859.     MessageBox(_T("Could not remove ACL connection"));
860.     DestroyWindow();
861. }
862. BOOL CRadioFileClientDlg::DestroyWindow()
863. {
864.     return CDialog::DestroyWindow();
865. }
866. void CRadioFileClientDlg::ShowAllDevicesFound()
867. {
868.     CDevice device;
869.     int iFound,i;
870.     iFound = m_DevicesFound.GetSize();
871.     for (i=0; i < iFound; i++)
872.     {
873.         device = m_DevicesFound.GetAt(i);
874.         AfxMessageBox("device1");
875.         hdevice1=m_tree.InsertItem(device.GetAddress(), hPA, TVI_SORT);
876.         OnSelDevice();
877.     }
878. }
879. void CRadioFileClientDlg::AddService(CString sService)
880. {
881.     CService service(sService);
882.     m_ServicesFound.Add(service);
883. }
884. void CRadioFileClientDlg::AddService(CService service)
885. {
886.     m_ServicesFound.Add(service);
887. }
888. void CRadioFileClientDlg::ShowAllServicesFound()
889. {
890.     CService service;
891.     int iFound,i;

```

```

892.     iFound = m_ServicesFound.GetSize();
893.     for (i=0; i < iFound; i++)
894.     {
895.         service = m_ServicesFound.GetAt(i);
896.         m_tree.InsertItem(service.GetService(),hdevice1,TVI_LAST);
897.     }
898. }
899. void CRadioFileClientDlg::AddDevice(CDevice device)
900. {
901.     m_DevicesFound.Add(device);
902. }
903. void CRadioFileClientDlg::OnInquiry()
904. {
905.     HCI_TLap    tLap = {0x9E,0x8B,0x33};
906.     HCI_TInquiryLength  tInquiryLength = 2;
907.     HCI_TNrOfResponses  tNrOfResponses = 0;
908.     HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses);
909. }
910. void CRadioFileClientDlg::OnSelDevice()
911. {
912.     CDevice device;
913.     device = m_DevicesFound.GetAt(0);
914.     m_ConnectionInfo.tAddress = device.tAddress;
915.     SCM_ReqConnect(0,
916.                   device.tAddress,
917.                   SCM_DM1,
918.                   SCM_R1,
919.                   SCM_MANDATORY_PAGE_SCAN_MODE,
920.                   0,
921.                   SCM_NOT_ACCEPT_ROLE_SWITCH);
922. }
923.
924. void CRadioFileClientDlg::OnSelServices()
925. {
926.     CService service;
927.     service = m_ServicesFound.GetAt(0);
928.     m_ConnectionInfo.ulServiceRecordHandle =
929.         service.m_ServiceRecordHandle;
929.     DBM_ReqRegisterService(0, DBM_StackDB);
930. }
931. void CRadioFileClientDlg::OnGetservices()
932. {
933.     m_ServiceCounter = 0;
934.     SD_ReqConnect(0,SD_DEFAULT_MFS,m_ConnectionInfo.tAclHandle);
935. }
936. void CRadioFileClientDlg::OnConnect()
937. {
938.     COM_ReqConnect(0,
939.                   (uint16)m_ConnectionInfo.ulDbmHandle,
940.                   m_ConnectionInfo.tAclHandle,
941.                   m_ConnectionInfo.uiMaxFrameSize);
942. }
943. void CRadioFileClientDlg::OnComConnectCnf(void **ppMsg)
944. {
945.     COM_TConnectCnf *ptConnectCnf = (COM_TConnectCnf *) *ppMsg;
946.

```

```

947.     m_ConnectionInfo.uiRFCCommHandle = ptConnectCnf->uiHandle;
948.     MessageBox(_T(" RFCOMM connection"));
949.     Beep (1000,200);
950.     Sleep(100);
951.     Beep (1000,200);
952. }
953. void CRadioFileClientDlg::OnComConnectCnfNeg(void **ppMsg)
954. {
955.     COM_TConnectCnfNeg *ptConnectCnfNeg = (COM_TConnectCnfNeg *) *ppMsg;
956.     ptConnectCnfNeg = ptConnectCnfNeg;
957.     m_ConnectionInfo.uiRFCCommHandle = 0;
958.     MessageBox(_T("Could not create a RFCOMM connection"));
959. }
960. void CRadioFileClientDlg::OnComDataInd(void **ppMsg)
961. {
962.     COM_TDataInd *tDataInd = (COM_TDataInd *)*ppMsg;
963.     uint8 *pucData;
964.     CHAR  sData[80];
965.     uint16 uiLength;
966.     uint16 uiHandle;
967.     int i;
968.     pucData = COM_DataExtract((MSG_TDataMsg *)*ppMsg,
969.                               &uiLength,
970.                               &uiHandle);
971.     COM_RspData(tDataInd->tHdr.ucSeqNr,COM_POS_RESULT,uiHandle);
972.     for (i=0; i < uiLength; i++)
973.     {
974.         sData[i] = pucData[i] ;
975.     }
976.     m_ChatArea.InsertString(index, (CString)sData);
977.     index++;
978. }
979. void CRadioFileClientDlg::OnComDataCnf(void **ppMsg)
980. {
981.     ppMsg = ppMsg;
982. }
983. void CRadioFileClientDlg::OnComDataCnfNeg(void **ppMsg)
984. {
985.     ppMsg = ppMsg;
986.     MessageBox(_T("Could not send data on RFCOMM channel"));
987. }
988. void CRadioFileClientDlg::OnComDisconnectEvt(void **ppMsg)
989. {
990.     COM_TDisconnectEvt *ptDisconnectEvt = (COM_TDisconnectEvt *)*ppMsg;
991.     ptDisconnectEvt = ptDisconnectEvt;
992.     m_ConnectionInfo.uiRFCCommHandle = 0;
993.     EndModalLoop(0);
994. }
995. void CRadioFileClientDlg::OnComDisconnectCnf(void **ppMsg)
996. {
997.     COM_TDisconnectCnf *ptDisconnectCnf = (COM_TDisconnectCnf *)*ppMsg;
998.     ptDisconnectCnf = ptDisconnectCnf;
999.     m_ConnectionInfo.uiRFCCommHandle = 0;
1000.     EndModalLoop(0);
1001. }
1002. void CRadioFileClientDlg::OnComDisconnectCnfNeg(void **ppMsg)

```

```

1003. {
1004.     COM_TDisconnectCnfNeg *ptDisconnectCnfNeg = (COM_TDisconnectCnfNeg
                                                *) *ppMsg;
1005.     ptDisconnectCnfNeg = ptDisconnectCnfNeg;
1006.     MessageBox(_T("Could not Disconnect the RFCOMM connection"));
1007. }
1008. void CRadioFileClientDlg::OnBrowse()
1009. {
1010.     CFile file,tFile,sFile;
1011.     CString fName,fPath;
1012.     int fnLength,fLength,i;
1013.     unsigned char *buffer,*sBuffer;
1014.     char cc[200],f1[1];
1015.     uint16 iCount=0;
1016.     uint8 *pucData;
1017.     tFile.Open("temp",CFile::modeCreate|CFile::modeWrite|
                CFile::modeRead,NULL);
1018.     CFileDialog dialog(1,"","*",OFN_OVERWRITEPROMPT|
                OFN_FILEMUSTEXIST,"All Files (*.*)");
1019.     GetCurrentDirectory(200,cc);
1020.     dialog.m_ofn.lpstrInitialDir= cc;
1021.     if(dialog.DoModal()==IDCANCEL)
1022.     {
1023.         return;
1024.     }
1025.
1026.     fName = dialog.GetFileName();
1027.     fPath = dialog.GetPathName();
1028.     m_InputChat.SetWindowText(fPath);
1029.     file.Open(fPath,CFile::modeRead,NULL);
1030.     fnLength = fName.GetLength();
1031.     itoa(fnLength,f1,10);
1032.     tFile.Write((void*)f1,sizeof(f1));
1033.     tFile.Write(fName,fnLength);
1034.     fLength=(int)file.GetLength();
1035.     buffer=(unsigned char *)malloc(fLength);
1036.     file.Read((void*)buffer,fLength);
1037.     tFile.SeekToEnd();
1038.     tFile.Write((void*)buffer,fLength);
1039.     file.Close();
1040.     tFile.Close();
1041.     SetCurrentDirectory(cc);
1042.     sFile.Open("temp",CFile::modeRead,NULL);
1043.     iCount=(uint16)sFile.GetLength();
1044.     sBuffer=(unsigned char *)malloc(iCount);
1045.     sFile.Read((void*)sBuffer,iCount);
1046.     if (iCount > 0)
1047.     {
1048.         pucData = COM_DataAlloc((uint16)iCount+1);
1049.         for (i=0;i < iCount; i++)
1050.         {
1051.             pucData[i]=sBuffer[i];
1052.         }
1053.         pucData[i]=0;
1054.         COM_DataSend(0,pucData,m_ConnectionInfo.uiRFCOMMHandle
                        ,(uint16)(iCount + 1));

```

```

1055.         CString str;
1056.         str.Format("%s Sent to Server", fName);
1057.         m_ChatArea.InsertString(index, (CString)str);
1058.         index++;
1059.         m_InputChat.SetWindowText(_T(""));
1060.         sFile.Close();
1061.         DeleteFile("temp");
1062.     }
1063. }

```

Code Description

- ◆ Line 1–15: The files required to implement `CRadioFileClientDlg` class are included.
- ◆ Lines 17–21: The VC++ editor includes these lines to provide a common framework for the MFC Application Wizard.
- ◆ Line 23: The variables `hPA` and `hdevice1` are required to implement a tree.
- ◆ Line 24–28: The union `MessageMapFunctions` is defined. It includes `pfn`, a pointer to `AFX_MSG_CALL` and the function call.
- ◆ Line 29: A constant is defined for PINCODE length.
- ◆ Line 30: Explained in `HCIInformationCommandsDlg.cpp`.
- ◆ Lines 31–32: Constants for serial port interface and USB interface are defined.
- ◆ Line 33: Constant for generic serial port ID is defined.
- ◆ Line 34: Constant for Bluetooth pincode is defined.
- ◆ Line 35: Constant for Security Manager's pincode.
- ◆ Lines 36–66: The About dialog is the default dialog to give the information about current application. It is provided with every VC++ MFC application.
- ◆ Lines 67–74: This is constructor framework provided by VC++ MFC application by putting the filename as class name. `m_pServerEvents` is an instance of Events class.
- ◆ Lines 75–80: The destructor is defined to free the memory for member variables and class references.
- ◆ Lines 81–89: The `DDX_Control` functions manage data transfer between dialog box controls and `CWnd` data members of the dialog box (see the following table).

<i>Dialog box control</i>	<i>CWnd Data member</i>
IDC_LIST1	m_ChatArea
IDC_EDIT1	m_InputChat
IDC_TREE1	m_tree

- ◆ Lines 90–163: The `ON_BLUETOOTH_EVENT` message map macro indicates which function handles a specified BLUETOOTH event. The important events and the handler function names as specified in the PC reference stack are listed in the following table.

<i>BLUETOOTH Event</i>	<i>Handler Function Name</i>
COM_START_CNF	OnComStartCnf
COM_START_CNF_NEG	OnComStartCnfNeg
COM_VERSION_CNF	OnComVersionCnf
SCM_REGISTER_CNF	OnScmRegisterCnf

SCM_REGISTER_CNF_NEG	OnScmRegisterCnfNeg
SCM_CONNECT_ACCEPT_IND	OnConnectAcceptInd
SCM_PINCODE_IND	OnScmPincodeInd
SCM_CONNECT_CNF	OnScmConnectCnf
SCM_CONNECT_CNF_NEG	OnScmConnectCnfNeg
SCM_CONNECT_EVT	OnScmConnectEvt
SCM_DISCONNECT_EVT	OnScmDisconnectEvt
SCM_DISCONNECT_CNF	OnScmDisconnectCnf
SCM_DISCONNECT_CNF_NEG	OnScmDisconnectCnfNeg
SCM_DEREGISTER_CNF	OnScmDeRegisterCnf
SCM_DEREGISTER_CNF_NEG	OnScmDeRegisterCnfNeg
SD_START_CNF	OnSdStartCnf
SD_CONNECT_CNF	OnSdConnectCnf
SD_CONNECT_CNF_NEG	OnSdConnectCnfNeg
SD_SERVICE_SEARCH_CNF	OnSdServiceSearchCnf
SD_SERVICE_SEARCH_CNF_NEG	OnSdServiceSearchCnfNeg
SD_SERVICE_ATTRIBUTE_CNF	OnSdServiceAttributeCnf
SD_SERVICE_ATTRIBUTE_CNF_NEG	OnSdServiceAttributeCnfNeg
SD_DISCONNECT_CNF	OnSdDisconnectCnf
DBM_REGISTER_SERVICE_CNF	OnDbmRegisterServiceCnf
DBM_REGISTER_SERVICE_CNF_NEG	OnDbmRegisterServiceCnfNeg
DBM_UNREGISTER_SERVICE_CNF	OnDbmUnRegisterServiceCnf
DBM_UNREGISTER_SERVICE_CNF_NEG	OnDbmUnRegisterServiceCnfNeg
DBM_ADD_DESCRIPTOR_CNF	OnDbmAddDescriptorCnf
DBM_ADD_DESCRIPTOR_CNF_NEG	OnDbmAddDescriptorCnfNeg
HCI_CONFIGURE_PORT_CNF	OnHciConfigurePortConfirm
HCI_CONFIGURE_PORT_CNF_NEG	OnHciConfigurePortConfirmNegative
HCI_INQUIRY_CNF	OnHciInquiryCnf
HCI_INQUIRY_EVT	OnHciInquiryEvt
HCI_LOCAL_ADDRESS_CNF	OnHciLocalAddressCnf
HCI_LOCAL_ADDRESS_CNF_NEG	OnHciLocalAddressCnfNeg
HCI_REMOTE_NAME_CNF	OnHciRemoteNameCnf
HCI_REMOTE_NAME_CNF_NEG	OnHciRemoteNameCnfNeg
HCI_START_CNF	OnHciStartCnf
HCI_WRITE_SCAN_ENABLE_CNF	OnHciWriteScanEnableCnf

HCI_WRITE_SCAN_ENABLE_CNF_NEG	OnHciWriteScanEnableCnfNeg
HCI_WRITE_AUTHENTICATION_MODE_CNF	OnHciWriteAuthenticationModeCnf
HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG	OnHciWriteAuthenticationModeCnfNeg
HCI_WRITE_ENCRYPTION_MODE_CNF	OnHciWriteEncryptionModeCnf
HCI_WRITE_ENCRYPTION_MODE_CNF_NEG	OnHciWriteEncryptionModeCnfNeg
HCI_WRITE_COD_CNF	OnHciWriteCodCnf
HCI_WRITE_COD_CNF_NEG	OnHciWriteCodCnfNeg
HCI_WRITE_NAME_CNF	OnHciWriteNameCnf
HCI_WRITE_NAME_CNF_NEG	OnHciWriteNameCnfNeg
HCI_WRITE_CONNECT_TIMEOUT_CNF	OnHciWriteConnectTimeoutCnf
HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG	OnHciWriteConnectTimeoutCnfNeg
HCI_WRITE_PAGE_TIMEOUT_CNF	OnHciWritePageTimeoutCnf
HCI_WRITE_PAGE_TIMEOUT_CNF_NEG	OnHciWritePageTimeoutCnfNeg
SIL_SET_DEVICE_CNF	OnSilSetDeviceCnf
SIL_SET_DEVICE_CNF_NEG	OnSilSetDeviceCnfNeg
SIL_REQ_DEVICE_CNF	OnSilReqDeviceCnf
SIL_REQ_DEVICE_CNF_NEG	OnSilReqDeviceCnfNeg
COM_CONNECT_CNF	OnComConnectCnf
COM_CONNECT_CNF_NEG	OnComConnectCnfNeg
COM_DATA_IND	OnComDataInd
COM_DATA_CNF	OnComDataCnf
COM_DATA_CNF_NEG	OnComDataCnfNeg
COM_DISCONNECT_EVT	OnComDisconnectEvt
COM_DISCONNECT_CNF	OnComDisconnectCnf
COM_DISCONNECT_CNF_NEG	OnComDisconnectCnfNeg

- ◆ Lines 164–182: This code has been explained in SDP Information CommandsDlg.cpp file
- ◆ Lines 183–192: TVINSERTSTRUCT is used to define a structure for a tree implementation. m_tree is a member variable to create nodes for a tree. To insert an item as a node of the tree the method InsertItem (&tvInsert) is used. The index variable is used to insert messages in the list box.
- ◆ Lines 193–204: This code has been explained in HCIInformationCommandsDlg.cpp
- ◆ Lines 205–237: This code has been explained in HCIInformationCommandsDlg.cpp. This Bluetooth event handling code is required to handle the Bluetooth events and is taken from the PC Reference stack code.
- ◆ Lines 238–288: This code is explained in HCIInformationCommandsDlg.cpp.
- ◆ Lines 289–297: When SIL_ReqDevice(0) command is called, the corresponding BLUETOOTH Event SIL_REQ_DEVICE_CNF is fired. The SIL_REQ_DEVICE_CNF event calls corresponding

message-handler function `OnSilReqDeviceCnf(void **ppMsg)`. This function gives a diagnostic message during the connection if the interface type has been changed.

- ◆ Lines 298–302: If the command `SIL_ReqDevice(0)` has failed, the corresponding event `SIL_REQ_DEVICE_CNF_NEG` will be fired. The `SIL_REQ_DEVICE_CNF_NEG` event calls corresponding message-handler function `OnSilReqDeviceCnfNeg(void **ppMsg)`. The message box will display that the device request failed.
- ◆ Lines 303–346: This code is explained in `HCIInformationCommandsDlg.cpp` file.
- ◆ Lines 347–351: `HCI_ReqWriteEncryptionMode(0, HCI_ENCRYPTION_OFF)` sets encryption mode to OFF. This command fires either the `HCI_WRITE_ENCRYPTION_MODE_CNF` or `HCI_WRITE_ENCRYPTION_MODE_CNF_NEG` event corresponding to success or failure.
- ◆ Lines 352–356: `HCI_ReqWriteAuthenticationMode(0, HCI_AUTH_DISABLE)` disables the authentication mode. This command fires either the `HCI_WRITE_AUTHENTICATION_MODE_CNF` or `HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG` event corresponding to success or failure.
- ◆ Lines 357–360: `HCI_WRITE_ENCRYPTION_MODE_CNF_NEG` event invokes this method.
- ◆ Lines 361–365: `HCI_ReqWriteConnectTimeout(0, 0x1FA0)` sets the timeout value OX1FA0 for HCI connection.
- ◆ Lines 366–369: `HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG` event invokes this method.
- ◆ Lines 370–374: `HCI_ReqWritePageTimeout(0, 8000)` sets page maximum time-out value to wait for the Response.
- ◆ Lines 375–378: `HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG` event invokes this method.
- ◆ Lines 379–387: `HCI_ReqWriteCod(0, _tCod)` writes COD of the local device.
- ◆ Lines 388–391: `HCI_WRITE_PAGE_TIMEOUT_CNF_NEG` event invokes this method.
- ◆ Lines 401–405: `HCI_ReqWriteName(0, (HCI_Tname*) "BT File")` changes the Bluetooth device name to BT File.
- ◆ Lines 406–409: `HCI_WRITE_COD_CNF_NEG` event invokes this method.
- ◆ Lines 410–414: Request to change scan settings to either page scan mode or inquiry scan mode.
- ◆ Lines 415–418: `HCI_WRITE_NAME_CNF_NEG` event invokes this method.
- ◆ Lines 419–423 : Request to register Security Manager.
- ◆ Lines 424–427: `HCI_WRITE_SCAN_ENABLE_CNF_NEG` event invokes this method.
- ◆ Lines 428–440: If the sequence number is zero, it again asks to register service with a different group. Otherwise, it invokes `OnInquiry()` method.
- ◆ Lines 441–446: If the registration of Security Manager fails, this method is invoked. A message box appears to show the failed command.
- ◆ Lines 447–516: This code is explained in `HCIInformationDlg.cpp`.
- ◆ Lines 517–525: When the link setup is successful, this method is invoked. `OnGetServices()` method is called to get services.
- ◆ Lines 526–531: If the link setup fails, the message box displays the message “No connection made”.
- ◆ Lines 532–545: This method requests a search procedure to find out a particular service.
- ◆ Lines 546–552: When `SD_CONNECT_CNF_NEG` event has been fired, the message box displays indicating that the device is unable to connect to service discovery.

- ◆ Lines 553 to 576: This method sends a request to get attributes of a particular service on a remote Bluetooth device.
- ◆ Lines 577–584: When the `SD_SERVICE_SEARCH_CNF_NEG` event is fired, the message box displays that the service search has failed.
- ◆ Lines 585–604: When the `SD_SERVICE_SEARCH_CNF` event is fired for the first time, this message handler function invokes `ReceiveServiceName(tServiceAttributeCnf)`. When `SD_SERVICE_SEARCH_CNF` event is fired for the second time, this message handler function invokes `SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle)`.
- ◆ Lines 605–612: If retrieval of service attribute fails, the message box displays the diagnostic message.
- ◆ Lines 613–618: When `SD_DISCONNECT_CNF` event is fired, the function `OnSelServices()` is invoked.
- ◆ Lines 619–671: This code is explained in `SDPInformationCommandsDlg.cpp` file.
- ◆ Lines 672–684: This code is explained in `HCIInformationCommandsDlg.cpp` file.
- ◆ Lines 685–694: After the pincode is received by the client, the response is sent to the server.
- ◆ Lines 695–701: After the Security Manager connect event is fired, the ACL handle and Bluetooth device address are returned and stored.
- ◆ Lines 702–707: When the Security Manager disconnect event is fired, `OnCloseApplication()` will be invoked.
- ◆ Lines 708–713: This code is explained in `HCIInformationCommandsdlg.cpp` file.
- ◆ Lines 714–730: This code retrieves the version of RFCOMM and displays in the dialog box.
- ◆ Lines 731–740: This method sends a request to retrieve the attributes of the particular service specified in the `BT_SERVICE_NAME`.
- ◆ Lines 741–765: The retrieved attributes of the requested service are stored and displayed in the dialog box.
- ◆ Lines 766–776: This method sends a request to retrieve the service record handle and protocol descriptor list.
- ◆ Lines 777–792: This method stores the requested service record handle to display on the user interface.
- ◆ Lines 793–796: The Security Manager's `SCM_SECURITY_HANDLER` is de-registered from Bluetooth module.
- ◆ Lines 797–825: The Security Manager's `SCM_MONITOR_GROUP` is de-registered from Bluetooth module. In the next step, if the DBM service has not yet been de-registered, it will de-register that service. Otherwise, the Security manager's disconnect request will be sent, provided the ACL Link is there.
- ◆ Lines 826–831: Corresponding to the negative event, the diagnostic message is displayed.
- ◆ Lines 832–843: After the de-registration of DBM service has been confirmed, the Security Manager sends a request to disconnect it.
- ◆ Lines 844–849: When the de-registration of DBM service has not been confirmed, the diagnostic message is displayed in the message box.
- ◆ Lines 850–855: The current dialog will be destroyed after receiving the `SCM_DISCONNECT_CNF` Event.
- ◆ Lines 856–861: After the `SCM_DISCONNECT_CNF_NEG` event is received, the diagnostic message is displayed.
- ◆ Lines 862–865: The destroy window method is defined to destroy the current dialog.

- ◆ Lines 866–878: The retrieved remote devices names are added as child nodes to the root node of the tree.
- ◆ Lines 879–887: The found service is added to `m_ServicesFound` structure.
- ◆ Lines 888–898: The available services on a particular remote BLUETOOTH device is displayed as their child nodes.
- ◆ Lines 899–902: The found devices are added to `m_DevicesFound` structure.
- ◆ Lines 903–909: This code is explained in `HCIInformationCommandsDlg.cpp` file.
- ◆ Lines 910–922: After a device is selected, the application requests to establish connection with Security Manager.
- ◆ Lines 924–930: After a service is selected, the application requests to register a retrieved service in local Database Manager.
- ◆ Lines 931–935: The application requests connection to discover services on remote devices.
- ◆ Lines 936–942: The local RFCOMM module requests the connection with the remote RFCOMM module.
- ◆ Lines 943–952: After RFCOMM connection is confirmed, the message box appears to Show that the issued command is a success.
- ◆ Lines 953–959: The message box displays that the establishment of RFCOMM connection has failed.
- ◆ Lines 960–978: The command to read the incoming data from the remote device is issued. The response is sent to the remote device to tell that the data has been received.
- ◆ Lines 979–982: The message is stored when the `COM_DATA_CNF` event has been received.
- ◆ Lines 983–987: The message box displays that the data cannot be sent on RFCOMM channel.
- ◆ Lines 988–994: After `COM_DISCONNECT_EVENT` is fired, the dialog is closed.
- ◆ Lines 995–1001: After the `COM_DISCONNECT_CNF` is fired, the dialog is closed.
- ◆ Lines 1002–1007: The message box shows that the RFCOMM cannot be disconnected.
- ◆ Lines 1008–1063: After the button `IDC_BUTTON1` is clicked, the corresponding message handler function `OnBrowse ()` is called. A temporary file is created in write mode. The File dialog has been created with filter `“*.*”`. The current directory is selected in which the file dialog has to be opened. The filename and the path of the selected file are obtained and the selected file is opened to read the content. The length of the selected file is also obtained. The length and name of the selected file are written into a temporary file by reading from the selected file and writing into the temporary file. The selected file and temporary files are closed. The memory is allocated to store the content of temporary file and the read data is copied into the memory variable. `COM_DataSend` is a function to send the data to the remote BLUETOOTH device. The status of the file transmission displays in the list box. The temporary file is deleted.

Code Output

When the preceding application is built in the VC++ environment and executed, the window in Figure 9-7 appears.

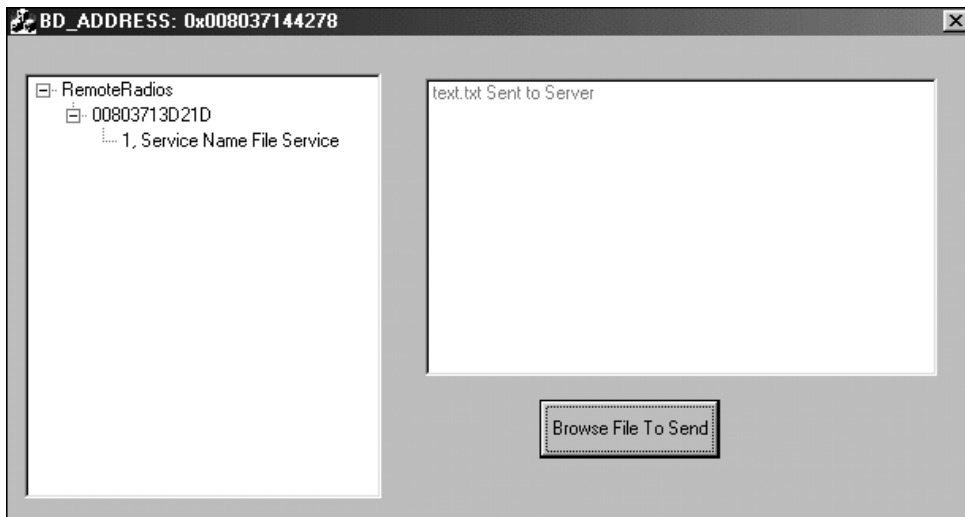


Figure 9-7: Output of the Client Module

The left side of the dialog in Figure 9-7 displays the Bluetooth device address of the server and available services on the server in the form of a tree structure. The right side of the above dialog contains one button labeled Browse File To Send. When this button is clicked, the file dialog appears to enable the user to select and send a file. The status file dispatch appears in the list box designed above the button.

Server Module

Listings 9-19 and 9-20 give the source code of `RadioFileServerDlg.h` and `RadioFileServerDlg.cpp`, respectively.

Listing 9-19: `RadioFileServerDlg.h`

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. // RadioFileServerDlg.h : header file
2.
3. #include "Events.h"
4. #include "ConnectionInfo.h"
5. #include "Remotedevice.h"
6. #include "RS232.h"
7. #include "service.h"
8. #include <exp\sd.h>
9. #include <exp\BT_COMServer.h>
10. #include <afxtempl.h>
11. CRadioFileServerDlg dialog
12. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
13. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
14.     { WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
15.     (AFX_PMSG) (AFX_PMSGW) (void (AFX_MSG_CALL CWnd::*) (void **)) &memberFxn },
16. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
17.     SendMessage( (HWND) this->
m_hWnd, WM_BLUETOOTH_EVENT, (WPARAM) uiBtEventID, (LPARAM) &pMsg)
18.
19. class CRadioFileServerDlg : public CDialog
20. {
21.

```

```

22. public:
23.   CRadioFileServerDlg(CWnd* pParent = NULL);
24.   ~CRadioFileServerDlg();
25.       CConnectionInfo m_ConnectionInfo;
26.
27. private:
28.   void InitSecurityClient();
29.   BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
30.
31.   //{AFX_DATA(CRadioFileServerDlg)
32.   enum { IDD = IDD_RADIOCHAT_DIALOG };
33.   CListBox      m_ChatArea;
34.   CEdit m_InputChat;
35.   CTreeCtrl      m_tree;
36.   //}}AFX_DATA
37.
38.   // ClassWizard generated virtual function overrides
39.   //{AFX_VIRTUAL(CRadioFileServerDlg)
40.
41. public:
42.   virtual BOOL DestroyWindow();
43. protected:
44.   virtual void DoDataExchange(CDataExchange* pDX);
45.       virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM
lParam);
46.   //}}AFX_VIRTUAL
47.
48. public:
49.   void AddDevice(CString sAddress, CString sName);
50.   void AddDevice(CDevice device);
51.   void ShowAllDevicesFound();
52.   void AddService(CString sService);
53.   void AddService(CService service);
54.   void ShowAllServicesFound();
55.   void AskForServiceName();
56.   void ReceiveServiceName(SD_TServiceAttributeCnf *tServiceAttributeCnf);
57.   void AskForServiceRecordHandle();
58.   void ReceiveServiceRecordHandle(SD_TServiceAttributeCnf
*tServiceAttributeCnf);
59. protected:
60.   HICON m_hIcon;
61.   int index;
62.   CServerEvents *m_pServerEvents;
63.   CRS232 *m_pSerialPort;
64.   CArray <CDevice,CDevice> m_DevicesFound;
65.   CArray <CService,CService> m_ServicesFound;
66.   int m_RemoteNameCounter;
67.   int m_ServiceCounter;
68.   uint8 m_RFServerChannel;
69.   uint16 m_RFCommHandle;
70.   // Generated message map functions
71.   //{AFX_MSG(CRadioFileServerDlg)
72.   virtual BOOL OnInitDialog();
73.   afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
74.   afx_msg void OnPaint();
75.   afx_msg HCURSOR OnQueryDragIcon();

```

```

76.  afx_msg void OnInquiry();
77.  afx_msg void OnSelDevice();
78.  afx_msg void OnConnect();
79.  afx_msg void OnGetServices();
80.  afx_msg void OnSelServices();
81.  afx_msg void OnDestroy();
82.  afx_msg void OnSerialport();
83.  afx_msg void OnCloseapplication();
84.  afx_msg void OnHCISerial();
85.  afx_msg void OnHCIUsb();
86.  afx_msg void OnComStartCnf(void **ppMsg);
87.  afx_msg void OnComStartCnfNeg(void **ppMsg);
88.  afx_msg void OnComFillPdlCnf(void **ppMsg);
89.  afx_msg void OnComFillPdlCnfNeg(void **ppMsg);
90.  afx_msg void OnComRegisterCnf(void **ppMsg);
91.  afx_msg void OnComRegisterCnfNeg(void **ppMsg);
92.  afx_msg void OnComConnectInd(void **ppMsg);
93.  afx_msg void OnComDataInd(void **ppMsg);
94.  afx_msg void OnComDataCnf(void **ppMsg);
95.  afx_msg void OnComDataCnfNeg(void **ppMsg);
96.
97.  afx_msg void OnComVersionCnf(void **ppMsg);
98.  afx_msg void OnScmRegisterCnf(void **ppMsg);
99.  afx_msg void OnScmRegisterCnfNeg(void **ppMsg);
100. afx_msg void OnConnectAcceptInd(void **ppMsg);
101. afx_msg void OnScmPincodeInd(void **ppMsg);
102. afx_msg void OnScmConnectCnf(void **ppMsg);
103. afx_msg void OnScmConnectCnfNeg(void **ppMsg);
104. afx_msg void OnScmConnectEvt(void **ppMsg);
105. afx_msg void OnScmDisconnectEvt(void **ppMsg);
106. afx_msg void OnScmDisconnectCnf(void **ppMsg);
107. afx_msg void OnScmDisconnectCnfNeg(void **ppMsg);
108. afx_msg void OnScmDeRegisterCnf(void **ppMsg);
109. afx_msg void OnScmDeRegisterCnfNeg(void **ppMsg);
110. afx_msg void OnSdStartCnf(void **ppMsg);
111. afx_msg void OnSdConnectCnf(void **ppMsg);
112. afx_msg void OnSdConnectCnfNeg(void **ppMsg);
113. afx_msg void OnSdServiceSearchCnf(void **ppMsg);
114. afx_msg void OnSdServiceSearchCnfNeg(void **ppMsg);
115. afx_msg void OnSdServiceAttributeCnf(void **ppMsg);
116. afx_msg void OnSdServiceAttributeCnfNeg(void **ppMsg);
117. afx_msg void OnSdDisconnectCnf(void **ppMsg);
118. afx_msg void OnDbmRegisterServiceCnf(void **ppMsg);
119. afx_msg void OnDbmRegisterServiceCnfNeg(void **ppMsg);
120. afx_msg void OnDbmUnRegisterServiceCnf(void **ppMsg);
121. afx_msg void OnDbmUnRegisterServiceCnfNeg(void **ppMsg);
122. afx_msg void OnDbmAddDescriptorCnf(void **ppMsg);
123. afx_msg void OnDbmAddDescriptorCnfNeg(void **ppMsg);
124. afx_msg void OnHciConfigurePortConfirm(void **ppMsg);
125. afx_msg void OnHciConfigurePortConfirmNegative(void **ppMsg);
126. afx_msg void OnHciInquiryCnf(void **ppMsg);
127. afx_msg void OnHciInquiryEvt(void **ppMsg);
128. afx_msg void OnHciLocalAddressCnf(void **ppMsg);
129. afx_msg void OnHciLocalAddressCnfNeg(void **ppMsg);
130. afx_msg void OnHciRemoteNameCnf(void **ppMsg);
131. afx_msg void OnHciRemoteNameCnfNeg(void **ppMsg);

```



```

132.  afx_msg void OnHciStartCnf(void **ppMsg);
133.  afx_msg void OnHciWriteScanEnableCnf(void **ppMsg);
134.  afx_msg void OnHciWriteScanEnableCnfNeg(void **ppMsg);
135.
136.
137.  afx_msg void OnHciWriteAuthenticationModeCnf(void **ppMsg);
138.  afx_msg void OnHciWriteAuthenticationModeCnfNeg(void **ppMsg);
139.  afx_msg void OnHciWriteEncryptionModeCnf(void **ppMsg);
140.  afx_msg void OnHciWriteEncryptionModeCnfNeg(void **ppMsg);
141.  afx_msg void OnHciWriteCodCnf(void **ppMsg);
142.  afx_msg void OnHciWriteCodCnfNeg(void **ppMsg);
143.  afx_msg void OnHciWriteNameCnf(void **ppMsg);
144.  afx_msg void OnHciWriteNameCnfNeg(void **ppMsg);
145.  afx_msg void OnHciWriteConnectTimeoutCnf(void **ppMsg);
146.  afx_msg void OnHciWriteConnectTimeoutCnfNeg(void **ppMsg);
147.  afx_msg void OnHciWritePageTimeoutCnf(void **ppMsg);
148.  afx_msg void OnHciWritePageTimeoutCnfNeg(void **ppMsg);
149.  afx_msg void OnSilSetDeviceCnf(void **ppMsg);
150.  afx_msg void OnSilSetDeviceCnfNeg(void **ppMsg);
151.  afx_msg void OnSilReqDeviceCnf(void **ppMsg);
152.  afx_msg void OnSilReqDeviceCnfNeg(void **ppMsg);
153.  afx_msg void OnComConnectCnf(void **ppMsg);
154.  afx_msg void OnComConnectCnfNeg(void **ppMsg);
155.  afx_msg void OnSdsStartCnf(void **ppMsg);
156.  afx_msg void OnButton2();
157.  //}}AFX_MSG
158.  DECLARE_MESSAGE_MAP()
159. };
160.

```

Code Description

Listing 9-19 is the header file in which the necessary variables are declared. These variables are used in the `RadioFileServerDlg.cpp`. The code explanation given for `RadioFileServerDlg.cpp` will clarify the use of the various variables.

Listing 9-20: RadioFileServerDlg.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

// RadioFileServerDlg.cpp
1. #include "stdafx.h"
2. #include "RadioChat.h"
3. #include "RadioFileServerDlg.h"
4. #include "Events.h"
5. #include <process.h>
6. #include "windows.h"
7. #include <exp/msg.h>
8. #include <exp/hci.h>
9. #include <exp/hci_drv.h>
10. #include <exp/scm.h>
11. #include <exp/com.h>
12. #include <exp/dbm.h>
13. #include <exp/sd.h>
14. #include <exp/sds.h>
15. #include <exp/vos2com.h>

```

```

16. #include <exp/sil.h>
17. #include <exp/Bstr.h>
18.
19. #ifdef _DEBUG
20. #define new DEBUG_NEW
21. #undef THIS_FILE
22. static char THIS_FILE[] = __FILE__;
23. #endif
24.
25. HTREEITEM hPA,hdevice1;
26. union MessageMapFunctions
27. {
28.   AFX_PMSG pfn;
29.   void      (AFX_MSG_CALL CWnd::*pfn_btfn)(void **);
30. };
31. #define PINCODE_LENGTH ((SCM_TPincodeLength) 4)
32. static const SCM_TPincode _tPincode =
    {'1','2','3','4','0','0','0','0','0','0','0','0','0','0','0','0','0','0'};
33. #define PORTSETTINGS (uint8 *)("COM1:Baud=57600 parity=N data=8 stop=1")
34. #define InterSelSerial((uint8) 0)
35. #define InterSelUSB      ((uint8) 1)
36. #define SRP_SERIAL_GENERIC_SERIALPORT_UUID          ((uint16) 0x1101)
37. static const HCI_TCod _tCod={0x20,0x04,0x04};
38. class CAboutDlg : public CDialog
39. {
40. public:
41.   CAboutDlg();
42.   //{AFX_DATA(CAboutDlg)
43.   enum { IDD = IDD_ABOUTBOX };
44.   //}AFX_DATA
45.   //{AFX_VIRTUAL(CAboutDlg)
46. protected:
47. virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
         support
48. //}AFX_VIRTUAL
49. protected:
50.   //{AFX_MSG(CAboutDlg)
51.   //}AFX_MSG
52. DECLARE_MESSAGE_MAP()
53. };
54. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
55. {
56.   //{AFX_DATA_INIT(CAboutDlg)
57.   //}AFX_DATA_INIT
58. }
59. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
60. {
61.   CDialog::DoDataExchange(pDX);
62.   //{AFX_DATA_MAP(CAboutDlg)
63.   //}AFX_DATA_MAP
64. }
65. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
66.   //{AFX_MSG_MAP(CAboutDlg)
67.   // No message handlers
68.   //}AFX_MSG_MAP
69. END_MESSAGE_MAP()

```

```

70. CRadioFileServerDlg::CRadioFileServerDlg(CWnd* pParent /*=NULL*/)
71. : CDialog(CRadioFileServerDlg::IDD, pParent)
72. {
73.     //{AFX_DATA_INIT(CRadioFileServerDlg)
74.     // NOTE: the ClassWizard will add member initialization here
75.     //}AFX_DATA_INIT
76.     m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
77.     m_pServerEvents = new CServerEvents();
78. }
79. CRadioFileServerDlg::~CRadioFileServerDlg()
80. {
81.     m_DevicesFound.RemoveAll();
82.     m_ServicesFound.RemoveAll();
83.     delete m_pServerEvents;
84. }
85. void CRadioFileServerDlg::DoDataExchange(CDataExchange* pDX)
86. {
87.     CDialog::DoDataExchange(pDX);
88.     //{AFX_DATA_MAP(CRadioFileServerDlg)
89.     DDX_Control(pDX, IDC_EDIT1, m_InputChat);
90.     DDX_Control(pDX, IDC_LIST1, m_ChatArea);
91.     DDX_Control(pDX, IDC_TREE1, m_tree);
92.     //}AFX_DATA_MAP
93. }
94. BEGIN_MESSAGE_MAP(CRadioFileServerDlg, CDialog)
95.     //{AFX_MSG_MAP(CRadioFileServerDlg)
96.     ON_WM_SYSCOMMAND()
97.     ON_WM_PAINT()
98.     ON_WM_QUERYDRAGICON()
99.     ON_BLUETOOTH_EVENT(COM_DATA_IND, OnComDataInd )
100.    ON_BLUETOOTH_EVENT(COM_REGISTER_CNF, OnComRegisterCnf )
101.    ON_BLUETOOTH_EVENT(COM_REGISTER_CNF_NEG, OnComRegisterCnfNeg )
102.    ON_BLUETOOTH_EVENT(COM_FILL_PDL_CNF, OnComFillPdlCnf )
103.    ON_BLUETOOTH_EVENT(COM_FILL_PDL_CNF_NEG, OnComFillPdlCnfNeg )
104.    ON_BLUETOOTH_EVENT(COM_CONNECT_IND, OnComConnectInd )
105.    ON_BLUETOOTH_EVENT(COM_START_CNF, OnComStartCnf)
106.    ON_BLUETOOTH_EVENT(COM_START_CNF_NEG, OnComStartCnfNeg)
107.    ON_BLUETOOTH_EVENT(COM_VERSION_CNF, OnComVersionCnf)
108.    ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF, OnScmRegisterCnf)
109.    ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF_NEG, OnScmRegisterCnfNeg)
110.    ON_BLUETOOTH_EVENT(SCM_CONNECT_ACCEPT_IND, OnConnectAcceptInd)
111.    ON_BLUETOOTH_EVENT(SCM_PINCODE_IND, OnScmPincodeInd)
112.    ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF, OnScmConnectCnf)
113.    ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF_NEG, OnScmConnectCnfNeg)
114.    ON_BLUETOOTH_EVENT(SCM_CONNECT_EVT, OnScmConnectEvt)
115.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_EVT, OnScmDisconnectEvt)
116.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF, OnScmDisconnectCnf)
117.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF_NEG, OnScmDisconnectCnfNeg)
118.    ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF, OnScmDeRegisterCnf)
119.    ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF_NEG, OnScmDeRegisterCnfNeg)
120.    ON_BLUETOOTH_EVENT(SD_START_CNF, OnSdStartCnf)
121.    ON_BLUETOOTH_EVENT(SD_CONNECT_CNF, OnSdConnectCnf)
122.    ON_BLUETOOTH_EVENT(SD_CONNECT_CNF_NEG, OnSdConnectCnfNeg)
123.    ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF, OnSdServiceSearchCnf)
124.    ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF_NEG, OnSdServiceSearchCnfNeg)
125.    ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF, OnSdServiceAttributeCnf)

```

```

126.     ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF_NEG,
        OnSdServiceAttributeCnfNeg)
127.     ON_BLUETOOTH_EVENT(SD_DISCONNECT_CNF, OnSdDisconnectCnf)
128.     ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF, OnDbmRegisterServiceCnf)
129.     ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF_NEG,
        OnDbmRegisterServiceCnfNeg)
130.     ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF, OnDbmUnRegisterServiceCnf)
131.     ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF_NEG,
        OnDbmUnRegisterServiceCnfNeg)
132.     ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF, OnDbmAddDescriptorCnf)
133.     ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF_NEG, OnDbmAddDescriptorCnfNeg)
134.     ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF, OnHciConfigurePortConfirm)
135.     ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF_NEG,
        OnHciConfigurePortConfirmNegative)
136.     ON_BLUETOOTH_EVENT(HCI_INQUIRY_CNF, OnHciInquiryCnf)
137.     ON_BLUETOOTH_EVENT(HCI_INQUIRY_EVT, OnHciInquiryEvt)
138.     ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF, OnHciLocalAddressCnf)
139.     ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF_NEG, OnHciLocalAddressCnfNeg)
140.     ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF, OnHciRemoteNameCnf)
141.     ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF_NEG, OnHciRemoteNameCnfNeg)
142.     ON_BLUETOOTH_EVENT(HCI_START_CNF, OnHciStartCnf)
143.     ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF, OnHciWriteScanEnableCnf)
144.     ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF_NEG,
        OnHciWriteScanEnableCnfNeg)
145.
146.
147.     ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF,
        OnHciWriteAuthenticationModeCnf)
148.     ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG,
        OnHciWriteAuthenticationModeCnfNeg)
149.     ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF,
        OnHciWriteEncryptionModeCnf)
150.     ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF_NEG,
        OnHciWriteEncryptionModeCnfNeg)
151.     ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF, OnHciWriteCodCnf)
152.     ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF_NEG, OnHciWriteCodCnfNeg)
153.     ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF, OnHciWriteNameCnf)
154.     ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF_NEG, OnHciWriteNameCnfNeg)
155.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF,
        OnHciWriteConnectTimeoutCnf)
156.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG,
        OnHciWriteConnectTimeoutCnfNeg)
157.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF, OnHciWritePageTimeoutCnf)
158.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF_NEG,
        OnHciWritePageTimeoutCnfNeg)
159.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF, OnSilSetDeviceCnf)
160.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF_NEG, OnSilSetDeviceCnfNeg)
161.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF, OnSilReqDeviceCnf)
162.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF_NEG, OnSilReqDeviceCnfNeg)
163.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF, OnComConnectCnf )
164.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF_NEG, OnComConnectCnfNeg )
165.     ON_BLUETOOTH_EVENT(SDS_START_CNF, OnSdsStartCnf)
166.     ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
167.     //{AFX_MSG_MAP
168.     END_MESSAGE_MAP()
169.     BOOL CRadioFileServerDlg::OnInitDialog()

```

```

170. {
171.     CDialog::OnInitDialog();
172.     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
173.     ASSERT(IDM_ABOUTBOX < 0xF000);
174.     CMenu* pSysMenu = GetSystemMenu(FALSE);
175.     if (pSysMenu != NULL)
176.     {
177.         CString strAboutMenu;
178.         strAboutMenu.LoadString(IDS_ABOUTBOX);
179.         if (!strAboutMenu.IsEmpty())
180.         {
181.             pSysMenu->AppendMenu(MF_SEPARATOR);
182.             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                strAboutMenu);
183.         }
184.     }
185.     SetIcon(m_hIcon, TRUE);
186.     SetIcon(m_hIcon, FALSE);
187.     m_pServerEvents->m_pParentDialog = this;
188.     TVINSERTSTRUCT tvInsert;
189.     tvInsert.hParent = NULL;
190.     tvInsert.hInsertAfter = NULL;
191.     tvInsert.item.mask = TVIF_TEXT;
192.     tvInsert.item.pszText = _T("RemoteRadios");
193.     hPA = m_tree.InsertItem(&tvInsert);
194.     index=0;
195.     m_InputChat.SetWindowText("Type And Press Enter To Send Your Message");
196.     InitSecurityClient();
197.     return TRUE;
198. }
199.
200. LRESULT CRadioFileServerDlg::WindowProc(UINT message, WPARAM wParam,
    LPARAM lParam)
201. {
202.     MSG_TMsg **ptMsg;
203.     if (message == WM_BLUETOOTH_EVENT)
204.     {
205.         OnBluetoothEvent( message, wParam, lParam);
206.         ptMsg = (MSG_TMsg**)lParam;
207.         if (*ptMsg != NULL)
208.             VOS_Free((void **)lParam);
209.     }
210.     return CDialog::WindowProc(message, wParam, lParam);
211. }
212. BOOL CRadioFileServerDlg::OnBluetoothEvent(UINT message, WPARAM wParam,
    LPARAM lParam)
213. {
214.     const AFX_MSGMAP* pMessageMap;
215.     const AFX_MSGMAP_ENTRY* lpEntry;
216.     #ifdef _AFXDLL
217.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
218.         pMessageMap = (*pMessageMap->pfnGetBaseMap)())
219.     #else
220.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
221.         pMessageMap = pMessageMap->pBaseMap)
222.     #endif

```

```

223. {
224. #ifdef _AFXDLL
225.     ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
226. #else
227.     ASSERT(pMessageMap != pMessageMap->pBaseMap);
228. #endif
229.     lpEntry = (AFX_MSGMAP_ENTRY*)(&pMessageMap->lpEntries[0]);
230.     while (lpEntry->nSig != AfxSig_end)
231.     {
232.         if ((lpEntry->nMessage == message) && (lpEntry->nCode ==
            wParam))
233.         {
234.             union MessageMapFunctions mmf;
235.             mmf.pfn = lpEntry->pfn;
236.             (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
237.             return TRUE;
238.         }
239.         lpEntry++;
240.     }
241.     return FALSE;
242. }
243. return FALSE;
244. }
245. void CRadioFileServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
246. {
247.     if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
248.     {
249.         CAboutDlg dlgAbout;
250.         dlgAbout.DoModal();
251.     }
252.     else
253.     {
254.         CDialog::OnSysCommand(nID, lParam);
255.     }
256. }
257. void CRadioFileServerDlg::OnPaint()
258. {
259.     if (IsIconic())
260.     {
261.         CPaintDC dc(this);
262.         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
263.         int cxIcon = GetSystemMetrics(SM_CXICON);
264.         int cyIcon = GetSystemMetrics(SM_CYICON);
265.         CRect rect;
266.         GetClientRect(&rect);
267.         int x = (rect.Width() - cxIcon + 1) / 2;
268.         int y = (rect.Height() - cyIcon + 1) / 2;
269.         dc.DrawIcon(x, y, m_hIcon);
270.     }
271.     else
272.     {
273.         CDialog::OnPaint();
274.     }
275. }
276. HCURSOR CRadioFileServerDlg::OnQueryDragIcon()
277. {

```

```

278. return (HCURSOR) m_hIcon;
279. }
280. void CRadioFileServerDlg::InitSecurityClient()
281. {
282.     SIL_SetDevice(0,SIL_SERIAL);
283. }
284. void CRadioFileServerDlg::OnSilSetDeviceCnf(void **ppMsg)
285. {
286.     ppMsg = ppMsg;
287.     HCI_ReqConfigurePort(0,PORTSETTINGS);
288. }
289. void CRadioFileServerDlg::OnSilSetDeviceCnfNeg(void **ppMsg)
290. {
291.     SIL_TSetDevice* ptSetDevice;
292.     ptSetDevice = (SIL_TSetDevice*)*ppMsg;
293.     if(ptSetDevice->tHdr.iResult == SIL_ERR_DEVICE)
294.         SIL_ReqDevice(0);
295. }
296. void CRadioFileServerDlg::OnSilReqDeviceCnf(void **ppMsg)
297. {
298.     SIL_TReqDevice* ptReq;
299.     ptReq = (SIL_TReqDevice*) *ppMsg;
300.     if(ptReq->uiDevice == SIL_SERIAL)
301.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
Interface is SERIAL"));
302.     if(ptReq->uiDevice == SIL_USB)
303.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
Interface is USB"));
304. }
305. void CRadioFileServerDlg::OnSilReqDeviceCnfNeg(void **ppMsg)
306. {
307.     ppMsg = ppMsg;
308.     MessageBox(_T("Device Request FAILED!"));
309. }
310. void CRadioFileServerDlg::OnHciConfigurePortConfirm(void **ppMsg)
311. {
312.     HCI_TConfigurePortCnf *tConfigurePort = (HCI_TConfigurePortCnf *)*ppMsg;
313.     tConfigurePort = tConfigurePort;
314.     COM_ReqStart(0);
315. }
316. void CRadioFileServerDlg::OnHciConfigurePortConfirmNegative(void **ppMsg)
317. {
318.     HCI_TConfigurePortCnfNeg *tConfigurePort = (HCI_TConfigurePortCnfNeg
*)*ppMsg;
319.     tConfigurePort = tConfigurePort;
320.     MessageBox(_T("Could not open port"));
321. }
322. void CRadioFileServerDlg::OnComStartCnf(void **ppMsg)
323. {
324.     COM_TStartCnf *tStartCnf = (COM_TStartCnf *)*ppMsg;
325.     tStartCnf = tStartCnf;
326.     HCI_ReqLocalAddress(0);
327. }
328. void CRadioFileServerDlg::OnComStartCnfNeg(void **ppMsg)
329. {
330.     COM_TStartCnfNeg *tStartCnfNeg = (COM_TStartCnfNeg *)*ppMsg;

```

```

331.     tStartCnfNeg = tStartCnfNeg;
332.     MessageBox(_T("Could not start RFCOMM"));
333. }
334. void CRadioFileServerDlg::OnHciLocalAddressCnf(void **ppMsg)
335. {
336.     HCI_TLocalAddressCnf *tLocalAddress = (HCI_TLocalAddressCnf
337.         *)*ppMsg;
338.     char lpStr[59];
339.     wsprintf(&lpStr[0], "BD_ADDRESS: 0x%02X%02X%02X%02X%02X%02X\0",
340.         tLocalAddress->tAddress.ucByte0,
341.         tLocalAddress->tAddress.ucByte1,
342.         tLocalAddress->tAddress.ucByte2,
343.         tLocalAddress->tAddress.ucByte3,
344.         tLocalAddress->tAddress.ucByte4,
345.         tLocalAddress->tAddress.ucByte5);
346.     SetWindowText(_T(lpStr));
347.     SD_ReqStart(0);
348. }
349. void CRadioFileServerDlg::OnHciLocalAddressCnfNeg(void **ppMsg)
350. {
351.     ppMsg = ppMsg;
352.     SetWindowText(_T("DEVICE NOT FOUND"));
353.     SD_ReqStart(0);
354. }
355. void CRadioFileServerDlg::OnSdStartCnf(void **ppMsg)
356. {
357.     ppMsg = ppMsg;
358.     HCI_ReqWriteEncryptionMode(0, HCI_ENCRYPTION_OFF);
359. }
360. void CRadioFileServerDlg::OnHciWriteEncryptionModeCnf(void **ppMsg)
361. {
362.     ppMsg = ppMsg;
363.     HCI_ReqWriteAuthenticationMode(0, HCI_AUTH_DISABLE);
364. }
365. void CRadioFileServerDlg::OnHciWriteEncryptionModeCnfNeg(void **ppMsg)
366. {
367.     ppMsg = ppMsg;
368. }
369. void CRadioFileServerDlg::OnHciWriteAuthenticationModeCnf(void **ppMsg)
370. {
371.     ppMsg = ppMsg;
372.     HCI_ReqWriteConnectTimeout(0, 0x1FA0);
373. }
374. void CRadioFileServerDlg::OnHciWriteAuthenticationModeCnfNeg(void **ppMsg)
375. {
376.     ppMsg = ppMsg;
377. }
378. void CRadioFileServerDlg::OnHciWriteConnectTimeoutCnf(void **ppMsg)
379. {
380.     ppMsg = ppMsg;
381.     HCI_ReqWritePageTimeout(0, 8000);
382. }
383. void CRadioFileServerDlg::OnHciWriteConnectTimeoutCnfNeg(void **ppMsg)
384. {
385.     ppMsg = ppMsg;
386. }

```



```

386. void CRadioFileServerDlg::OnHciWritePageTimeoutCnf(void **ppMsg)
387. {
388.     ppMsg = ppMsg;
389.
390.
391.     HCI_ReqWriteCod(0,_tCod);
392.
393.
394.
395. }
396. void CRadioFileServerDlg::OnHciWritePageTimeoutCnfNeg(void **ppMsg)
397. {
398.     ppMsg = ppMsg;
399. }
400.
401.
402.
403.
404.
405.
406.
407.
408.
409. void CRadioFileServerDlg::OnHciWriteCodCnf(void **ppMsg)
410. {
411.     ppMsg = ppMsg;
412.     HCI_ReqWriteName (0,(HCI_TName*) "BT Chat");
413. }
414. void CRadioFileServerDlg::OnHciWriteCodCnfNeg(void **ppMsg)
415. {
416.     ppMsg = ppMsg;
417. }
418. void CRadioFileServerDlg::OnHciWriteNameCnf(void **ppMsg)
419. {
420.     ppMsg = ppMsg;
421.     HCI_ReqWriteScanEnable(0,HCI_PAGE_SCAN_ENABLED |
        HCI_INQUIRY_SCAN_ENABLED);
422. }
423. void CRadioFileServerDlg::OnHciWriteNameCnfNeg(void **ppMsg)
424. {
425.     ppMsg = ppMsg;
426. }
427. void CRadioFileServerDlg::OnHciWriteScanEnableCnf(void **ppMsg)
428. {
429.     ppMsg = ppMsg;
430.     SCM_ReqRegister(0,SCM_SECURITY_HANDLER);
431. }
432. void CRadioFileServerDlg::OnHciWriteScanEnableCnfNeg(void **ppMsg)
433. {
434.     ppMsg = ppMsg;
435. }
436. void CRadioFileServerDlg::OnHciInquiryCnf(void **ppMsg)
437. {
438.     HCI_TInquiryCnf      *ptInquiryCnf;
439.     int count;
440.     CDevice device;

```

```

441.   ptInquiryCnf =(HCI_TInquiryCnf *) *ppMsg;
442.   count = m_DevicesFound.GetSize();
443.   m_RemoteNameCounter = 0;
444.   if (count > 0)
445.   {
446.       device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
447.       HCI_ReqRemoteName(10,
448.           device.tAddress,
449.           device.tPageScanPeriodMode,
450.           device.tPageScanMode,
451.           device.tClockOffset );
452.   }
453.   else
454.   {
455.       AfxMessageBox("No device found");
456.   }
457. }
458. void CRadioFileServerDlg::OnHciRemoteNameCnf(void **ppMsg)
459. {
460.     HCI_TRemoteNameCnf      *ptRemoteNameCnf;
461.     CDevice device;
462.     char sName[248];
463.     int count;
464.     ptRemoteNameCnf =(HCI_TRemoteNameCnf *) *ppMsg;
465.     sprintf(sName,"%s",&ptRemoteNameCnf->tName);
466.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)sName);
467.     m_RemoteNameCounter++;
468.     count = m_DevicesFound.GetSize();
469.     if (count > m_RemoteNameCounter)
470.     {
471.         device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
472.         HCI_ReqRemoteName(10,
473.             device.tAddress,
474.             device.tPageScanPeriodMode,
475.             device.tPageScanMode,
476.             device.tClockOffset );
477.     }
478.     else
479.     {
480.         ShowAllDevicesFound();
481.     }
482. }
483. void CRadioFileServerDlg::OnHciRemoteNameCnfNeg(void **ppMsg)
484. {
485.     HCI_TRemoteNameCnfNeg    *ptRemoteNameCnfNeg;
486.     CDevice device;
487.     int count;
488.     ptRemoteNameCnfNeg =(HCI_TRemoteNameCnfNeg *) *ppMsg;
489.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)_T("UNKNOWN"));
490.     m_RemoteNameCounter++;
491.     count = m_DevicesFound.GetSize();
492.     if (count > m_RemoteNameCounter)
493.     {
494.         device = (CDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
495.         HCI_ReqRemoteName(10,
496.             device.tAddress,

```

```

497.         device.tPageScanPeriodMode,
498.         device.tPageScanMode,
499.         device.tClockOffset );
500.     }
501.     else
502.     {
503.         ShowAllDevicesFound();
504.     }
505. }
506. void CRadioFileServerDlg::OnScmConnectCnf(void **ppMsg)
507. {
508.     SCM_TConnectCnf *tConnectCnf = (SCM_TConnectCnf *)*ppMsg;
509.     AfxMessageBox("connected");
510.     tConnectCnf = tConnectCnf;
511.     m_ConnectionInfo.tAclHandle = tConnectCnf->tHandle;
512.     m_ConnectionInfo.tAddress = tConnectCnf->tAddress;
513.     OnGetServices();
514. }
515. void CRadioFileServerDlg::OnScmConnectCnfNeg(void **ppMsg)
516. {
517.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
518.
519.     tConnectCnfNeg = tConnectCnfNeg;
520.     AfxMessageBox("No Connection made");
521. }
522. void CRadioFileServerDlg::OnSdConnectCnf(void **ppMsg)
523. {
524.     SD_TConnectCnf *tConnectCnf = (SD_TConnectCnf *)*ppMsg;
525.     SD_TUuid          *ptSearchPatternList;
526.     uint16             uiMaxRecords;
527.     uint8              ucNrOfUuids;
528.     m_ConnectionInfo.uiSdcHandle = tConnectCnf->uiSdcHandle;
529.     uiMaxRecords = 6;
530.     ucNrOfUuids = 1;
531.     ptSearchPatternList = (SD_TUuid*)VOS_Alloc((uint16)(ucNrOfUuids *
        sizeof(SD_TUuid)));
532.     ptSearchPatternList[0].eUuidType = SD_DET_UUID16;
533.     ptSearchPatternList[0].TUuid.uiUuid16 =
        SRP_SERIAL_GENERIC_SERIALPORT_UUID ; //SRP_HEADSET_UUID;
534.     SD_ReqServiceSearch (0, m_ConnectionInfo.uiSdcHandle, uiMaxRecords,
        ucNrOfUuids, ptSearchPatternList);
535. }
536. void CRadioFileServerDlg::OnSdConnectCnfNeg(void **ppMsg)
537. {
538.     SD_TConnectCnfNeg *tConnectCnfNeg = (SD_TConnectCnfNeg *)*ppMsg;
539.     CString str;
540.     str.Format("Could not connect to SD , Error %d",tConnectCnfNeg-
        >tHdr.iResult);
541.     MessageBox(str);
542. }
543. void CRadioFileServerDlg::OnSdServiceSearchCnf(void **ppMsg)
544. {
545.     SD_TServiceSearchCnf *tServiceSearchCnf = (SD_TServiceSearchCnf
        *)*ppMsg;
546.     uint16             uiCurrentServiceRecordCount;
547.     uint32             *pulSRHandles;

```

```

548.     uint16             *puiAttributeIDList;
549.     uint8              ucNrOfAttr;
550.     CService    service;
551.     uiCurrentServiceRecordCount = tServiceSearchCnf-
        >uiCurrentServiceRecordCount;
552.     pulSRHandles = (uint32*)
        VOS_Alloc(((uint16) (uiCurrentServiceRecordCount*sizeof(uint32))));
553.     (void*)memcpy(pulSRHandles,
554.                  &tServiceSearchCnf->ulServiceRecordHandleList,
555.                  (uiCurrentServiceRecordCount*sizeof(uint32)));
556.     m_ConnectionInfo.ulServiceRecordHandle = tServiceSearchCnf-
        >ulServiceRecordHandleList;
557.     ucNrOfAttr = 1;
558.     puiAttributeIDList =
        (uint16*)VOS_Alloc((uint16) (ucNrOfAttr*sizeof(uint16)));
559.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
560.     service.m_SDCHandle = m_ConnectionInfo.uiSdcHandle;
561.     service.m_ServiceRecordHandle = tServiceSearchCnf-
        >ulServiceRecordHandleList;
562.     m_ServicesFound.SetAtGrow(m_ServiceCounter,service);
563.     SD_RegServiceAttribute(1, m_ConnectionInfo.uiSdcHandle, pulSRHandles[0],
        ucNrOfAttr, puiAttributeIDList);
564.     VOS_Free((void**)&puiAttributeIDList);
565.     VOS_Free((void**)&pulSRHandles);
566. }
567. void CRadioFileServerDlg::OnSdServiceSearchCnfNeg(void **ppMsg)
568. {
569.     SD_TServiceSearchCnfNeg *tConnectCnfNeg = (SD_TServiceSearchCnfNeg
        *)*ppMsg;
570.     CString str;
571.     tConnectCnfNeg = tConnectCnfNeg;
572.     str.Format("Service Search Confirm Negative, Error %d",tConnectCnfNeg-
        >tHdr.iResult);
573.     MessageBox(str);
574. }
575. void CRadioFileServerDlg::OnSdServiceAttributeCnf(void **ppMsg)
576. {
577.     SD_TServiceAttributeCnf *tServiceAttributeCnf = (SD_TServiceAttributeCnf
        *)*ppMsg;
578.     CService service;
579.     switch (tServiceAttributeCnf->tHdr.uiSeqNr)
580.     {
581.     case 1:
582.         ReceiveServiceName(tServiceAttributeCnf);
583.         AskForServiceRecordHandle();
584.         break;
585.     case 2:
586.         ReceiveServiceRecordHandle(tServiceAttributeCnf);
587.         SD_RegDisconnect(0,m_ConnectionInfo.uiSdcHandle);
588.         break;
589.     default:
590.         break;
591.     }
592. }
593. void CRadioFileServerDlg::OnSdServiceAttributeCnfNeg(void **ppMsg)

```

```

594. {
595.     SD_TServiceAttributeCnfNeg *tConnectCnfNeg = (SD_TServiceAttributeCnfNeg
596.         *)*ppMsg;
597.     CString str;
598.     tConnectCnfNeg = tConnectCnfNeg;
599.     str.Format("Service Attribute Confirm negative, error
600.         %d", tConnectCnfNeg->tHdr.iResult);
601.     MessageBox(str);
602. }
603. void CRadioFileServerDlg::OnSdDisconnectCnf(void **ppMsg)
604. {
605.     ppMsg = ppMsg;
606.     Beep (1000,200);
607.     OnSelservices();
608. }
609. void CRadioFileServerDlg::OnDbmRegisterServiceCnf(void **ppMsg)
610. {
611.     DBM_TRegisterServiceCnf *ptRegisterCnf = (DBM_TRegisterServiceCnf *)
612.         *ppMsg;
613.     m_pSerialPort->WriteProfile(ptRegisterCnf->ulDbmHandle);
614.     COM_ReqRegister(PROFILE_SERIAL,
615.         0);
616. }
617. void CRadioFileServerDlg::OnDbmRegisterServiceCnfNeg(void **ppMsg)
618. {
619.     ppMsg = ppMsg;
620.     MessageBox(_T("Could not register to Data Base Manager"));
621.     DestroyWindow();
622. }
623. void CRadioFileServerDlg::OnDbmAddDescriptorCnf(void **ppMsg)
624. {
625.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
626.     tConnectCnfNeg = tConnectCnfNeg;
627.     Beep (1000,200);
628. }
629. void CRadioFileServerDlg::OnDbmAddDescriptorCnfNeg(void **ppMsg)
630. {
631.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
632.     tConnectCnfNeg = tConnectCnfNeg;
633.     MessageBox(_T("Could not register the service to DBM"));
634. }
635. void CRadioFileServerDlg::OnConnectAcceptInd(void **ppMsg)
636. {
637.     SCM_TConnectAcceptInd *ptConnectAcceptInd;
638.     ptConnectAcceptInd = (SCM_TConnectAcceptInd *) *ppMsg;
639.     SCM_RspConnectAccept (MSG_TMsg **)ppMsg,
640.         SCM_POS_RESULT,
641.         ptConnectAcceptInd->tAddress,
642.         SCM_SLAVE);
643.     *ppMsg = NULL;
644. }
645. void CRadioFileServerDlg::OnHciInquiryEvt(void **ppMsg)
646. {
647.     HCI_TInquiryEvt *ptInquiryEvt;
648.     CDevice device;
649.     ptInquiryEvt = (HCI_TInquiryEvt *) *ppMsg;

```

```

647. device.tAddress = ptInquiryEvt->tAddress;
648. device.tPageScanMode = ptInquiryEvt->tPageScanMode;
649. device.tPageScanPeriodMode = ptInquiryEvt->tPageScanPeriodMode;
650. device.tClockOffset = ptInquiryEvt->tClockOffset;
651. device.tCod = ptInquiryEvt->tCod;
652. device.tPageScanRepMode = ptInquiryEvt->tPageScanRepMode;
653. AddDevice(device);
654. }
655. void CRadioFileServerDlg::OnScmPincodeInd(void **ppMsg)
656. {
657.     SCM_TPincodeInd *ptPincodeInd;
658.     ptPincodeInd = (SCM_TPincodeInd *) *ppMsg;
659.     SCM_RspPincode( (MSG_TMsg **)ppMsg,
660.                     SCM_POS_RESULT,
661.                     ptPincodeInd->tAddress,
662.                     _tPincode,
663.                     PINCODE_LENGTH);
664. }
665. void CRadioFileServerDlg::OnScmConnectEvt(void **ppMsg)
666. {
667.     SCM_TConnectEvt *tConnectEvt = (SCM_TConnectEvt *) *ppMsg;
668.     tConnectEvt = tConnectEvt;
669.     m_ConnectionInfo.tAclHandle = tConnectEvt->tHandle;
670.     m_ConnectionInfo.tAddress = tConnectEvt->tAddress;
671. }
672. void CRadioFileServerDlg::OnScmDisconnectEvt(void **ppMsg)
673. {
674.     ppMsg = ppMsg;
675.     m_ConnectionInfo.tAclHandle = 0;
676.     OnCloseapplication();
677. }
678. void CRadioFileServerDlg::OnHciStartCnf(void **ppMsg)
679. {
680.     HCI_TStartCnf *ptStartCnf = (HCI_TStartCnf *) *ppMsg;
681.     ptStartCnf = ptStartCnf;
682.     HCI_ReqConfigurePort(0, PORTSETTINGS);
683. }
684. void CRadioFileServerDlg::OnComVersionCnf(void **ppMsg)
685. {
686.     CAboutDlg Abodlg;
687.     COM_TVersionCnf* ptVersionCnf;
688.     char* cpVerStr = NULL;
689.     int8 iCharCount = 9;
690.     char cpStr[3];
691.     ptVersionCnf = (COM_TVersionCnf *) *ppMsg;
692.     cpVerStr = &ptVersionCnf->cVersion;
693.     do
694.     {
695.         iCharCount++;
696.         cpStr[iCharCount-10] = cpVerStr[iCharCount];
697.     }while(iCharCount <= 11);
698.     cpStr[3] = ((char)0);
699.     Abodlg.DoModal();
700. }
701. void CRadioFileServerDlg::AskForServiceName()
702. {

```

```

703. uint16          *puiAttributeIDList;
704. uint8           ucNrOfAttr;
705.   ucNrOfAttr = 1;
706.   puiAttributeIDList =
       (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof(uint16)));
707.   puiAttributeIDList[0] = BT_SERVICE_NAME(0);
708.   SD_RegServiceAttribute(0, m_ConnectionInfo.uiSdcHandle,
       m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr, puiAttributeIDList);
709.   VOS_Free((void*)&puiAttributeIDList);
710. }
711. void CRadioFileServerDlg::ReceiveServiceName(SD_TServiceAttributeCnf
       *tServiceAttributeCnf)
712. {
713.   CService service;
714.   service = m_ServicesFound.GetAt(m_ServiceCounter);
715.   service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
716.   service.m_AttributeListByteCount = tServiceAttributeCnf-
       >uiAttributeListByteCount;
717.   (void*)memcpy(service.m_pAttributeData,
718.                 &tServiceAttributeCnf->ucAttributeData,
719.                 service.m_AttributeListByteCount);
720.   (void*)memcpy(service.m_pServiceName,
721.                 &service.m_pAttributeData[7],
722.                 service.m_pAttributeData[6]);
723.   service.m_pServiceName[service.m_pAttributeData[6]] = NULL;
724.   m_ServicesFound.SetAt(m_ServiceCounter, service);
725.   m_ServiceCounter++;
726.   m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf-
       >uiAttributeListByteCount;
727.   (void*)memcpy(m_ConnectionInfo.pucAttributeData,
728.                 &tServiceAttributeCnf->ucAttributeData,
729.                 m_ConnectionInfo.uiAttributeListByteCount);
730.   (void*)memcpy(m_ConnectionInfo.pcServiceName,
731.                 &service.m_pAttributeData[7],
732.                 service.m_pAttributeData[6]);
733.   m_ConnectionInfo.pcServiceName[service.m_pAttributeData[6]] = NULL;
734.   ShowAllServicesFound();
735. }
736. void CRadioFileServerDlg::AskForServiceRecordHandle()
737. {
738.   uint16          *puiAttributeIDList;
739.   uint8           ucNrOfAttr;
740.   ucNrOfAttr = 2;
741.   puiAttributeIDList =
       (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof(uint16)));
742.   puiAttributeIDList[0] = BT_SERVICE_RECORD_HANDLE;
743.   puiAttributeIDList[1] = BT_PROTOCOL_DESCRIPTOR_LIST;
744.   SD_RegServiceAttribute(2, m_ConnectionInfo.uiSdcHandle,
       m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr, puiAttributeIDList);
745.   VOS_Free((void*)&puiAttributeIDList);
746. }
747. void CRadioFileServerDlg::ReceiveServiceRecordHandle
       (SD_TServiceAttributeCnf *tServiceAttributeCnf)
748. {
749.   CService service;
750.   service = m_ServicesFound.GetAt(m_ServiceCounter-1);

```

```

751.     service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
752.     service.m_AttributeListByteCount = tServiceAttributeCnf-
        >uiAttributeListByteCount;
753.     (void*)memcpy(service.m_pAttributeData,
754.                   &tServiceAttributeCnf->ucAttributeData,
755.                   service.m_AttributeListByteCount);
756.     m_ServicesFound.SetAt(m_ServiceCounter-1,service);
757.     m_ServiceCounter++;
758.     m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf-
        >uiAttributeListByteCount;
759.     (void*)memcpy(m_ConnectionInfo.pucAttributeData,
760.                   &tServiceAttributeCnf->ucAttributeData,
761.                   m_ConnectionInfo.uiAttributeListByteCount);
762. }
763. void CRadioFileServerDlg::OnCloseapplication()
764. {
765.     SCM_ReqDeRegister(1,SCM_SECURITY_HANDLER);
766. }
767. void CRadioFileServerDlg::OnScmDeRegisterCnf(void **ppMsg)
768. {
769.     SCM_TDeRegisterCnf *ptDeRegisterCnf = (SCM_TDeRegisterCnf *) *ppMsg;
770.     switch (ptDeRegisterCnf->tHdr.uiSeqNr)
771.     {
772.     case 1:
773.         SCM_ReqDeRegister(2,SCM_MONITOR_GROUP);
774.         break;
775.     case 2:
776.         if (m_ConnectionInfo.ulDbmHandle > 0)
777.         {
778.             DBM_ReqUnRegisterService(3,m_ConnectionInfo.ulDbmHandle);
779.         }
780.         else
781.         {
782.             if (m_ConnectionInfo.tAclHandle>0)
783.             {
784.                 SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
785.             }
786.             else
787.             {
788.                 DestroyWindow();
789.             }
790.         }
791.         break;
792.     default:
793.         break;
794.     }
795. }
796. void CRadioFileServerDlg::OnScmDeRegisterCnfNeg(void **ppMsg)
797. {
798.     ppMsg = ppMsg;
799.     MessageBox(_T("Could not unregister from SCM"));
800.     DestroyWindow();
801. }
802. void CRadioFileServerDlg::OnDbmUnRegisterServiceCnf(void **ppMsg)
803. {
804.     ppMsg = ppMsg;

```



```

805.     if (m_ConnectionInfo.tAclHandle>0)
806.     {
807.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
808.     }
809.     else
810.     {
811.         DestroyWindow();
812.     }
813. }
814. void CRadioFileServerDlg::OnDbmUnRegisterServiceCnfNeg(void **ppMsg)
815. {
816.     ppMsg = ppMsg;
817.     MessageBox(_T("Not possible to UnRegister from DBM"));
818.     DestroyWindow();
819. }
820. void CRadioFileServerDlg::OnScmDisconnectCnf(void **ppMsg)
821. {
822.     ppMsg = ppMsg;
823.     m_ConnectionInfo.tAclHandle = 0;
824.     DestroyWindow();
825. }
826. void CRadioFileServerDlg::OnScmDisconnectCnfNeg(void **ppMsg)
827. {
828.     ppMsg = ppMsg;
829.     MessageBox(_T("Could not remove ACL connection"));
830.     DestroyWindow();
831. }
832. BOOL CRadioFileServerDlg::DestroyWindow()
833. {
834.     return CDialog::DestroyWindow();
835. }
836. void CRadioFileServerDlg::ShowAllDevicesFound()
837. {
838.     CDevice device;
839.     int iFound,i;
840.     iFound = m_DevicesFound.GetSize();
841.     for (i=0; i < iFound; i++)
842.     {
843.         device = m_DevicesFound.GetAt(i);
844.         hdevice1=m_tree.InsertItem(device.GetAddress(), hPA, TVI_SORT);
845.         OnSelDevice();
846.     }
847. }
848. void CRadioFileServerDlg::AddService(CString sService)
849. {
850.     CService service(sService);
851.     m_ServicesFound.Add(service);
852. }
853. void CRadioFileServerDlg::AddService(CService service)
854. {
855.     m_ServicesFound.Add(service);
856. }
857. void CRadioFileServerDlg::ShowAllServicesFound()
858. {
859.     CService service;
860.     int iFound,i;

```

```

861.     iFound = m_ServicesFound.GetSize();
862.     for (i=0; i < iFound; i++)
863.     {
864.         service = m_ServicesFound.GetAt(i);
865.         m_tree.InsertItem(service.GetService(),hdevice1,TVI_LAST);
866.     }
867. }
868. void CRadioFileServerDlg::AddDevice(CDevice device)
869. {
870.     m_DevicesFound.Add(device);
871. }
872. void CRadioFileServerDlg::OnInquiry()
873. {
874.     HCI_TLap    tLap = {0x9E,0x8B,0x33};
875.     HCI_TInquiryLength  tInquiryLength = 2;
876.     HCI_TNrOfResponses  tNrOfResponses = 0;
877.     HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses);
878. }
879. void CRadioFileServerDlg::OnSelDevice()
880. {
881.     CDevice device;
882.     device = m_DevicesFound.GetAt(0);
883.     m_ConnectionInfo.tAddress = device.tAddress;
884.     SCM_ReqConnect(0,                                /* I don't make use of the
885.         device.tAddress,
886.         SCM_DM1,
887.         SCM_R1,
888.         SCM_MANDATORY_PAGE_SCAN_MODE,
889.         0,
890.         SCM_NOT_ACCEPT_ROLE_SWITCH);
891. }
892. void CRadioFileServerDlg::OnSelservices()
893. {
894.     CService service;
895.     service = m_ServicesFound.GetAt(0);
896.     m_ConnectionInfo.ulServiceRecordHandle = service.m_ServiceRecordHandle;
897.     DBM_ReqRegisterService(0, DBM_StackDB);
898. }
899. void CRadioFileServerDlg::OnGetservices()
900. {
901.     m_ServiceCounter = 0;
902.     SD_ReqConnect(0,SD_DEFAULT_MFS,m_ConnectionInfo.tAclHandle);
903. }
904. void CRadioFileServerDlg::OnComConnectCnf(void **ppMsg)
905. {
906.     COM_TConnectCnf *ptConnectCnf = (COM_TConnectCnf *) *ppMsg;
907.     m_ConnectionInfo.uiRFCCommHandle = ptConnectCnf->uiHandle;
908.     MessageBox(_T(" RFCOMM connection"));
909.     Beep (1000,200);
910.     Sleep(100);
911.     Beep (1000,200);
912. }
913. void CRadioFileServerDlg::OnComConnectCnfNeg(void **ppMsg)
914. {
915.     COM_TConnectCnfNeg *ptConnectCnfNeg = (COM_TConnectCnfNeg *) *ppMsg;

```

```

916.   ptConnectCnfNeg = ptConnectCnfNeg;
917.
918.   m_ConnectionInfo.uiRFCCommHandle = 0;
919.   MessageBox(_T("Could not create a RFCOMM connection"));
920. }
921. void CRadioFileServerDlg::OnButton2()
922. {
923.   SCM_ReqRegister(0, SCM_MONITOR_GROUP);
924. }
925. void CRadioFileServerDlg::OnScmRegisterCnf(void **ppMsg) //2
926. {
927.   ppMsg = ppMsg;
928.   SDS_ReqStart(0);
929. }
930. void CRadioFileServerDlg::OnScmRegisterCnfNeg(void **ppMsg) //2
931. {
932.   SCM_TRegisterCnfNeg *ptMsg = (SCM_TRegisterCnfNeg *) *ppMsg;
933.   ptMsg = ptMsg;
934.   MessageBox(_T("Not possible to register to SCM"));
935.   DestroyWindow();
936. }
937. void CRadioFileServerDlg::OnSdsStartCnf(void **ppMsg)
938. {
939.   CString sAddress;
940.   SDS_TStartCnf *ptStartCnf;
941.   ptStartCnf = (SDS_TStartCnf *) *ppMsg;
942.   m_pSerialPort = new CRS232(PROFILE_SERIAL);
943. }
944. void CRadioFileServerDlg::OnComRegisterCnf(void **ppMsg)
945. {
946.   COM_TRegisterCnf *ptRegisterCnf = (COM_TRegisterCnf *) *ppMsg;
947.   m_RFServerChannel = ptRegisterCnf->ucServerChannel;
948.   COM_ReqFillPdl(PROFILE_SERIAL,
949.     ptRegisterCnf->ucServerChannel,
950.     (uint16)m_pSerialPort->GetSRPHandle());
951. }
952. void CRadioFileServerDlg::OnComRegisterCnfNeg(void **ppMsg)
953. {
954.   ppMsg = ppMsg;
955.   MessageBox(_T("Not possible to register to RFCOM"));
956.   DestroyWindow();
957. }
958. void CRadioFileServerDlg::OnComFillPdlCnf(void **ppMsg)
959. {
960.   COM_TFillPdlCnf *ptFillPdlCnf = (COM_TFillPdlCnf *) *ppMsg;
961.   char *pcName = NULL;
962.   CString sName;
963.   ptFillPdlCnf = ptFillPdlCnf;
964.   m_pSerialPort->GetProfileName(&pcName);
965.   Beep(1000,100);
966.
967. }
968. void CRadioFileServerDlg::OnComFillPdlCnfNeg(void **ppMsg)
969. {
970.   ppMsg = ppMsg;
971.   MessageBox(_T("Not possible to fill PDL for RF COM"));

```

```

972. DestroyWindow();
973.
974. }
975. void CRadioFileServerDlg::OnComConnectInd(void **ppMsg)
976. {
977.     COM_TConnectInd *tConnectInd = (COM_TConnectInd *)*ppMsg;
978.     m_RFCommHandle = tConnectInd->uiHandle;
979.     COM_RspConnect((MSG_TMsg **)ppMsg, COM_POS_RESULT, tConnectInd-
        >uiMaxFrameSize);
980.     *ppMsg = NULL;
981. }
982. void CRadioFileServerDlg::OnComDataInd(void **ppMsg)
983. {
984.     COM_TDataInd *tDataInd = (COM_TDataInd *)*ppMsg;
985.     uint8 *pucData;
986.     uint16 uiLength;
987.     uint16 uiHandle;
988.     FILE *file;
989.     unsigned char *buffer;
990.     CFile file1, file2;
991.     char buff1[1], *buff2, *buff3;
992.     int fnLength, i, a;
993.
994.     pucData = COM_DataExtract((MSG_TDataMsg *)*ppMsg,
995.                               &uiLength,
996.                               &uiHandle);
997.     COM_RspData(tDataInd->tHdr.ucSeqNr, COM_POS_RESULT, uiHandle);
998.     buffer = (unsigned char *)malloc(uiLength);
999.     file = fopen("temp", "wb+");
1000.     for (i=0; i < uiLength; i++)
1001.     {
1002.         buffer[i] = pucData[i];
1003.         putc(buffer[i], file);
1004.     }
1005.     fclose(file);
1006.     file1.Open("temp", CFile::modeRead, NULL);
1007.     file1.Read((void*)buff1, 1);
1008.     fnLength = atoi(buff1);
1009.     buff2=(char *)malloc(fnLength);
1010.     file1.Read((void*)buff2, fnLength);
1011.     buff2[fnLength] = 0;
1012.     a = file1.GetLength()-1-fnLength;
1013.     buff3=(char *)malloc(a);
1014.     file1.Read((void*)buff3, a);
1015.     file2.Open(buff2, CFile::modeCreate|CFile::modeWrite, NULL);
1016.     file2.Write((void*)buff3, a);
1017.     file1.Close();
1018.     file2.Close();
1019.     DeleteFile("temp");
1020.     CString str;
1021.     str.Format("%s Was Received From Client", buff2);
1022.     m_ChatArea.InsertString(index, (CString)str);
1023.     index++;
1024. }

```

Code Description

- ◆ Lines 1–17: The files required to implement CRadioServerDlg class are included.
- ◆ Lines 19–23: The VC++ Editor adds this code to provide a common framework for the MFC Application Wizard.
- ◆ Line 25: The variables hPA, hdevice1 are required to implement a tree.
- ◆ Lines 26–78: Explained in RadioFileClientDlg.cpp file.
- ◆ Lines 79–84: The destructor is defined to free the memory for member variables and class references.
- ◆ Lines 85–93: Explained in RadioFileClientDlg.cpp file.
- ◆ Lines 94–168: The ON_BLUETOOTH_EVENT message map macro indicates which function will handle a specified Bluetooth event. The following table lists the various Bluetooth events and their associated handler functions.

BLUETOOTH Event	Handler Function Name
COM_DATA_IND	OnComDataInd
COM_REGISTER_CNF	OnComRegisterCnf
COM_REGISTER_CNF_NEG	OnComRegisterCnfNeg
COM_FILL_PDL_CNF	OnComFillPdlCnf
COM_FILL_PDL_CNF_NEG	OnComFillPdlCnfNeg
COM_CONNECT_IND	OnComConnectInd
COM_START_CNF	OnComStartCnf
COM_START_CNF_NEG	OnComStartCnfNeg
COM_VERSION_CNF	OnComVersionCnf
COM_CONNECT_CNF	OnComConnectCnf
COM_CONNECT_CNF_NEG	OnComConnectCnfNeg
SCM_REGISTER_CNF	OnScmRegisterCnf
SCM_REGISTER_CNF_NEG	OnScmRegisterCnfNeg
SCM_CONNECT_ACCEPT_IND	OnConnectAcceptInd
SCM_PINCODE_IND	OnScmPincodeInd
SCM_CONNECT_CNF	OnScmConnectCnf
SCM_CONNECT_CNF_NEG	OnScmConnectCnfNeg
SCM_CONNECT_EVT	OnScmConnectEvt
SCM_DISCONNECT_EVT	OnScmDisconnectEvt
SCM_DISCONNECT_CNF	OnScmDisconnectCnf
SCM_DISCONNECT_CNF_NEG	OnScmDisconnectCnfNeg
SCM_DEREGISTER_CNF	OnScmDeRegisterCnf
SCM_DEREGISTER_CNF_NEG	OnScmDeRegisterCnfNeg
SD_START_CNF	OnSdStartCnf

SD_CONNECT_CNF	OnSdConnectCnf
SD_CONNECT_CNF_NEG	OnSdConnectCnfNeg
SD_SERVICE_SEARCH_CNF	OnSdServiceSearchCnf
SD_SERVICE_SEARCH_CNF_NEG	OnSdServiceSearchCnfNeg
SD_SERVICE_ATTRIBUTE_CNF	OnSdServiceAttributeCnf
SD_SERVICE_ATTRIBUTE_CNF_NEG	OnSdServiceAttributeCnfNeg
SD_DISCONNECT_CNF	OnSdDisconnectCnf
DBM_REGISTER_SERVICE_CNF	OnDbmRegisterServiceCnf
DBM_REGISTER_SERVICE_CNF_NEG	OnDbmRegisterServiceCnfNeg
DBM_UNREGISTER_SERVICE_CNF	OnDbmUnRegisterServiceCnf
DBM_UNREGISTER_SERVICE_CNF_NEG	OnDbmUnRegisterServiceCnfNeg
DBM_ADD_DESCRIPTOR_CNF	OnDbmAddDescriptorCnf
DBM_ADD_DESCRIPTOR_CNF_NEG	OnDbmAddDescriptorCnfNeg
HCI_CONFIGURE_PORT_CNF	OnHciConfigurePortConfirm
HCI_CONFIGURE_PORT_CNF_NEG	OnHciConfigurePortConfirmNegative
HCI_INQUIRY_CNF	OnHciInquiryCnf
HCI_INQUIRY_EVT	OnHciInquiryEvt
HCI_LOCAL_ADDRESS_CNF	OnHciLocalAddressCnf
HCI_LOCAL_ADDRESS_CNF_NEG	OnHciLocalAddressCnfNeg
HCI_REMOTE_NAME_CNF	OnHciRemoteNameCnf
HCI_REMOTE_NAME_CNF_NEG	OnHciRemoteNameCnfNeg
HCI_START_CNF	OnHciStartCnf
HCI_WRITE_SCAN_ENABLE_CNF	OnHciWriteScanEnableCnf
HCI_WRITE_SCAN_ENABLE_CNF_NEG	OnHciWriteScanEnableCnfNeg
HCI_WRITE_AUTHENTICATION_MODE_CNF	OnHciWriteAuthenticationModeCnf
HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG	OnHciWriteAuthenticationModeCnfNeg
HCI_WRITE_ENCRYPTION_MODE_CNF	OnHciWriteEncryptionModeCnf
HCI_WRITE_ENCRYPTION_MODE_CNF_NEG	OnHciWriteEncryptionModeCnfNeg
HCI_WRITE_COD_CNF	OnHciWriteCodCnf
HCI_WRITE_COD_CNF_NEG	OnHciWriteCodCnfNeg
HCI_WRITE_NAME_CNF	OnHciWriteNameCnf

HCI_WRITE_NAME_CNF_NEG	OnHciWriteNameCnfNeg
HCI_WRITE_CONNECT_TIMEOUT_CN F	OnHciWriteConnectTimeoutCnf
HCI_WRITE_CONNECT_TIMEOUT_CN F_NEG	OnHciWriteConnectTimeoutCnfN eg
HCI_WRITE_PAGE_TIMEOUT_CNF	OnHciWritePageTimeoutCnf
HCI_WRITE_PAGE_TIMEOUT_CNF_N EG	OnHciWritePageTimeoutCnfNeg
SIL_SET_DEVICE_CNF	OnSilSetDeviceCnf
SIL_SET_DEVICE_CNF_NEG	OnSilSetDeviceCnfNeg
SIL_REQ_DEVICE_CNF	OnSilReqDeviceCnf
SIL_REQ_DEVICE_CNF_NEG	OnSilReqDeviceCnfNeg
SDS_START_CNF	OnSdsStartCnf

- ◆ Lines 169–606: Explained in `RadioFileClientDlg.cpp` file.
- ◆ Lines 607–641: Explained in `SDPInformationCommandsDlg.cpp` file.
- ◆ Lines 642–654: Explained in `HCIIInformationCommandsDlg.cpp` file.
- ◆ Lines 655–677: Explained in `RadioFileClientDlg.cpp` file.
- ◆ Lines 678–683: Explained in `HCIIInformationCommandsDlg.cpp` file.
- ◆ Lines 684–871: Explained in `RadioFileClientDlg.cpp` file.
- ◆ Lines 872–878: Explained in `HCIIInformationCommandsDlg.cpp` file.
- ◆ Lines 879–920: Explained in `RadioFileClientDlg.cpp` file.
- ◆ Lines 921–924: Request to register Security Manager with the attribute `SCM_MONITOR_GROUP`.
- ◆ Lines 925–929: After the security manager registration is confirmed, the request to start service discovery component is issued.
- ◆ Lines 930–936: If the security manager registration fails, the diagnostic message displays in the message box.
- ◆ Lines 937–943: After the service discovery component start is confirmed, the instance of `CRS232` is created.
- ◆ Lines 944–951: After the `RFCOMM` registration is confirmed, the command to fill the `DBM` component with protocol specified parameters of `COM` component is issued.
- ◆ Lines 952–957: If the `RFCOMM` registrations are not confirmed, the diagnostic message is displayed.
- ◆ Lines 958–967: After filling up the `DBM` component with `COM` specific parameters is finished, remote device name is retrieved.
- ◆ Lines 968–974: If filling up the `DBM` component with `COM` specific parameters fails, the diagnostic message will be displayed on the message box.
- ◆ Lines 975–981: Explained in `CradioFileClientDlg.cpp` file.
- ◆ Lines 982–1024: When the data arrival indication has been received by the server, it gets data from the remote Bluetooth device. It sends the response to indicate that the data from the remote device has been received successfully. A temporary file is opened in write mode to copy the received data into that file and then closed. The temporary file is opened in read mode. The first character is checked to see the length of the filename. The filename and the length of the file are obtained. A

file with the received filename is opened in write mode. The data is read from the temporary file and copied into the created file. The two files are closed. The temporary file is deleted. The status of file transmission is displayed in the dialog box.

Code Output

At the server side, if the application is built in the VC++ environment, the window in Figure 9-8 is displayed.

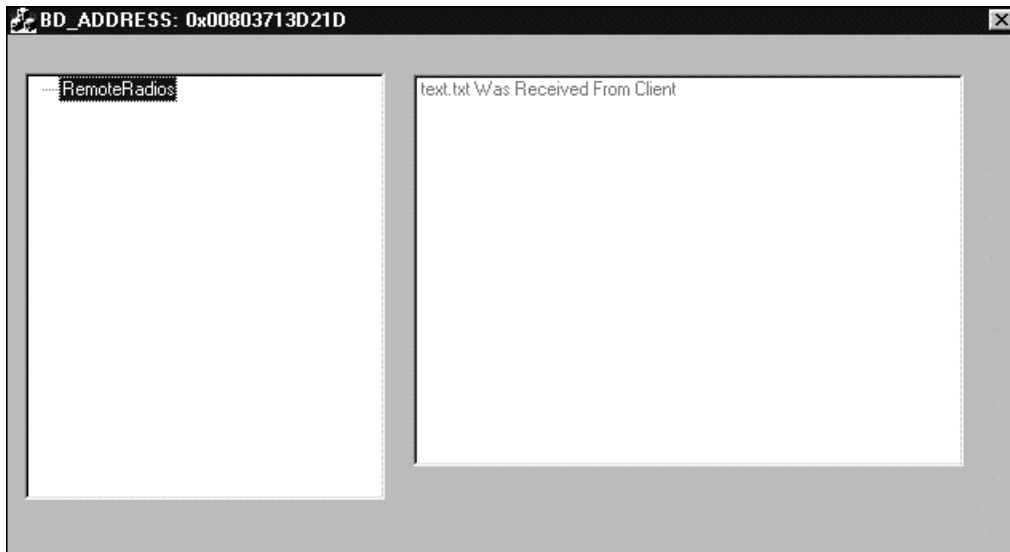


Figure 9-8: Output of the Server Module

The right portion of this server-side screen displays the message that the file has been received. The file received from the other Bluetooth device will be stored here with the same name. Also note that in this application, the file size is limited to the RFCOMM packet size, meaning 127 bytes. If larger files are to be transferred, the following modifications are required: In `product.h` (of the PC reference stack), `COM_MAX_MFS` has to be set to 32K.

In this example, we transferred a text file from one device to another. This text file can be a WML file containing a set of WML cards. When the WML code is received by the Bluetooth device, it can be displayed on a WML browser or emulator. The examples we discussed in Chapter 8 essentially work on this principle — a WAP server discovers the nearby Bluetooth devices and does a file transfer of the WML file to be displayed on the WAP browser. The browser interprets this file and the WML cards are displayed.

Application: Chat

In this application, we create a text chat utility through which two Bluetooth-enabled PCs can exchange small messages. The software contains the following three modules:

- ◆ **Common Module** (common to both server and client): The common module provides a mechanism to setup ACL link and to access the service. This module is implemented with the classes `Cservice` and `CconnectionInfo` in the files named `Service.cpp` and `ConnectionInfo.cpp` respectively. The code is same as that of the file transfer application.
- ◆ **Client module**: The client module gets the service from the server and exchanges the data over RFCOMM. The client module is implemented with the class `CRadioChatClientDlg` in the file named `RadioChatClientDlg.cpp`.

- ◆ Server module: The server module registers a service with DBM. The chat service is made available to all nearby clients. The server module is implemented with the class `CRadioChatServerDlg` in the file named `RadioChatServerDlg.cpp`.

Client Module

Listings 9-21 and 9-22 give the source code for `RadioChatClientDlg.h` and `RadioChatClientDlg.cpp`, respectively.

Listing 9-21: RadiochatClientDlg.h

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. // RadioChatClientDlg.h : header file
2.
3. #include "Events.h"
4. #include "ConnectionInfo.h"
5. #include "Remotedevice.h"
6. #include "service.h"
7. #include <exp\sd.h>
8. #include <exp\BT_COMServer.h>
9. #include <afxtempl.h>
10.
11. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
12. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
13.     { WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
14.         (AFX_PMSG) (AFX_PMSGW) (void (AFX_MSG_CALL CWnd::*) (void
15.         **)) &memberFxn },
16. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
17.     SendMessage ( (HWND) this-
18.         > m_hWnd, WM_BLUETOOTH_EVENT, (WPARAM) uiBtEventID, (LPARAM) &pMsg)
19.
20. class CRadioChatClientDlg : public CDialog
21. {
22. public:
23.     CRadioChatClientDlg(CWnd* pParent = NULL); // standard constructor
24.     ~CRadioChatClientDlg();
25.     CConnectionInfo m_ConnectionInfo;
26. private:
27.     void InitSecurityClient();
28.     BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
29.     //{AFX_DATA(CRadioFileClientDlg)
30.     enum { IDD = IDD_RADIOCHAT_DIALOG };
31.     CListBox      m_ChatArea;
32.     CEdit m_InputChat;
33.     CTreeCtrl      m_tree;
34.     //{AFX_DATA
35.     //{AFX_VIRTUAL(CRadioFileClientDlg)
36. public:
37.     virtual BOOL DestroyWindow();
38. protected:
39.     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
40.     virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);
41.     //{AFX_VIRTUAL
42. public:
43.     void HandleReturn();

```

```

42. void AddDevice(CString sAddress, CString sName);
43. void AddDevice(CDevice device);
44. void ShowAllDevicesFound();
45. void AddService(CString sService);
46. void AddService(CService service);
47. void ShowAllServicesFound();
48. void AskForServiceName();
49. void ReceiveServiceName(SD_TServiceAttributeCnf *tServiceAttributeCnf);
50. void AskForServiceRecordHandle();
51. void ReceiveServiceRecordHandle(SD_TServiceAttributeCnf
*tServiceAttributeCnf);
52. protected:
53. HICON m_hIcon;
54. int index;
55. Events *m_pServerEvents;
56. CArray <CDevice,CDevice> m_DevicesFound;
57. CArray <CService,CService> m_ServicesFound;
58. int m_RemoteNameCounter;
59. int m_ServiceCounter;
60. //{AFX_MSG(CRadioChatClientDlg)
61. virtual BOOL OnInitDialog();
62. afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
63. afx_msg void OnPaint();
64. afx_msg HCURSOR OnQueryDragIcon();
65. afx_msg void OnInquiry();
66. afx_msg void OnSelDevice();
67. afx_msg void OnConnect();
68. afx_msg void OnGetServices();
69. afx_msg void OnSelServices();
70. afx_msg void OnDestroy();
71. afx_msg void OnSerialport();
72. afx_msg void OnCloseapplication();
73. afx_msg void OnHCISerial();
74. afx_msg void OnHCIUsb();
75. afx_msg void OnComStartCnf(void **ppMsg);
76. afx_msg void OnComStartCnfNeg(void **ppMsg);
77. afx_msg void OnComVersionCnf(void **ppMsg);
78. afx_msg void OnScmRegisterCnf(void **ppMsg);
79. afx_msg void OnScmRegisterCnfNeg(void **ppMsg);
80. afx_msg void OnConnectAcceptInd(void **ppMsg);
81. afx_msg void OnScmPincodeInd(void **ppMsg);
82. afx_msg void OnScmConnectCnf(void **ppMsg);
83. afx_msg void OnScmConnectCnfNeg(void **ppMsg);
84. afx_msg void OnScmConnectEvt(void **ppMsg);
85. afx_msg void OnScmDisconnectEvt(void **ppMsg);
86. afx_msg void OnScmDisconnectCnf(void **ppMsg);
87. afx_msg void OnScmDisconnectCnfNeg(void **ppMsg);
88. afx_msg void OnScmDeRegisterCnf(void **ppMsg);
89. afx_msg void OnScmDeRegisterCnfNeg(void **ppMsg);
90. afx_msg void OnSdStartCnf(void **ppMsg);
91. afx_msg void OnSdConnectCnf(void **ppMsg);
92. afx_msg void OnSdConnectCnfNeg(void **ppMsg);
93. afx_msg void OnSdServiceSearchCnf(void **ppMsg);
94. afx_msg void OnSdServiceSearchCnfNeg(void **ppMsg);
95. afx_msg void OnSdServiceAttributeCnf(void **ppMsg);
96. afx_msg void OnSdServiceAttributeCnfNeg(void **ppMsg);

```

```

97.  afx_msg void OnSdDisconnectCnf(void **ppMsg);
98.  afx_msg void OnDbmRegisterServiceCnf(void **ppMsg);
99.  afx_msg void OnDbmRegisterServiceCnfNeg(void **ppMsg);
100. afx_msg void OnDbmUnRegisterServiceCnf(void **ppMsg);
101. afx_msg void OnDbmUnRegisterServiceCnfNeg(void **ppMsg);
102. afx_msg void OnDbmAddDescriptorCnf(void **ppMsg);
103. afx_msg void OnDbmAddDescriptorCnfNeg(void **ppMsg);
104. afx_msg void OnHciConfigurePortConfirm(void **ppMsg);
105. afx_msg void OnHciConfigurePortConfirmNegative(void **ppMsg);
106. afx_msg void OnHciInquiryCnf(void **ppMsg);
107. afx_msg void OnHciInquiryEvt(void **ppMsg);
108. afx_msg void OnHciLocalAddressCnf(void **ppMsg);
109. afx_msg void OnHciLocalAddressCnfNeg(void **ppMsg);
110. afx_msg void OnHciRemoteNameCnf(void **ppMsg);
111. afx_msg void OnHciRemoteNameCnfNeg(void **ppMsg);
112. afx_msg void OnHciStartCnf(void **ppMsg);
113. afx_msg void OnHciWriteScanEnableCnf(void **ppMsg);
114. afx_msg void OnHciWriteScanEnableCnfNeg(void **ppMsg);
115.
116.
117. afx_msg void OnHciWriteAuthenticationModeCnf(void **ppMsg);
118. afx_msg void OnHciWriteAuthenticationModeCnfNeg(void **ppMsg);
119. afx_msg void OnHciWriteEncryptionModeCnf(void **ppMsg);
120. afx_msg void OnHciWriteEncryptionModeCnfNeg(void **ppMsg);
121. afx_msg void OnHciWriteCodCnf(void **ppMsg);
122. afx_msg void OnHciWriteCodCnfNeg(void **ppMsg);
123. afx_msg void OnHciWriteNameCnf(void **ppMsg);
124. afx_msg void OnHciWriteNameCnfNeg(void **ppMsg);
125. afx_msg void OnHciWriteConnectTimeoutCnf(void **ppMsg);
126. afx_msg void OnHciWriteConnectTimeoutCnfNeg(void **ppMsg);
127. afx_msg void OnHciWritePageTimeoutCnf(void **ppMsg);
128. afx_msg void OnHciWritePageTimeoutCnfNeg(void **ppMsg);
129. afx_msg void OnSilSetDeviceCnf(void **ppMsg);
130. afx_msg void OnSilSetDeviceCnfNeg(void **ppMsg);
131. afx_msg void OnSilReqDeviceCnf(void **ppMsg);
132. afx_msg void OnSilReqDeviceCnfNeg(void **ppMsg);
133. afx_msg void OnComConnectCnf(void **ppMsg);
134. afx_msg void OnComConnectCnfNeg(void **ppMsg);
135. afx_msg void OnComDataInd(void **ppMsg);
136. afx_msg void OnComDataCnf(void **ppMsg);
137. afx_msg void OnComDataCnfNeg(void **ppMsg);
138. afx_msg void OnComDisconnectEvt(void **ppMsg);
139. afx_msg void OnComDisconnectCnf(void **ppMsg);
140. afx_msg void OnComDisconnectCnfNeg(void **ppMsg);
141. afx_msg void OnBrowse();
142. //}}AFX_MSG
143. DECLARE_MESSAGE_MAP()
144. };
145.
146.

```

Code Description

This header file contains all variables, member functions, and classes required to implement the class `CRadioChatClientDlgcpp`.

Listing 9-22: RadioChatClientDlg.cpp

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. // RadioChatClientDlg.cpp : implementation file
2. #include "stdafx.h"
3. #include "RadioChat.h"
4. #include "RadioChatClientDlg.h"
5. #include "Events.h"
6. #include <process.h>
7. #include "windows.h"
8. #include <exp/msg.h>
9. #include <exp/hci.h>
10. #include <exp/hci_drv.h>
11. #include <exp/scm.h>
12. #include <exp/com.h>
13. #include <exp/dbm.h>
14. #include <exp/sd.h>
15. #include <exp/vos2com.h>
16. #include <exp/sil.h>
17.
18. #ifdef _DEBUG
19. #define new DEBUG_NEW
20. #undef THIS_FILE
21. static char THIS_FILE[] = __FILE__;
22. #endif
23.
24. HTREEITEM hPA,hdevice1;
25. union MessageMapFunctions
26. {
27.     AFX_PMSG pfn;
28.     void      (AFX_MSG_CALL CWnd::*pfn_btfn)(void **);
29. };
30. #define PINCODE_LENGTH ((SCM_TPIncodeLength) 4)
31. static const SCM_TPIncode _tPincode =
        {'1','2','3','4','0','0','0','0','0','0','0','0','0','0','0','0','0','0'};
32. #define PORTSETTINGS (uint8 *)("COM1:Baud=57600 parity=N data=8
                                     stop=1")
33. #define InterSelSerial ((uint8) 0)
34. #define InterSelUSB      ((uint8) 1)
35. #define SRP_SERIAL_GENERIC_SERIALPORT_UUID ((uint16) 0x1101)
36. static const HCI_TCod _tCod={0x20,0x04,0x04};
37.
38. class CAboutDlg : public CDialog
39. {
40. public:
41.     CAboutDlg();
42.     //{AFX_DATA(CAboutDlg)
43.     enum { IDD = IDD_ABOUTBOX };
44.     //{AFX_DATA
45.     //{AFX_VIRTUAL(CAboutDlg)
46. protected:
47.     virtual void DoDataExchange(CDataExchange* pDX);
48.     //{AFX_VIRTUAL
49. protected:
50.     //{AFX_MSG(CAboutDlg)

```

```

51.  //}}AFX_MSG
52.  DECLARE_MESSAGE_MAP()
53.  };
54. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
55. {
56.  //{{AFX_DATA_INIT(CAboutDlg)
57.  //}}AFX_DATA_INIT
58. }
59. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
60. {
61.  CDialog::DoDataExchange(pDX);
62.  //{{AFX_DATA_MAP(CAboutDlg)
63.  //}}AFX_DATA_MAP
64. }
65. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
66.  //{{AFX_MSG_MAP(CAboutDlg)
67.  //}}AFX_MSG_MAP
68. END_MESSAGE_MAP()
69.
70. CRadioChatClientDlg::CRadioChatClientDlg(CWnd* pParent /*=NULL*/)
71. : CDialog(CRadioChatClientDlg::IDD, pParent)
72. {
73.  //{{AFX_DATA_INIT(CRadioChatClientDlg)
74.  //}}AFX_DATA_INIT
75.  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
76.  m_pServerEvents = new Events();
77. }
78. CRadioChatClientDlg::~CRadioChatClientDlg()
79. {
80.  m_DevicesFound.RemoveAll();
81.  m_ServicesFound.RemoveAll();
82.  delete m_pServerEvents;
83. }
84. void CRadioChatClientDlg::DoDataExchange(CDataExchange* pDX)
85. {
86.  CDialog::DoDataExchange(pDX);
87.  //{{AFX_DATA_MAP(CRadioChatClientDlg)
88.  DDX_Control(pDX, IDC_LIST1, m_CharArea);
89.  DDX_Control(pDX, IDC_EDIT1, m_InputChat);
90.  DDX_Control(pDX, IDC_TREE1, m_tree);
91.  //}}AFX_DATA_MAP
92. }
93. BEGIN_MESSAGE_MAP(CRadioChatClientDlg, CDialog)
94.  //{{AFX_MSG_MAP(CRadioChatClientDlg)
95.  ON_WM_SYSCOMMAND()
96.  ON_WM_PAINT()
97.  ON_WM_QUERYDRAGICON()
98.  ON_BLUETOOTH_EVENT(COM_START_CNF, OnComStartCnf)
99.  ON_BLUETOOTH_EVENT(COM_START_CNF_NEG, OnComStartCnfNeg)
100.  ON_BLUETOOTH_EVENT(COM_VERSION_CNF, OnComVersionCnf)
101.  ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF, OnScmRegisterCnf)
102.  ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF_NEG, OnScmRegisterCnfNeg)
103.  ON_BLUETOOTH_EVENT(SCM_CONNECT_ACCEPT_IND, OnConnectAcceptInd)
104.  ON_BLUETOOTH_EVENT(SCM_PINCODE_IND, OnScmPincodeInd)
105.  ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF, OnScmConnectCnf)
106.  ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF_NEG, OnScmConnectCnfNeg)

```

```

107. ON_BLUETOOTH_EVENT(SCM_CONNECT_EVT, OnScmConnectEvt)
108. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_EVT, OnScmDisconnectEvt)
109. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF, OnScmDisconnectCnf)
110. ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF_NEG, OnScmDisconnectCnfNeg)
111. ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF, OnScmDeRegisterCnf)
112. ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF_NEG, OnScmDeRegisterCnfNeg)
113. ON_BLUETOOTH_EVENT(SD_START_CNF, OnSdStartCnf)
114. ON_BLUETOOTH_EVENT(SD_CONNECT_CNF, OnSdConnectCnf)
115. ON_BLUETOOTH_EVENT(SD_CONNECT_CNF_NEG, OnSdConnectCnfNeg)
116. ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF, OnSdServiceSearchCnf)
117. ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF_NEG,
    OnSdServiceSearchCnfNeg)
118. ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF, OnSdServiceAttributeCnf)
119. ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF_NEG,
    OnSdServiceAttributeCnfNeg)
120. ON_BLUETOOTH_EVENT(SD_DISCONNECT_CNF, OnSdDisconnectCnf)
121. ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF, OnDbmRegisterServiceCnf)
122. ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF_NEG,
    OnDbmRegisterServiceCnfNeg)
123. ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF,
    OnDbmUnRegisterServiceCnf)
124. ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF_NEG,
    OnDbmUnRegisterServiceCnfNeg)
125. ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF, OnDbmAddDescriptorCnf)
126. ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF_NEG,
    OnDbmAddDescriptorCnfNeg)
127. ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF, OnHciConfigurePortConfirm)
128. ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF_NEG,
    OnHciConfigurePortConfirmNegative)
129. ON_BLUETOOTH_EVENT(HCI_INQUIRY_CNF, OnHciInquiryCnf)
130. ON_BLUETOOTH_EVENT(HCI_INQUIRY_EVT, OnHciInquiryEvt)
131. ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF, OnHciLocalAddressCnf)
132. ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF_NEG,
    OnHciLocalAddressCnfNeg)
133. ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF, OnHciRemoteNameCnf)
134. ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF_NEG, OnHciRemoteNameCnfNeg)
135. ON_BLUETOOTH_EVENT(HCI_START_CNF, OnHciStartCnf)
136. ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF,
    OnHciWriteScanEnableCnf)
137. ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF_NEG,
    OnHciWriteScanEnableCnfNeg)
138.
139.
140. ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF,
    OnHciWriteAuthenticationModeCnf)
141. ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG,
    OnHciWriteAuthenticationModeCnfNeg)
142. ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF,
    OnHciWriteEncryptionModeCnf)
143. ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF_NEG,
    OnHciWriteEncryptionModeCnfNeg)
144. ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF, OnHciWriteCodCnf)
145. ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF_NEG, OnHciWriteCodCnfNeg)
146. ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF, OnHciWriteNameCnf)
147. ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF_NEG, OnHciWriteNameCnfNeg)

```

```

148.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF,
                           OnHciWriteConnectTimeoutCnf)
149.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG,
                           OnHciWriteConnectTimeoutCnfNeg)
150.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF,
                           OnHciWritePageTimeoutCnf)
151.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF_NEG,
                           OnHciWritePageTimeoutCnfNeg)
152.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF, OnSilSetDeviceCnf)
153.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF_NEG, OnSilSetDeviceCnfNeg)
154.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF, OnSilReqDeviceCnf)
155.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF_NEG, OnSilReqDeviceCnfNeg)
156.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF, OnComConnectCnf )
157.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF_NEG, OnComConnectCnfNeg )
158.     ON_BLUETOOTH_EVENT(COM_DATA_IND, OnComDataInd )
159.     ON_BLUETOOTH_EVENT(COM_DATA_CNF, OnComDataCnf )
160.     ON_BLUETOOTH_EVENT(COM_DATA_CNF_NEG, OnComDataCnfNeg )
161.     ON_BLUETOOTH_EVENT(COM_DISCONNECT_EVT, OnComDisconnectEvt )
162.     ON_BLUETOOTH_EVENT(COM_DISCONNECT_CNF, OnComDisconnectCnf )
163.     ON_BLUETOOTH_EVENT(COM_DISCONNECT_CNF_NEG, OnComDisconnectCnfNeg )
164.     ///}AFX_MSG_MAP
165. END_MESSAGE_MAP()
166. BOOL CRadioChatClientDlg::OnInitDialog()
167. {
168.     CDialog::OnInitDialog();
169.     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
170.     ASSERT(IDM_ABOUTBOX < 0xF000);
171.     CMenu* pSysMenu = GetSystemMenu(FALSE);
172.     if (pSysMenu != NULL)
173.     {
174.         CString strAboutMenu;
175.         strAboutMenu.LoadString(IDS_ABOUTBOX);
176.         if (!strAboutMenu.IsEmpty())
177.         {
178.             pSysMenu->AppendMenu(MF_SEPARATOR);
179.             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
180.         }
181.     }
182.     SetIcon(m_hIcon, TRUE);
183.     SetIcon(m_hIcon, FALSE);
184.     m_pServerEvents->m_pParentDialog = this;
185.     TVINSERTSTRUCT tvInsert;
186.     tvInsert.hParent = NULL;
187.     tvInsert.hInsertAfter = NULL;
188.     tvInsert.item.mask = TVIF_TEXT;
189.     tvInsert.item.pszText = _T("RemoteRadios");
190.     hPA = m_tree.InsertItem(&tvInsert);
191.     InitSecurityClient();
192.     return TRUE;
193. }
194. LRESULT CRadioChatClientDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
195. {
196.     MSG_TMsg **ptMsg;
197.     if (message == WM_BLUETOOTH_EVENT)
198.     {

```

```

199.     OnBluetoothEvent( message, wParam, lParam);
200.     ptMsg = (MSG_TMsg**)lParam;
201.     if (*ptMsg != NULL)
202.         VOS_Free((void **)lParam);
203. }
204. return CDialog::WindowProc(message, wParam, lParam);
205. }
206. BOOL CRadioChatClientDlg::OnBluetoothEvent(UINT message, WPARAM wParam,
                                           LPARAM lParam)
207. {
208.     const AFX_MSGMAP* pMessageMap;
209.     const AFX_MSGMAP_ENTRY* lpEntry;
210. #ifdef _AFXDLL
211.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
212.         pMessageMap = (*pMessageMap->pfnGetBaseMap)())
213. #else
214.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
215.         pMessageMap = pMessageMap->pBaseMap)
216. #endif
217.     {
218. #ifdef _AFXDLL
219.         ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
220. #else
221.         ASSERT(pMessageMap != pMessageMap->pBaseMap);
222. #endif
223.         lpEntry = (AFX_MSGMAP_ENTRY*)&pMessageMap->lpEntries[0];
224.         while (lpEntry->nSig != AfxSig_end)
225.         {
226.             if((lpEntry->nMessage==message)&&(lpEntry->nCode== wParam))
227.             {
228.                 union MessageMapFunctions mmf;
229.                 mmf.pfn = lpEntry->pfn;
230.                 (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
231.                 return TRUE;
232.             }
233.             lpEntry++;
234.         }
235.         return FALSE;
236.     }
237.     return FALSE;
238. }
239. void CRadioChatClientDlg::OnSysCommand(UINT nID, LPARAM lParam)
240. {
241.     if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
242.     {
243.         CAboutDlg dlgAbout;
244.         dlgAbout.DoModal();
245.     }
246.     else
247.     {
248.         CDialog::OnSysCommand(nID, lParam);
249.     }
250. }
251. void CRadioChatClientDlg::OnPaint()
252. {
253.     if (IsIconic())

```



```

254. {
255.     CPaintDC dc(this);
256.     SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
257.     int cxIcon = GetSystemMetrics(SM_CXICON);
258.     int cyIcon = GetSystemMetrics(SM_CYICON);
259.     CRect rect;
260.     GetClientRect(&rect);
261.     int x = (rect.Width() - cxIcon + 1) / 2;
262.     int y = (rect.Height() - cyIcon + 1) / 2;
263.     dc.DrawIcon(x, y, m_hIcon);
264. }
265. else
266. {
267.     CDialog::OnPaint();
268. }
269. }
270. HCURSOR CRadioChatClientDlg::OnQueryDragIcon()
271. {
272.     return (HCURSOR) m_hIcon;
273. }
274. void CRadioChatClientDlg::InitSecurityClient()
275. {
276.     SIL_SetDevice(0, SIL_SERIAL);
277. }
278. void CRadioChatClientDlg::OnSilSetDeviceCnf(void **ppMsg)
279. {
280.     ppMsg = ppMsg;
281.     HCI_ReqConfigurePort(0, PORTSETTINGS);
282. }
283. void CRadioChatClientDlg::OnSilSetDeviceCnfNeg(void **ppMsg)
284. {
285.     SIL_TSetDevice* ptSetDevice;
286.     ptSetDevice = (SIL_TSetDevice*)*ppMsg;
287.     if(ptSetDevice->tHdr.iResult == SIL_ERR_DEVICE)
288.         SIL_ReqDevice(0);
289. }
290. void CRadioChatClientDlg::OnSilReqDeviceCnf(void **ppMsg)
291. {
292.     SIL_TReqDevice* ptReq;
293.     ptReq = (SIL_TReqDevice*) *ppMsg;
294.     if(ptReq->uiDevice == SIL_SERIAL)
295.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
                Interface is SERIAL"));
296.     if(ptReq->uiDevice == SIL_USB)
297.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI
                Interface is USB"));
298. }
299. void CRadioChatClientDlg::OnSilReqDeviceCnfNeg(void **ppMsg)
300. {
301.     ppMsg = ppMsg;
302.     MessageBox(_T("Device Request FAILED!"));
303. }
304. void CRadioChatClientDlg::OnHciConfigurePortConfirm(void **ppMsg)
305. {
306.     HCI_TConfigurePortCnf *tConfigurePort=(HCI_TConfigurePortCnf
                *)*ppMsg;

```

```

307.     tConfigurePort = tConfigurePort;
308.     COM_ReqStart(0);
309. }
310. void CRadioChatClientDlg::OnHciConfigurePortConfirmNegative(void **ppMsg)
311. {
312.     HCI_TConfigurePortCnfNeg *tConfigurePort = (HCI_TConfigurePortCnfNeg
                                                    *)*ppMsg;
313.     tConfigurePort = tConfigurePort;
314.     MessageBox(_T("Could not open port"));
315. }
316. void CRadioChatClientDlg::OnComStartCnf(void **ppMsg)
317. {
318.     COM_TStartCnf *tStartCnf = (COM_TStartCnf *)*ppMsg;
319.     tStartCnf = tStartCnf;
320.     HCI_ReqLocalAddress(0);
321. }
322. void CRadioChatClientDlg::OnComStartCnfNeg(void **ppMsg)
323. {
324.     COM_TStartCnfNeg *tStartCnfNeg = (COM_TStartCnfNeg *)*ppMsg;
325.     tStartCnfNeg = tStartCnfNeg;
326.     MessageBox(_T("Could not start RFCOMM"));
327. }
328. void CRadioChatClientDlg::OnHciLocalAddressCnf(void **ppMsg)
329. {
330.     HCI_TLocalAddressCnf *tLocalAddress = (HCI_TLocalAddressCnf
                                                    *)*ppMsg;
331.     char lpStr[59];
332.     wsprintf(&lpStr[0], "BD_ADDRESS: 0x%02X%02X%02X%02X%02X%02X\\0",
333.             tLocalAddress->tAddress.ucByte0,
334.             tLocalAddress->tAddress.ucByte1,
335.             tLocalAddress->tAddress.ucByte2,
336.             tLocalAddress->tAddress.ucByte3,
337.             tLocalAddress->tAddress.ucByte4,
338.             tLocalAddress->tAddress.ucByte5);
339.     SetWindowText(_T(lpStr));
340.     SD_ReqStart(0);
341. }
342. void CRadioChatClientDlg::OnHciLocalAddressCnfNeg(void **ppMsg)
343. {
344.     ppMsg = ppMsg;
345.     SetWindowText(_T("DEVICE NOT FOUND"));
346.     SD_ReqStart(0);
347. }
348. void CRadioChatClientDlg::OnSdStartCnf(void **ppMsg)
349. {
350.     ppMsg = ppMsg;
351.     HCI_ReqWriteEncryptionMode(0, HCI_ENCRYPTION_OFF);
352. }
353. void CRadioChatClientDlg::OnHciWriteEncryptionModeCnf(void **ppMsg)
354. {
355.     ppMsg = ppMsg;
356.     HCI_ReqWriteAuthenticationMode(0, HCI_AUTH_DISABLE);
357. }
358. void CRadioChatClientDlg::OnHciWriteEncryptionModeCnfNeg(void **ppMsg)
359. {
360.     ppMsg = ppMsg;

```



```

416. }
417. void CRadioChatClientDlg::OnHciWriteNameCnfNeg(void **ppMsg)
418. {
419.     ppMsg = ppMsg;
420. }
421. void CRadioChatClientDlg::OnHciWriteScanEnableCnf(void **ppMsg)
422. {
423.     ppMsg = ppMsg;
424.     SCM_ReqRegister(0, SCM_SECURITY_HANDLER);
425. }
426. void CRadioChatClientDlg::OnHciWriteScanEnableCnfNeg(void **ppMsg)
427. {
428.     ppMsg = ppMsg;
429. }
430. void CRadioChatClientDlg::OnScmRegisterCnf(void **ppMsg)
431. {
432.     SCM_TRegisterCnf *tRegisterCnf = (SCM_TRegisterCnf *)*ppMsg;
433.     tRegisterCnf = tRegisterCnf;
434.     if (tRegisterCnf->tHdr.uiSeqNr == 0)
435.     {
436.         SCM_ReqRegister(1, SCM_MONITOR_GROUP);
437.     }
438.     else
439.     {
440.         OnInquiry();
441.     }
442. }
443. void CRadioChatClientDlg::OnScmRegisterCnfNeg(void **ppMsg)
444. {
445.     SCM_TRegisterCnfNeg *tRegisterCnfNeg = (SCM_TRegisterCnfNeg *)*ppMsg;
446.     tRegisterCnfNeg = tRegisterCnfNeg;
447.     MessageBox(_T("Could not register to SCM"));
448. }
449. void CRadioChatClientDlg::OnHciInquiryCnf(void **ppMsg)
450. {
451.     HCI_TInquiryCnf *ptInquiryCnf;
452.     int count;
453.     CRemoteDevice device;
454.     ptInquiryCnf = (HCI_TInquiryCnf *) *ppMsg;
455.     count = m_DevicesFound.GetSize();
456.     m_RemoteNameCounter = 0;
457.     if (count > 0)
458.     {
459.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
460.         HCI_ReqRemoteName(10,
461.             device.tAddress,
462.             device.tPageScanPeriodMode,
463.             device.tPageScanMode,
464.             device.tClockOffset );
465.     }
466.     else
467.     {
468.         AfxMessageBox("No device found");
469.     }
470. }
471. void CRadioChatClientDlg::OnHciRemoteNameCnf(void **ppMsg)

```

```

472. {
473.     HCI_TRemoteNameCnf      *ptRemoteNameCnf;
474.     CRemoteDevice device;
475.     char sName[248];
476.     int count;
477.     ptRemoteNameCnf =(HCI_TRemoteNameCnf *) *ppMsg;
478.     sprintf(sName,"%s",&ptRemoteNameCnf->tName);
479.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)sName);
480.     m_RemoteNameCounter++;
481.     count = m_DevicesFound.GetSize();
482.     if (count > m_RemoteNameCounter)
483.     {
484.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
485.         HCI_ReqRemoteName(10,
486.                             device.tAddress,
487.                             device.tPageScanPeriodMode,
488.                             device.tPageScanMode,
489.                             device.tClockOffset );
490.     }
491.     else
492.     {
493.         ShowAllDevicesFound();
494.     }
495. }
496. void CRadioChatClientDlg::OnHciRemoteNameCnfNeg(void **ppMsg)
497. {
498.     HCI_TRemoteNameCnfNeg      *ptRemoteNameCnfNeg;
499.     CDevice device;
500.     int count;
501.     ptRemoteNameCnfNeg =(HCI_TRemoteNameCnfNeg *) *ppMsg;
502.     m_DevicesFound[m_RemoteNameCounter].SetName((CString)_T("UNKNOWN"));
503.     m_RemoteNameCounter++;
504.     count = m_DevicesFound.GetSize();
505.     if (count > m_RemoteNameCounter)
506.     {
507.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
508.         HCI_ReqRemoteName(10,
509.                             device.tAddress,
510.                             device.tPageScanPeriodMode,
511.                             device.tPageScanMode,
512.                             device.tClockOffset );
513.     }
514.     else
515.     {
516.         ShowAllDevicesFound();
517.     }
518. }
519. void CRadioChatClientDlg::OnScmConnectCnf(void **ppMsg)
520. {
521.     SCM_TConnectCnf *tConnectCnf = (SCM_TConnectCnf *)*ppMsg;
522.     AfxMessageBox("connected");
523.     tConnectCnf = tConnectCnf;
524.     m_ConnectionInfo.tAclHandle = tConnectCnf->tHandle;
525.     m_ConnectionInfo.tAddress = tConnectCnf->tAddress;
526.     OnGetservices();
527. }

```

```

528. void CRadioChatClientDlg::OnScmConnectCnfNeg(void **ppMsg)
529. {
530.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
531.
532.     tConnectCnfNeg = tConnectCnfNeg;
533.     AfxMessageBox("No Connection made");
534. }
535. void CRadioChatClientDlg::OnSdConnectCnf(void **ppMsg)
536. {
537.     SD_TConnectCnf *tConnectCnf = (SD_TConnectCnf *)*ppMsg;
538.     SD_TUUid          *ptSearchPatternList;
539.     uint16             uiMaxRecords;
540.     uint8              ucNrOfUuids;
541.     m_ConnectionInfo.uiSdcHandle = tConnectCnf->uiSdcHandle;
542.     uiMaxRecords = 6;
543.     ucNrOfUuids = 1;
544.     ptSearchPatternList = (SD_TUUid*)VOS_Alloc((uint16)(ucNrOfUuids *
                    sizeof(SD_TUUid)));
545.     ptSearchPatternList[0].eUuidType = SD_DET_UUID16;
546.     ptSearchPatternList[0].TUUid.uiUuid16 =
                    SRP_SERIAL_GENERIC_SERIALPORT_UUID ;
547.     SD_ReqServiceSearch (0, m_ConnectionInfo.uiSdcHandle, uiMaxRecords,
                    ucNrOfUuids, ptSearchPatternList);
548. }
549. void CRadioChatClientDlg::OnSdConnectCnfNeg(void **ppMsg)
550. {
551.     SD_TConnectCnfNeg *tConnectCnfNeg = (SD_TConnectCnfNeg *)*ppMsg;
552.     CString str;
553.     str.Format("Could not connect to SD , Error %d",tConnectCnfNeg-
                    >tHdr.iResult);
554.     MessageBox(str);
555. }
556. void CRadioChatClientDlg::OnSdServiceSearchCnf(void **ppMsg)
557. {
558.     SD_TServiceSearchCnf *tServiceSearchCnf = (SD_TServiceSearchCnf
                    *)*ppMsg;
559.     uint16             uiCurrentServiceRecordCount;
560.     uint32             *pulSRHandles;
561.     uint16             *puiAttributeIDList;
562.     uint8              ucNrOfAttr;
563.     CService          service;
564.     uiCurrentServiceRecordCount = tServiceSearchCnf-
                    >uiCurrentServiceRecordCount;
565.     pulSRHandles = (uint32*)OS_Alloc(((uint16)
                    (uiCurrentServiceRecordCount*sizeof(uint32))));
566.     (void*)memcpy(pulSRHandles,
567.         &tServiceSearchCnf->ulServiceRecordHandleList,
568.         (uiCurrentServiceRecordCount*sizeof(uint32)));
569.     m_ConnectionInfo.ulServiceRecordHandle = tServiceSearchCnf-
                    >ulServiceRecordHandleList;
570.     ucNrOfAttr = 1;
571.     puiAttributeIDList =
                    (uint16*)VOS_Alloc((uint16)(ucNrOfAttr*sizeof(uint16)));
572.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
573.     service.m_SDCHandle = m_ConnectionInfo.uiSdcHandle;

```

```

574.     service.m_ServiceRecordHandle = tServiceSearchCnf-
                                         >ulServiceRecordHandleList;
575.     m_ServicesFound.SetAtGrow(m_ServiceCounter,service);
576.     SD_ReqServiceAttribute(1, m_ConnectionInfo.uiSdcHandle,
                             pulSRHandles[0], ucNrOfAttr, puiAttributeIDList);
577.     VOS_Free((void**)&puiAttributeIDList);
578.     VOS_Free((void**)&pulSRHandles);
579. }
580. void CRadioChatClientDlg::OnSdServiceSearchCnfNeg(void **ppMsg)
581. {
582.     SD_TServiceSearchCnfNeg *tConnectCnfNeg = (SD_TServiceSearchCnfNeg
                                                *)*ppMsg;
583.     CString str;
584.     tConnectCnfNeg = tConnectCnfNeg;
585.     str.Format("Service Search Confirm Negative, Error
                 %d",tConnectCnfNeg->tHdr.iResult);
586.     MessageBox(str);
587. }
588. void CRadioChatClientDlg::OnSdServiceAttributeCnf(void **ppMsg)
589. {
590.     SD_TServiceAttributeCnf *tServiceAttributeCnf =
                    (SD_TServiceAttributeCnf *)*ppMsg;
591.     CService service;
592.     switch (tServiceAttributeCnf->tHdr.uiSeqNr)
593.     {
594.     case 1:
595.         ReceiveServiceName(tServiceAttributeCnf);
596.         AskForServiceRecordHandle();
597.         break;
598.     case 2:
599.         ReceiveServiceRecordHandle(tServiceAttributeCnf);
600.         AfxMessageBox("SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle)");
601.         SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle);
602.         break;
603.     default:
604.         break;
605.     }
606. }
607. void CRadioChatClientDlg::OnSdServiceAttributeCnfNeg(void **ppMsg)
608. {
609.     SD_TServiceAttributeCnfNeg *tConnectCnfNeg =
                    (SD_TServiceAttributeCnfNeg *)*ppMsg;
610.     CString str;
611.     tConnectCnfNeg = tConnectCnfNeg;
612.     str.Format("Service Attribute Confirm negative, error
                 %d",tConnectCnfNeg->tHdr.iResult);
613.     MessageBox(str);
614. }
615. void CRadioChatClientDlg::OnSdDisconnectCnf(void **ppMsg)
616. {
617.     ppMsg = ppMsg;
618.     Beep (1000,200);
619.     OnSelservices();
620. }
621. void CRadioChatClientDlg::OnDbmRegisterServiceCnf(void **ppMsg)
622. {

```

```

623.     uint16                uiDescriptorUuidValue;
624.     DBM_TDescriptorValue   tDescriptorValue;
625.     DBM_TDescriptorUuid    tDescriptor;
626.     DBM_TRegisterServiceCnf *tRegisterCnf = (DBM_TRegisterServiceCnf
                                                *)*ppMsg;
627.     m_ConnectionInfo.ulDbmHandle = tRegisterCnf->ulDbmHandle;
628.     uiDescriptorUuidValue        = BT_PSM_COM;
629.     tDescriptor.tType            = DBM_DET_UUID16;
630.     tDescriptor.pucDescriptorUuidValue = (uint8*) &uiDescriptorUuidValue;
631.     tDescriptorValue.uiNrOfParams      = 1;
632.     tDescriptorValue.uiSizeOfValueInBytes = 2;
633.     tDescriptorValue.pucValue = (uint8 *) VOS_Alloc( (sizeof(uint16)) );
634.     *tDescriptorValue.pucValue      = DBM_DET_UINT8;
635.     tDescriptorValue.pucValue++;
636.     *tDescriptorValue.pucValue = m_ConnectionInfo.pucAttributeData
[m_ConnectionInfo.uiAttributeListByteCount - 1];
637.     tDescriptorValue.pucValue--;
638.     DBM_ReqAddDescriptor(0,
639.                          m_ConnectionInfo.ulDbmHandle,
640.                          BT_PROTOCOL_DESCRIPTOR_LIST,
641.                          &tDescriptor,
642.                          &tDescriptorValue);
643.     VOS_Free((void**) &tDescriptorValue.pucValue);
644. }
645. void CRadioChatClientDlg::OnDbmRegisterServiceCnfNeg(void **ppMsg)
646. {
647.     ppMsg = ppMsg;
648.     MessageBox(_T("Not possible to Register to DBM"));
649.     SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle);
650. }
651. void CRadioChatClientDlg::OnDbmAddDescriptorCnf(void **ppMsg)
652. {
653.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
654.     tConnectCnfNeg = tConnectCnfNeg;
655.     OnConnect();
656.     Beep (1000,200);
657. }
658. void CRadioChatClientDlg::OnDbmAddDescriptorCnfNeg(void **ppMsg)
659. {
660.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
661.     tConnectCnfNeg = tConnectCnfNeg;
662.     MessageBox(_T("Could not register the service to DBM"));
663. }
664. void CRadioChatClientDlg::OnConnectAcceptInd(void **ppMsg)
665. {
666.     SCM_TConnectAcceptInd *ptConnectAcceptInd;
667.     ptConnectAcceptInd = (SCM_TConnectAcceptInd *) *ppMsg;
668.     SCM_RspConnectAccept( (MSG_TMMsg **)ppMsg,
669.                          SCM_POS_RESULT,
670.                          ptConnectAcceptInd->tAddress,
671.                          SCM_SLAVE);
672.     *ppMsg = NULL;
673. }
674. void CRadioChatClientDlg::OnHciInquiryEvt(void **ppMsg)
675. {
676.     HCI_TInquiryEvt      *ptInquiryEvt;

```



```

677.     CRemoteDevice device;
678.     ptInquiryEvt =(HCI_TInquiryEvt *) *ppMsg;
679.     device.tAddress = ptInquiryEvt->tAddress;
680.     device.tPageScanMode = ptInquiryEvt->tPageScanMode;
681.     device.tPageScanPeriodMode = ptInquiryEvt->tPageScanPeriodMode,
682.     device.tClockOffset = ptInquiryEvt->tClockOffset;
683.     device.tCod = ptInquiryEvt->tCod;
684.     device.tPageScanRepMode = ptInquiryEvt->tPageScanRepMode;
685.     AddDevice(device);
686. }
687. void CRadioChatClientDlg::OnScmPincodeInd(void **ppMsg)
688. {
689.     SCM_TPINcodeInd      *ptPincodeInd;
690.     ptPincodeInd =(SCM_TPINcodeInd *) *ppMsg;
691.     SCM_RspPincode( (MSG_TMsg **)ppMsg,
692.                    SCM_POS_RESULT,
693.                    ptPincodeInd->tAddress,
694.                    _tPincode,
695.                    PINCODE_LENGTH);
696. }
697. void CRadioChatClientDlg::OnScmConnectEvt(void **ppMsg)
698. {
699.     SCM_TConnectEvt *tConnectEvt = (SCM_TConnectEvt *)*ppMsg;
700.     tConnectEvt = tConnectEvt;
701.     m_ConnectionInfo.tAclHandle = tConnectEvt->tHandle;
702.     m_ConnectionInfo.tAddress = tConnectEvt->tAddress;
703. }
704. void CRadioChatClientDlg::OnScmDisconnectEvt(void **ppMsg)
705. {
706.     ppMsg = ppMsg;
707.     m_ConnectionInfo.tAclHandle = 0;
708.     OnCloseapplication();
709. }
710. void CRadioChatClientDlg::OnHciStartCnf(void **ppMsg)
711. {
712.     HCI_TStartCnf *ptStartCnf = (HCI_TStartCnf *)*ppMsg;
713.     ptStartCnf = ptStartCnf;
714.     HCI_ReqConfigurePort(0,PORTSETTINGS);
715. }
716. void CRadioChatClientDlg::OnComVersionCnf(void **ppMsg)
717. {
718.     CAboutDlg      Abodlg;
719.     COM_TVersionCnf* ptVersionCnf;
720.     char* cpVerStr = NULL;
721.     int8 iCharCount = 9;
722.     char cpStr[3];
723.     ptVersionCnf = (COM_TVersionCnf *) *ppMsg;
724.     cpVerStr = &ptVersionCnf->cVersion;
725.     do
726.     {
727.         iCharCount++;
728.         cpStr[iCharCount-10] = cpVerStr[iCharCount];
729.     }while(iCharCount <= 11);
730.     cpStr[3] = ((char)0);
731.     Abodlg.DoModal();
732. }

```

```

733. void CRadioChatClientDlg::AskForServiceName()
734. {
735.     uint16                *puiAttributeIDList;
736.     uint8                 ucNrOfAttr;
737.     ucNrOfAttr = 1;
738.     puiAttributeIDList = (uint16*)VOS_Alloc((uint16)
                                           (ucNrOfAttr*sizeof(uint16)));
739.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
740.     SD_ReqServiceAttribute(0, m_ConnectionInfo.uiSdcHandle,
                             m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr,
                             puiAttributeIDList);
741.     VOS_Free((void*)&puiAttributeIDList);
742. }
743. void CRadioChatClientDlg::ReceiveServiceName(SD_TServiceAttributeCnf
        *tServiceAttributeCnf)
744. {
745.     CService service;
746.     service = m_ServicesFound.GetAt(m_ServiceCounter);
747.     service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
748.     service.m_AttributeListByteCount = tServiceAttributeCnf-
        >uiAttributeListByteCount;
749.     VOS_Alloc(service.m_AttributeListByteCount);
750.     (void*)memcpy(service.m_pAttributeData,
751.                   &tServiceAttributeCnf->ucAttributeData,
752.                   service.m_AttributeListByteCount);
753.     (void*)memcpy(service.m_pServiceName,
754.                   &service.m_pAttributeData[7],
755.                   service.m_pAttributeData[6]);
756.     service.m_pServiceName[service.m_pAttributeData[6]] = NULL;
757.     m_ServicesFound.SetAt(m_ServiceCounter, service);
758.     m_ServiceCounter++;
759.     m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf-
        >uiAttributeListByteCount;
760.     VOS_Alloc(service.m_AttributeListByteCount);
761.     (void*)memcpy(m_ConnectionInfo.pucAttributeData,
762.                   &tServiceAttributeCnf->ucAttributeData,
763.                   m_ConnectionInfo.uiAttributeListByteCount);
764.     VOS_Alloc(service.m_pAttributeData[6] + 1); /* +1 for the NULL char */
765.     (void*)memcpy(m_ConnectionInfo.pcServiceName,
766.                   &service.m_pAttributeData[7],
767.                   service.m_pAttributeData[6]);
768.     m_ConnectionInfo.pcServiceName[service.m_pAttributeData[6]] = NULL;
769.     ShowAllServicesFound();
770. }
771. void CRadioChatClientDlg::AskForServiceRecordHandle()
772. {
773.     uint16                *puiAttributeIDList;
774.     uint8                 ucNrOfAttr;
775.     ucNrOfAttr = 2;
776.     puiAttributeIDList = (uint16*)VOS_Alloc((uint16)
                                           (ucNrOfAttr*sizeof(uint16)));
777.     puiAttributeIDList[0] = BT_SERVICE_RECORD_HANDLE;
778.     puiAttributeIDList[1] = BT_PROTOCOL_DESCRIPTOR_LIST;
779.     SD_ReqServiceAttribute(2, m_ConnectionInfo.uiSdcHandle,
                             m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr,
                             puiAttributeIDList);

```

```

780.   VOS_Free((void**)&uiAttributeIDList);
781. }
782. void CRadioChatClientDlg::ReceiveServiceRecordHandle(SD_TServiceAttributeCnf
    *tServiceAttributeCnf)
783. {
784.   CService service;
785.   service = m_ServicesFound.GetAt(m_ServiceCounter-1);
786.   service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
787.   service.m_AttributeListByteCount = tServiceAttributeCnf-
    >uiAttributeListByteCount;
788. VOS_Alloc(service.m_AttributeListByteCount);
789.   (void*)memcpy(service.m_pAttributeData,
790.                  &tServiceAttributeCnf->ucAttributeData,
791.                  service.m_AttributeListByteCount);
792.   m_ServicesFound.SetAt(m_ServiceCounter-1,service);
793.   m_ServiceCounter++;
794.   m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf-
    >uiAttributeListByteCount;
795.   (void*)memcpy(m_ConnectionInfo.pucAttributeData,
796.                  &tServiceAttributeCnf->ucAttributeData,
797.                  m_ConnectionInfo.uiAttributeListByteCount);
798. }
799. void CRadioChatClientDlg::OnCloseapplication()
800. {
801.   SCM_ReqDeRegister(1,SCM_SECURITY_HANDLER);
802. }
803. void CRadioChatClientDlg::OnScmDeRegisterCnf(void **ppMsg)
804. {
805.   SCM_TDeRegisterCnf *ptDeRegisterCnf = (SCM_TDeRegisterCnf *) *ppMsg;
806.   switch (ptDeRegisterCnf->tHdr.uiSeqNr)
807.   {
808.   case 1:
809.     SCM_ReqDeRegister(2,SCM_MONITOR_GROUP);
810.     break;
811.   case 2:
812.     if (m_ConnectionInfo.ulDbmHandle > 0)
813.     {
814.       DBM_ReqUnRegisterService(3,m_ConnectionInfo.ulDbmHandle);
815.     }
816.     else
817.     {
818.       if (m_ConnectionInfo.tAclHandle>0)
819.       {
820.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
821.       }
822.       else
823.       {
824.         DestroyWindow();
825.       }
826.     }
827.     break;
828.   default:
829.     break;
830.   }
831. }
832. void CRadioChatClientDlg::OnScmDeRegisterCnfNeg(void **ppMsg)

```

```

833. {
834.     ppMsg = ppMsg;
835.     MessageBox(_T("Could not unregister from SCM"));
836.     DestroyWindow();
837. }
838. void CRadioChatClientDlg::OnDbmUnRegisterServiceCnf(void **ppMsg)
839. {
840.     ppMsg = ppMsg;
841.     if (m_ConnectionInfo.tAclHandle>0)
842.     {
843.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
844.     }
845.     else
846.     {
847.         DestroyWindow();
848.     }
849. }
850. void CRadioChatClientDlg::OnDbmUnRegisterServiceCnfNeg(void **ppMsg)
851. {
852.     ppMsg = ppMsg;
853.     MessageBox(_T("Not possible to UnRegister from DBM"));
854.     DestroyWindow();
855. }
856. void CRadioChatClientDlg::OnScmDisconnectCnf(void **ppMsg)
857. {
858.     ppMsg = ppMsg;
859.     m_ConnectionInfo.tAclHandle = 0;
860.     DestroyWindow();
861. }
862. void CRadioChatClientDlg::OnScmDisconnectCnfNeg(void **ppMsg)
863. {
864.     ppMsg = ppMsg;
865.     MessageBox(_T("Could not remove ACL connection"));
866.     DestroyWindow();
867. }
868. BOOL CRadioChatClientDlg::DestroyWindow()
869. {
870.     return CDialog::DestroyWindow();
871. }
872. void CRadioChatClientDlg::ShowAllDevicesFound()
873. {
874.     CRemoteDevice device;
875.     int iFound,i;
876.     iFound = m_DevicesFound.GetSize();
877.     for (i=0; i < iFound; i++)
878.     {
879.         device = m_DevicesFound.GetAt(i);
880.         AfxMessageBox("device1");
881.         hdevicel=m_tree.InsertItem(device.GetAddress(), hPA, TVI_SORT);
882.         OnSelDevice();
883.     }
884. }
885. void CRadioChatClientDlg::AddService(CString sService)
886. {
887.     CService service(sService);
888.     m_ServicesFound.Add(service);

```

```

889. }
890. void CRadioChatClientDlg::AddService(CService service)
891. {
892.     m_ServicesFound.Add(service);
893. }
894. void CRadioChatClientDlg::ShowAllServicesFound()
895. {
896.     CService service;
897.     int iFound,i;
898.     iFound = m_ServicesFound.GetSize();
899.     for (i=0; i < iFound; i++)
900.     {
901.         service = m_ServicesFound.GetAt(i);
902.         m_tree.InsertItem(service.GetService(),hdevice1,TVI_LAST);
903.     }
904. }
905. void CRadioChatClientDlg::AddDevice(CRemoteDevice device)
906. {
907.     m_DevicesFound.Add(device);
908. }
909. void CRadioChatClientDlg::OnInquiry()
910. {
911.     HCI_TLap    tLap = {0x9E,0x8B,0x33};
912.     HCI_TInquiryLength  tInquiryLength = 2;
913.     HCI_TNrOfResponses  tNrOfResponses = 0;
914.     HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses);
915. }
916. void CRadioChatClientDlg::OnSelDevice()
917. {
918.     CRemoteDevice device;
919.     device = m_DevicesFound.GetAt(0);
920.     m_ConnectionInfo.tAddress = device.tAddress;
921.     SCM_ReqConnect(0,
922.                   device.tAddress,
923.                   SCM_DM1,
924.                   SCM_R1,
925.                   SCM_MANDATORY_PAGE_SCAN_MODE,
926.                   0,
927.                   SCM_NOT_ACCEPT_ROLE_SWITCH);
928. }
929.
930. void CRadioChatClientDlg::OnSelservices()
931. {
932.     CService service;
933.     service = m_ServicesFound.GetAt(0);
934.     m_ConnectionInfo.ulServiceRecordHandle = service.
                                                m_ServiceRecordHandle;
935.     DBM_ReqRegisterService(0, DBM_StackDB);
936. }
937. void CRadioChatClientDlg::OnGetservices()
938. {
939.     m_ServiceCounter = 0;
940.     SD_ReqConnect(0,SD_DEFAULT_MFS,m_ConnectionInfo.tAclHandle);
941. }
942. void CRadioChatClientDlg::OnConnect()
943. {

```

```

944.     COM_ReqConnect(0,
945.                     (uint16)m_ConnectionInfo.ulDbmHandle,
946.                     m_ConnectionInfo.tAclHandle,
947.                     m_ConnectionInfo.uiMaxFrameSize);
948. }
949. void CRadioChatClientDlg::OnComConnectCnf(void **ppMsg)
950. {
951.     COM_TConnectCnf *ptConnectCnf = (COM_TConnectCnf *) *ppMsg;
952.     m_ConnectionInfo.uiRFCOMMHandle = ptConnectCnf->uiHandle;
953.     MessageBox(_T(" RFCOMM connection"));
954.     Beep (1000,200);
955.     Sleep(100);
956.     Beep (1000,200);
957. }
958. void CRadioChatClientDlg::OnComConnectCnfNeg(void **ppMsg)
959. {
960.     COM_TConnectCnfNeg *ptConnectCnfNeg = (COM_TConnectCnfNeg *) *ppMsg;
961.     ptConnectCnfNeg = ptConnectCnfNeg;
962.     m_ConnectionInfo.uiRFCOMMHandle = 0;
963.     MessageBox(_T("Could not create a RFCOMM connection"));
964. }
965. void CRadioChatClientDlg::OnComDataInd(void **ppMsg)
966. {
967.     COM_TDataInd *tDataInd = (COM_TDataInd *)*ppMsg;
968.     uint8 *pucData;
969.     CHAR sData[80];
970.     uint16 uiLength;
971.     uint16 uiHandle;
972.     int i;
973.     pucData = COM_DataExtract((MSG_TDataMsg *)*ppMsg,
974.                               &uiLength,
975.                               &uiHandle);
976.     COM_RspData(tDataInd->tHdr.ucSeqNr, COM_POS_RESULT, uiHandle);
977.     for (i=0; i < uiLength; i++)
978.     {
979.         sData[i] = pucData[i] ;
980.     }
981.     m_ChatArea.ReceiveMessageFromClient((CString)sData);
982. }
983. void CRadioChatClientDlg::OnComDataCnf(void **ppMsg)
984. {
985.     ppMsg = ppMsg;
986. }
987. void CRadioChatClientDlg::OnComDataCnfNeg(void **ppMsg)
988. {
989.     ppMsg = ppMsg;
990.     MessageBox(_T("Could not send data on RFCOMM channel"));
991. }
992. void CRadioChatClientDlg::OnComDisconnectEvt(void **ppMsg)
993. {
994.     COM_TDisconnectEvt *ptDisconnectEvt = (COM_TDisconnectEvt *)*ppMsg;
995.     ptDisconnectEvt = ptDisconnectEvt;
996.     m_ConnectionInfo.uiRFCOMMHandle = 0;
997.     EndModalLoop(0);
998. }
999. void CRadioChatClientDlg::OnComDisconnectCnf(void **ppMsg)

```

```

1000. {
1001.     COM_TDisconnectCnf *ptDisconnectCnf = (COM_TDisconnectCnf *)*ppMsg;
1002.     ptDisconnectCnf = ptDisconnectCnf;
1003.     m_ConnectionInfo.uiRFCCommHandle = 0;
1004.     EndModalLoop(0);
1005. }
1006. void CRadioChatClientDlg::OnComDisconnectCnfNeg(void **ppMsg)
1007. {
1008.     COM_TDisconnectCnfNeg *ptDisconnectCnfNeg = (COM_TDisconnectCnfNeg
1009.                                                    *)*ppMsg;
1009.     ptDisconnectCnfNeg = ptDisconnectCnfNeg;
1010.     MessageBox(_T("Could not Disconnect the RFCOMM connection"));
1011. }
1012. void CRadioChatClientDlg::HandleReturn()
1013. {
1014.     CHAR sChatStr[80];
1015.     uint8 *pucData;
1016.     uint16 iCount,i;
1017.     iCount = (uint16) m_InputChat.GetWindowText(sChatStr,80);
1018.     if (iCount > 0)
1019.     {
1020.         pucData = COM_DataAlloc((uint16)(iCount + 1));
1021.         for (i=0; i < iCount; i++)
1022.         {
1023.             pucData[i] = sChatStr[i];
1024.         }
1025.         pucData[i] = 0;
1026.         COM_DataSend(0,pucData,m_ConnectionInfo.uiRFCCommHandle,
1027.                     (uint16)(iCount + 1));
1027.         m_ChatArea.SendMessageToServer((CString)sChatStr);
1028.         m_InputChat.SetWindowText(_T(""));
1029.     }
1030. }
1031. BOOL CRadioChatClientDlg::PreTranslateMessage(MSG* pMsg)
1032. {
1033.     if (pMsg->message == WM_KEYDOWN)
1034.     {
1035.         if (pMsg->wParam == VK_RETURN)
1036.         {
1037.             HandleReturn();
1038.             return FALSE;
1039.         }
1040.     }
1041.     return CDialog::PreTranslateMessage(pMsg);
1042. }

```

Code Description

The code of file transfer application is reused here to the maximum possible extent. The differences are explained below.

- ◆ Lines 1–964: Explained in RadioFileClientDlg.cpp file.
- ◆ Lines 965–982: After the Bluetooth module fires the COM_DATA_IND event, the command to read the incoming data from the server is issued. The response is sent to the server to indicate that the data has been received. The data that has been read is added to chat area in the dialog box.
- ◆ Lines 983–1011: Explained in RadioFileClientDlg.cpp file.

- ◆ Lines 1012–1030: The data entered in the chat input will be retrieved and sent to the remote Bluetooth peer device. The data will be displayed in the chat area list box.
- ◆ Lines 1031–1042: As soon as the enter key is pressed, the function `HandleReturn` is invoked to send the data to the remote device.

Code Output

When the application is built in the VC++ environment, the screen in Figure 9-9 is displayed. The left side of the screen shows a tree structure. This tree structure contains one root node named `RemoteRadios`. When the client device gets connected to the server Bluetooth device, its address is added to the root as a child node. The services supported by the server will be added to the BLUETOOTH device address node. The right side of the screen contains one edit box and list boxes to display the exchange of messages. The user has to press Enter in the edit box to send the message to the server.



Figure 9-9: Output of the Client Module

Server Module

Listings 9-23 and 9-24 give the source code of `RadioChatServerDlg.h` and `RadioChatServerDlg.cpp`, respectively.

Listing 9-23: Source Code for `RadioChatServerDlg.h`

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. // RadioChatServerDlg.h : header file
2.
3. #include "Events.h"
4. #include "ConnectionInfo.h"
5. #include "Remotedevice.h"
6. #include "RS232.h"
7. #include "service.h"
8. #include <exp\sd.h>
9. #include <exp\BT_COMServer.h>
10. #include <afxtempl.h>
```



```

11. CRadioChatServerDlg dialog
12. #define WM_BLUETOOTH_EVENT (WM_USER + 100)
13. #define ON_BLUETOOTH_EVENT(uiBtEventID, memberFxn) \
14.     { WM_BLUETOOTH_EVENT, uiBtEventID, 0, 0, 1, \
15.     (AFX_PMSG) (AFX_PMSGW) (void (AFX_MSG_CALL CWnd::*) (void **)) &memberFxn },
16. #define SEND_BT_EVENT(uiBtEventID, pMsg) \
17.     SendMessage( (HWND) this->
m_hWnd, WM_BLUETOOTH_EVENT, (WPARAM) uiBtEventID, (LPARAM) &pMsg)
18.
19. class CRadioChatServerDlg : public CDialog
20. {
21.
22. public:
23.     CRadioChatServerDlg(CWnd* pParent = NULL);
24.     ~CRadioChatServerDlg();
25.         CConnectionInfo m_ConnectionInfo;
26.
27. private:
28.     void InitSecurityClient();
29.     BOOL OnBluetoothEvent(UINT message, WPARAM wParam, LPARAM lParam);
30.
31.     //{AFX_DATA(CRadioFileServerDlg)
32.     enum { IDD = IDD_RADIOCHAT_DIALOG };
33.     CListBox      m_ChatArea;
34.     CEdit m_InputChat;
35.     CTreeCtrl m_tree;
36.     //}}AFX_DATA
37.
38.     // ClassWizard generated virtual function overrides
39.     //{AFX_VIRTUAL(CRadioFileServerDlg)
40.
41. public:
42.     virtual BOOL DestroyWindow();
43. protected:
44.     virtual void DoDataExchange(CDataExchange* pDX);
45.         virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM
lParam);
46.     //}}AFX_VIRTUAL
47.
48. public:
49.     void AddDevice(CString sAddress, CString sName);
50.     void AddDevice(CDevice device);
51.     void ShowAllDevicesFound();
52.     void AddService(CString sService);
53.     void AddService(CService service);
54.     void ShowAllServicesFound();
55.     void AskForServiceName();
56. void ReceiveServiceName(SD_TServiceAttributeCnf *tServiceAttributeCnf);
57.     void AskForServiceRecordHandle();
58. void ReceiveServiceRecordHandle(SD_TServiceAttributeCnf
*tServiceAttributeCnf);
59. protected:
60.     HICON m_hIcon;
61.     int index;
62.     CServerEvents *m_pServerEvents;
63.     CRS232 *m_pSerialPort;

```

```

64. CArray <CDevice,CDevice> m_DevicesFound;
65. CArray <CService,CService> m_ServicesFound;
66. int m_RemoteNameCounter;
67. int m_ServiceCounter;
68. uint8 m_RFServerChannel;
69. uint16 m_RFCommHandle;
70. // Generated message map functions
71. //{AFX_MSG(CRadioFileServerDlg)
72. virtual BOOL OnInitDialog();
73. afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
74. afx_msg void OnPaint();
75. afx_msg HCURSOR OnQueryDragIcon();
76. afx_msg void OnInquiry();
77. afx_msg void OnSelDevice();
78. afx_msg void OnConnect();
79. afx_msg void OnGetServices();
80. afx_msg void OnSelServices();
81. afx_msg void OnDestroy();
82. afx_msg void OnSerialport();
83. afx_msg void OnCloseapplication();
84. afx_msg void OnHCISerial();
85. afx_msg void OnHCIUsb();
86. afx_msg void OnComStartCnf(void **ppMsg);
87. afx_msg void OnComStartCnfNeg(void **ppMsg);
88. afx_msg void OnComFillPdlCnf(void **ppMsg);
89. afx_msg void OnComFillPdlCnfNeg(void **ppMsg);
90. afx_msg void OnComRegisterCnf(void **ppMsg);
91. afx_msg void OnComRegisterCnfNeg(void **ppMsg);
92. afx_msg void OnComConnectInd(void **ppMsg);
93. afx_msg void OnComDataInd(void **ppMsg);
94. afx_msg void OnComDataCnf(void **ppMsg);
95. afx_msg void OnComDataCnfNeg(void **ppMsg);
96.
97. afx_msg void OnComVersionCnf(void **ppMsg);
98. afx_msg void OnScmRegisterCnf(void **ppMsg);
99. afx_msg void OnScmRegisterCnfNeg(void **ppMsg);
100. afx_msg void OnConnectAcceptInd(void **ppMsg);
101. afx_msg void OnScmPincodeInd(void **ppMsg);
102. afx_msg void OnScmConnectCnf(void **ppMsg);
103. afx_msg void OnScmConnectCnfNeg(void **ppMsg);
104. afx_msg void OnScmConnectEvt(void **ppMsg);
105. afx_msg void OnScmDisconnectEvt(void **ppMsg);
106. afx_msg void OnScmDisconnectCnf(void **ppMsg);
107. afx_msg void OnScmDisconnectCnfNeg(void **ppMsg);
108. afx_msg void OnScmDeRegisterCnf(void **ppMsg);
109. afx_msg void OnScmDeRegisterCnfNeg(void **ppMsg);
110. afx_msg void OnSdStartCnf(void **ppMsg);
111. afx_msg void OnSdConnectCnf(void **ppMsg);
112. afx_msg void OnSdConnectCnfNeg(void **ppMsg);
113. afx_msg void OnSdServiceSearchCnf(void **ppMsg);
114. afx_msg void OnSdServiceSearchCnfNeg(void **ppMsg);
115. afx_msg void OnSdServiceAttributeCnf(void **ppMsg);
116. afx_msg void OnSdServiceAttributeCnfNeg(void **ppMsg);
117. afx_msg void OnSdDisconnectCnf(void **ppMsg);
118. afx_msg void OnDbmRegisterServiceCnf(void **ppMsg);
119. afx_msg void OnDbmRegisterServiceCnfNeg(void **ppMsg);

```

```

120.  afx_msg void OnDbmUnRegisterServiceCnf(void **ppMsg);
121.  afx_msg void OnDbmUnRegisterServiceCnfNeg(void **ppMsg);
122.  afx_msg void OnDbmAddDescriptorCnf(void **ppMsg);
123.  afx_msg void OnDbmAddDescriptorCnfNeg(void **ppMsg);
124.  afx_msg void OnHciConfigurePortConfirm(void **ppMsg);
125.  afx_msg void OnHciConfigurePortConfirmNegative(void **ppMsg);
126.  afx_msg void OnHciInquiryCnf(void **ppMsg);
127.  afx_msg void OnHciInquiryEvt(void **ppMsg);
128.  afx_msg void OnHciLocalAddressCnf(void **ppMsg);
129.  afx_msg void OnHciLocalAddressCnfNeg(void **ppMsg);
130.  afx_msg void OnHciRemoteNameCnf(void **ppMsg);
131.  afx_msg void OnHciRemoteNameCnfNeg(void **ppMsg);
132.  afx_msg void OnHciStartCnf(void **ppMsg);
133.  afx_msg void OnHciWriteScanEnableCnf(void **ppMsg);
134.  afx_msg void OnHciWriteScanEnableCnfNeg(void **ppMsg);
135.
136.
137.  afx_msg void OnHciWriteAuthenticationModeCnf(void **ppMsg);
138.  afx_msg void OnHciWriteAuthenticationModeCnfNeg(void **ppMsg);
139.  afx_msg void OnHciWriteEncryptionModeCnf(void **ppMsg);
140.  afx_msg void OnHciWriteEncryptionModeCnfNeg(void **ppMsg);
141.  afx_msg void OnHciWriteCodCnf(void **ppMsg);
142.  afx_msg void OnHciWriteCodCnfNeg(void **ppMsg);
143.  afx_msg void OnHciWriteNameCnf(void **ppMsg);
144.  afx_msg void OnHciWriteNameCnfNeg(void **ppMsg);
145.  afx_msg void OnHciWriteConnectTimeoutCnf(void **ppMsg);
146.  afx_msg void OnHciWriteConnectTimeoutCnfNeg(void **ppMsg);
147.  afx_msg void OnHciWritePageTimeoutCnf(void **ppMsg);
148.  afx_msg void OnHciWritePageTimeoutCnfNeg(void **ppMsg);
149.  afx_msg void OnSilSetDeviceCnf(void **ppMsg);
150.  afx_msg void OnSilSetDeviceCnfNeg(void **ppMsg);
151.  afx_msg void OnSilReqDeviceCnf(void **ppMsg);
152.  afx_msg void OnSilReqDeviceCnfNeg(void **ppMsg);
153.  afx_msg void OnComConnectCnf(void **ppMsg);
154.  afx_msg void OnComConnectCnfNeg(void **ppMsg);
155.  afx_msg void OnSdsStartCnf(void **ppMsg);
156.  afx_msg void OnButton2();
157.  //}}AFX_MSG
158.  DECLARE_MESSAGE_MAP()
159. };
160.

```

Code Description

In this file, all variables, constants, methods, and classes necessary to implement CRadioChatServerDlg class were declared.

Listing 9-24: Source code for RadioChatServerDlg.cpp

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. // RadioChatServerDlg.cpp : implementation file
2. #include "stdafx.h"
3. #include "RadioChat.h"
4. #include "RadioChatServerDlg.h"
5. #include "Events.h"

```

```
6. #include <process.h>
7. #include "windows.h"
8. #include <exp/msg.h>
9. #include <exp/hci.h>
10. #include <exp/hci_drv.h>
11. #include <exp/scm.h>
12. #include <exp/com.h>
13. #include <exp/dbm.h>
14. #include <exp/sd.h>
15. #include <exp/sds.h>
16. #include <exp/vos2com.h>
17. #include <exp/sil.h>
18. #include <exp/Bstr.h>
19.
20. #ifdef _DEBUG
21. #define new DEBUG_NEW
22. #undef THIS_FILE
23. static char THIS_FILE[] = __FILE__;
24. #endif
25.
26. HTREEITEM hPA,hdevice1;
27. union MessageMapFunctions
28. {
29.     AFX_PMSG pfn;
30.     void      (AFX_MSG_CALL CWnd::*pfn_btfn)(void **);
31. };
32. #define PINCODE_LENGTH ((SCM_TPIncodeLength) 4)
33. static const SCM_TPIncode _tPincode =
{'1','2','3','4','0','0','0','0','0','0','0','0','0','0','0','0','0','0'};
34. #define PORTSETTINGS (uint8 *)("COM1:Baud=57600 parity=N data=8 stop=1")
35. #define InterSelSerial ((uint8) 0)
36. #define InterSelUSB      ((uint8) 1)
37. #define SRP_SERIAL_GENERIC_SERIALPORT_UUID(uint16) 0x1101
38. static const HCI_TCod _tCod={0x20,0x04,0x04};
39.
40. class CAboutDlg : public CDialog
41. {
42. public:
43.     CAboutDlg();
44. // Dialog Data
45.     //{AFX_DATA(CAboutDlg)
46.     enum { IDD = IDD_ABOUTBOX };
47.     //}AFX_DATA
48. // ClassWizard generated virtual function overrides
49.     //{AFX_VIRTUAL(CAboutDlg)
50. protected:
51.     virtual void DoDataExchange(CDataExchange* pDX);
52.     //}AFX_VIRTUAL
53. // Implementation
54. protected:
55.     //{AFX_MSG(CAboutDlg)
56.     //}AFX_MSG
57.     DECLARE_MESSAGE_MAP()
58. };
59. CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
60. {
```

```

61.  //{AFX_DATA_INIT(CAboutDlg)
62.  //}}AFX_DATA_INIT
63. }
64. void CAboutDlg::DoDataExchange(CDataExchange* pDX)
65. {
66.  CDialog::DoDataExchange(pDX);
67.  //{AFX_DATA_MAP(CAboutDlg)
68.  //}}AFX_DATA_MAP
69. }
70. BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
71.  //{AFX_MSG_MAP(CAboutDlg)
72.  // No message handlers
73.  //}}AFX_MSG_MAP
74. END_MESSAGE_MAP()
75.
76. CRadioChatServerDlg::CRadioChatServerDlg(CWnd* pParent /*=NULL*/)
77. : CDialog(CRadioChatServerDlg::IDD, pParent)
78. {
79.  //{AFX_DATA_INIT(CRadioChatServerDlg)
80.  // NOTE: the ClassWizard will add member initialization here
81.  //}}AFX_DATA_INIT
82.
83.  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
84.  m_pServerEvents = new Events();
85. }
86. CRadioChatServerDlg::~CRadioChatServerDlg()
87. {
88.  m_DevicesFound.RemoveAll();
89.  m_ServicesFound.RemoveAll();
90.
91.  delete m_pServerEvents;
92. }
93. void CRadioChatServerDlg::DoDataExchange(CDataExchange* pDX)
94. {
95.  CDialog::DoDataExchange(pDX);
96.  //{AFX_DATA_MAP(CRadioChatServerDlg)
97.  DDX_Control(pDX, IDC_EDIT1, m_InputChat);
98.  DDX_Control(pDX, IDC_LIST1, m_ChatArea);
99.  DDX_Control(pDX, IDC_TREE1, m_tree);
100.  //}}AFX_DATA_MAP
101. }
102. BEGIN_MESSAGE_MAP(CRadioChatServerDlg, CDialog)
103.  //{AFX_MSG_MAP(CRadioChatServerDlg)
104.  ON_WM_SYSCOMMAND()
105.  ON_WM_PAINT()
106.  ON_WM_QUERYDRAGICON()
107.
108.  ON_BLUETOOTH_EVENT(COM_DATA_IND, OnComDataInd )
109.  ON_BLUETOOTH_EVENT(COM_REGISTER_CNF, OnComRegisterCnf )
110.  ON_BLUETOOTH_EVENT(COM_REGISTER_CNF_NEG, OnComRegisterCnfNeg )
111.  ON_BLUETOOTH_EVENT(COM_FILL_PDL_CNF, OnComFillPdlCnf )
112.  ON_BLUETOOTH_EVENT(COM_FILL_PDL_CNF_NEG, OnComFillPdlCnfNeg )
113.  ON_BLUETOOTH_EVENT(COM_CONNECT_IND, OnComConnectInd )
114.  ON_BLUETOOTH_EVENT(COM_START_CNF, OnComStartCnf)
115.  ON_BLUETOOTH_EVENT(COM_START_CNF_NEG, OnComStartCnfNeg)
116.  ON_BLUETOOTH_EVENT(COM_VERSION_CNF, OnComVersionCnf)

```

```

117.    ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF, OnScmRegisterCnf)
118.    ON_BLUETOOTH_EVENT(SCM_REGISTER_CNF_NEG, OnScmRegisterCnfNeg)
119.    ON_BLUETOOTH_EVENT(SCM_CONNECT_ACCEPT_IND, OnConnectAcceptInd)
120.    ON_BLUETOOTH_EVENT(SCM_PINCODE_IND, OnScmPincodeInd)
121.    ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF, OnScmConnectCnf)
122.    ON_BLUETOOTH_EVENT(SCM_CONNECT_CNF_NEG, OnScmConnectCnfNeg)
123.    ON_BLUETOOTH_EVENT(SCM_CONNECT_EVT, OnScmConnectEvt)
124.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_EVT, OnScmDisconnectEvt)
125.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF, OnScmDisconnectCnf)
126.    ON_BLUETOOTH_EVENT(SCM_DISCONNECT_CNF_NEG, OnScmDisconnectCnfNeg)
127.    ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF, OnScmDeRegisterCnf)
128.    ON_BLUETOOTH_EVENT(SCM_DEREGISTER_CNF_NEG, OnScmDeRegisterCnfNeg)
129.    ON_BLUETOOTH_EVENT(SD_START_CNF, OnSdStartCnf)
130.    ON_BLUETOOTH_EVENT(SD_CONNECT_CNF, OnSdConnectCnf)
131.    ON_BLUETOOTH_EVENT(SD_CONNECT_CNF_NEG, OnSdConnectCnfNeg)
132.    ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF, OnSdServiceSearchCnf)
133.    ON_BLUETOOTH_EVENT(SD_SERVICE_SEARCH_CNF_NEG, OnSdServiceSearchCnfNeg)
134.    ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF, OnSdServiceAttributeCnf)
135.    ON_BLUETOOTH_EVENT(SD_SERVICE_ATTRIBUTE_CNF_NEG,
        OnSdServiceAttributeCnfNeg)
136.    ON_BLUETOOTH_EVENT(SD_DISCONNECT_CNF, OnSdDisconnectCnf)
137.    ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF, OnDbmRegisterServiceCnf)
138.    ON_BLUETOOTH_EVENT(DBM_REGISTER_SERVICE_CNF_NEG,
        OnDbmRegisterServiceCnfNeg)
139.    ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF,
        OnDbmUnRegisterServiceCnf)
140.    ON_BLUETOOTH_EVENT(DBM_UNREGISTER_SERVICE_CNF_NEG,
        OnDbmUnRegisterServiceCnfNeg)
141.    ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF, OnDbmAddDescriptorCnf)
142.    ON_BLUETOOTH_EVENT(DBM_ADD_DESCRIPTOR_CNF_NEG,
        OnDbmAddDescriptorCnfNeg)
143.    ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF, OnHciConfigurePortConfirm)
144.    ON_BLUETOOTH_EVENT(HCI_CONFIGURE_PORT_CNF_NEG,
        OnHciConfigurePortConfirmNegative)
145.    ON_BLUETOOTH_EVENT(HCI_INQUIRY_CNF, OnHciInquiryCnf)
146.    ON_BLUETOOTH_EVENT(HCI_INQUIRY_EVT, OnHciInquiryEvt)
147.    ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF, OnHciLocalAddressCnf)
148.    ON_BLUETOOTH_EVENT(HCI_LOCAL_ADDRESS_CNF_NEG, OnHciLocalAddressCnfNeg)
149.    ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF, OnHciRemoteNameCnf)
150.    ON_BLUETOOTH_EVENT(HCI_REMOTE_NAME_CNF_NEG, OnHciRemoteNameCnfNeg)
151.    ON_BLUETOOTH_EVENT(HCI_START_CNF, OnHciStartCnf)
152.    ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF, OnHciWriteScanEnableCnf)
153.    ON_BLUETOOTH_EVENT(HCI_WRITE_SCAN_ENABLE_CNF_NEG,
        OnHciWriteScanEnableCnfNeg)
154.
155.
156.    ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF,
        OnHciWriteAuthenticationModeCnf)
157.    ON_BLUETOOTH_EVENT(HCI_WRITE_AUTHENTICATION_MODE_CNF_NEG,
        OnHciWriteAuthenticationModeCnfNeg)
158.    ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF,
        OnHciWriteEncryptionModeCnf)
159.    ON_BLUETOOTH_EVENT(HCI_WRITE_ENCRYPTION_MODE_CNF_NEG,
        OnHciWriteEncryptionModeCnfNeg)
160.    ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF, OnHciWriteCodCnf)

```

```

161.     ON_BLUETOOTH_EVENT(HCI_WRITE_COD_CNF_NEG, OnHciWriteCodCnfNeg)
162.     ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF, OnHciWriteNameCnf)
163.     ON_BLUETOOTH_EVENT(HCI_WRITE_NAME_CNF_NEG, OnHciWriteNameCnfNeg)
164.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF,
OnHciWriteConnectTimeoutCnf)
165.     ON_BLUETOOTH_EVENT(HCI_WRITE_CONNECT_TIMEOUT_CNF_NEG,
OnHciWriteConnectTimeoutCnfNeg)
166.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF,
OnHciWritePageTimeoutCnf)
167.     ON_BLUETOOTH_EVENT(HCI_WRITE_PAGE_TIMEOUT_CNF_NEG,
OnHciWritePageTimeoutCnfNeg)
168.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF, OnSilSetDeviceCnf)
169.     ON_BLUETOOTH_EVENT(SIL_SET_DEVICE_CNF_NEG, OnSilSetDeviceCnfNeg)
170.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF, OnSilReqDeviceCnf)
171.     ON_BLUETOOTH_EVENT(SIL_REQ_DEVICE_CNF_NEG, OnSilReqDeviceCnfNeg)
172.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF, OnComConnectCnf )
173.     ON_BLUETOOTH_EVENT(COM_CONNECT_CNF_NEG, OnComConnectCnfNeg )
174.     ON_BLUETOOTH_EVENT(SDS_START_CNF, OnSdsStartCnf)
175.     ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
176.     //}}AFX_MSG_MAP
177. END_MESSAGE_MAP()
178.
179. BOOL CRadioChatServerDlg::OnInitDialog()
180. {
181.     CDialog::OnInitDialog();
182.
183.     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
184.     ASSERT(IDM_ABOUTBOX < 0xF000);
185.
186.     CMenu* pSysMenu = GetSystemMenu(FALSE);
187.     if (pSysMenu != NULL)
188.     {
189.         CString strAboutMenu;
190.         strAboutMenu.LoadString(IDS_ABOUTBOX);
191.         if (!strAboutMenu.IsEmpty())
192.         {
193.             pSysMenu->AppendMenu(MF_SEPARATOR);
194.             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
195.         }
196.     }
197.
198.     SetIcon(m_hIcon, TRUE);
199.     SetIcon(m_hIcon, FALSE);
200.     m_pServerEvents->m_pParentDialog = this;
201.
202.     TVINSERTSTRUCT tvInsert;
203.     tvInsert.hParent = NULL;
204.     tvInsert.hInsertAfter = NULL;
205.     tvInsert.item.mask = TVIF_TEXT;
206.     tvInsert.item.pszText = _T("RemoteRadios");
207.
208.     hPA = m_tree.InsertItem(&tvInsert);
209.     index=0;
210.     m_InputChat.SetWindowText("Type And Press Enter To Send Your Message");
211.     InitSecurityClient();
212.     return TRUE;

```

```

213. }
214.
215. LRESULT CRadioChatServerDlg::WindowProc(UINT message, WPARAM wParam, LPARAM
LPARAM)
216. {
217.
218.     MSG_TMsg **ptMsg;
219.
220.     if (message == WM_BLUETOOTH_EVENT)
221.     {
222.
223.         OnBluetoothEvent( message, wParam, lParam);
224.         ptMsg = (MSG_TMsg**)lParam;
225.         if (*ptMsg != NULL)
226.             VOS_Free((void **)lParam);
227.     }
228.
229.     return CDialog::WindowProc(message, wParam, lParam);
230. }
231. BOOL CRadioChatServerDlg::OnBluetoothEvent(UINT message, WPARAM wParam,
LPARAM lParam)
232. {
233.     const AFX_MSGMAP* pMessageMap;
234.     const AFX_MSGMAP_ENTRY* lpEntry;
235.
236. #ifdef _AFXDLL
237.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
238.         pMessageMap = (*pMessageMap->pfnGetBaseMap)())
239. #else
240.     for (pMessageMap = GetMessageMap(); pMessageMap != NULL;
241.         pMessageMap = pMessageMap->pBaseMap)
242. #endif
243.     {
244.
245. #ifdef _AFXDLL
246.         ASSERT(pMessageMap != (*pMessageMap->pfnGetBaseMap)());
247. #else
248.         ASSERT(pMessageMap != pMessageMap->pBaseMap);
249. #endif
250.
251.
252.         lpEntry = (AFX_MSGMAP_ENTRY*)&pMessageMap->lpEntries[0];
253.         while (lpEntry->nSig != AfxSig_end)
254.         {
255.             if ((lpEntry->nMessage == message) && (lpEntry->nCode == wParam))
256.             {
257.
258.                 union MessageMapFunctions mmf;
259.                 mmf.pfn = lpEntry->pfn;
260.
261.                 (((CWnd *)this)->*mmf.pfn_btfn)((void **)lParam);
262.
263.                 return TRUE;
264.             }
265.             lpEntry++;
266.         }

```



```

267.         return FALSE;
268.     }
269.     return FALSE;
270. }
271.
272. void CRadioChatServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
273. {
274.     if ((nID & 0xFFFF) == IDM_ABOUTBOX)
275.     {
276.         CAboutDlg dlgAbout;
277.         dlgAbout.DoModal();
278.     }
279.     else
280.     {
281.         CDialog::OnSysCommand(nID, lParam);
282.     }
283. }
284.
285. void CRadioChatServerDlg::OnPaint()
286. {
287.     if (IsIconic())
288.     {
289.         CPaintDC dc(this);
290.
291.         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
292.
293.         int cxIcon = GetSystemMetrics(SM_CXICON);
294.         int cyIcon = GetSystemMetrics(SM_CYICON);
295.         CRect rect;
296.         GetClientRect(&rect);
297.         int x = (rect.Width() - cxIcon + 1) / 2;
298.         int y = (rect.Height() - cyIcon + 1) / 2;
299.
300.         dc.DrawIcon(x, y, m_hIcon);
301.     }
302.     else
303.     {
304.         CDialog::OnPaint();
305.     }
306. }
307.
308. HCURSOR CRadioChatServerDlg::OnQueryDragIcon()
309. {
310.     return (HCURSOR) m_hIcon;
311. }
312.
313. void CRadioChatServerDlg::InitSecurityClient()
314. {
315.
316.     SIL_SetDevice(0, SIL_SERIAL);
317.
318. }
319. void CRadioChatServerDlg::OnSilSetDeviceCnf(void **ppMsg)
320. {
321.     ppMsg = ppMsg;
322.

```

```

323. HCI_ReqConfigurePort(0,PORTSETTINGS);
324. }
325.
326.
327. void CRadioChatServerDlg::OnSilSetDeviceCnfNeg(void **ppMsg)
328. {
329.     SIL_TSetDevice*      ptSetDevice;
330.
331.     ptSetDevice = (SIL_TSetDevice*)*ppMsg;
332.     if(ptSetDevice->tHdr.iResult == SIL_ERR_DEVICE)
333.         SIL_ReqDevice(0);
334. }
335.
336. void CRadioChatServerDlg::OnSilReqDeviceCnf(void **ppMsg)
337. {
338.     SIL_TReqDevice*      ptReq;
339.     ptReq = (SIL_TReqDevice*) *ppMsg;
340.
341.     if(ptReq->uiDevice == SIL_SERIAL)
342.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI Interface is
SERIAL"));
343.     if(ptReq->uiDevice == SIL_USB)
344.         MessageBox(_T("NOT Possible To Change Interface\nCurrent HCI Interface is
USB"));
345.
346. }
347.
348. void CRadioChatServerDlg::OnSilReqDeviceCnfNeg(void **ppMsg)
349. {
350.     ppMsg = ppMsg;
351.
352.     MessageBox(_T("Device Request FAILED!"));
353. }
354.
355. void CRadioChatServerDlg::OnHciConfigurePortConfirm(void **ppMsg)
356. {
357.     HCI_TConfigurePortCnf *tConfigurePort = (HCI_TConfigurePortCnf *)*ppMsg;
358.
359.     tConfigurePort = tConfigurePort;
360.
361.     COM_ReqStart(0);
362. }
363. void CRadioChatServerDlg::OnHciConfigurePortConfirmNegative(void **ppMsg)
364. {
365.     HCI_TConfigurePortCnfNeg *tConfigurePort = (HCI_TConfigurePortCnfNeg
*)*ppMsg;
366.
367.     tConfigurePort = tConfigurePort;
368.
369.     MessageBox(_T("Could not open port"));
370. }
371. void CRadioChatServerDlg::OnComStartCnf(void **ppMsg)
372. {
373.     COM_TStartCnf *tStartCnf = (COM_TStartCnf *)*ppMsg;
374.
375.     tStartCnf = tStartCnf;

```

```

376.
377.     HCI_ReqLocalAddress(0);
378.
379. }
380. void CRadioChatServerDlg::OnComStartCnfNeg(void **ppMsg)
381. {
382.     COM_TStartCnfNeg *tStartCnfNeg = (COM_TStartCnfNeg *)*ppMsg;
383.
384.     tStartCnfNeg = tStartCnfNeg;
385.
386.     MessageBox(_T("Could not start RFCOMM"));
387. }
388. void CRadioChatServerDlg::OnHciLocalAddressCnf(void **ppMsg)
389. {
390.     HCI_TLocalAddressCnf *tLocalAddress =
391.     (HCI_TLocalAddressCnf *)*ppMsg;
392.     char lpStr[59];
393.     wsprintf(&lpStr[0], "BD_ADDRESS: 0x%02X%02X%02X%02X%02X%02X",
394.         tLocalAddress->tAddress.ucByte0,
395.         tLocalAddress->tAddress.ucByte1,
396.         tLocalAddress->tAddress.ucByte2,
397.         tLocalAddress->tAddress.ucByte3,
398.         tLocalAddress->tAddress.ucByte4,
399.         tLocalAddress->tAddress.ucByte5);
400.
401.     SetWindowText(_T(lpStr));
402.     SD_ReqStart(0);
403. }
404. void CRadioChatServerDlg::OnHciLocalAddressCnfNeg(void **ppMsg)
405. {
406.     ppMsg = ppMsg;
407.     SetWindowText(_T("DEVICE NOT FOUND"));
408.     SD_ReqStart(0);
409. }
410. void CRadioChatServerDlg::OnSdStartCnf(void **ppMsg)
411. {
412.     ppMsg = ppMsg;
413.
414.     HCI_ReqWriteEncryptionMode(0, HCI_ENCRYPTION_OFF);
415.
416. }
417. void CRadioChatServerDlg::OnHciWriteEncryptionModeCnf(void **ppMsg)
418. {
419.     ppMsg = ppMsg;
420.
421.     HCI_ReqWriteAuthenticationMode(0, HCI_AUTH_DISABLE);
422. }
423. void CRadioChatServerDlg::OnHciWriteEncryptionModeCnfNeg(void **ppMsg)
424. {
425.     ppMsg = ppMsg;
426. }
427. void CRadioChatServerDlg::OnHciWriteAuthenticationModeCnf(void **ppMsg)
428. {
429.     ppMsg = ppMsg;
430.

```

```

431.     HCI_ReqWriteConnectTimeout(0,0x1FA0);
432. }
433.
434. void CRadioChatServerDlg::OnHciWriteAuthenticationModeCnfNeg(void **ppMsg)
435. {
436.     ppMsg = ppMsg;
437. }
438.
439. void CRadioChatServerDlg::OnHciWriteConnectTimeoutCnf(void **ppMsg)
440. {
441.     ppMsg = ppMsg;
442.
443.     HCI_ReqWritePageTimeout(0,8000);
444.
445. }
446.
447. void CRadioChatServerDlg::OnHciWriteConnectTimeoutCnfNeg(void **ppMsg)
448. {
449.     ppMsg = ppMsg;
450. }
451.
452. void CRadioChatServerDlg::OnHciWritePageTimeoutCnf(void **ppMsg)
453. {
454.     ppMsg = ppMsg;
455.
456.     HCI_ReqWriteCod(0,_tCod);
457.
458.
459.
460.
461.
462. }
463. void CRadioChatServerDlg::OnHciWritePageTimeoutCnfNeg(void **ppMsg)
464. {
465.     ppMsg = ppMsg;
466. }
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477. void CRadioChatServerDlg::OnHciWriteCodCnf(void **ppMsg)
478. {
479.     ppMsg = ppMsg;
480.
481.
482. }
483. void CRadioChatServerDlg::OnHciWriteCodCnfNeg(void **ppMsg)
484. {
485.     ppMsg = ppMsg;
486. }

```

```

487. void CRadioChatServerDlg::OnHciWriteNameCnf(void **ppMsg)
488. {
489.     ppMsg = ppMsg;
490.
491.     HCI_ReqWriteScanEnable(0, HCI_PAGE_SCAN_ENABLED |
HCI_INQUIRY_SCAN_ENABLED);
492. }
493. void CRadioChatServerDlg::OnHciWriteNameCnfNeg(void **ppMsg)
494. {
495.     ppMsg = ppMsg;
496. }
497. void CRadioChatServerDlg::OnHciWriteScanEnableCnf(void **ppMsg)
498. {
499.     ppMsg = ppMsg;
500.
501.     SCM_ReqRegister(0, SCM_SECURITY_HANDLER);
502. }
503. void CRadioChatServerDlg::OnHciWriteScanEnableCnfNeg(void **ppMsg)
504. {
505.     ppMsg = ppMsg;
506. }
507.
508. void CRadioChatServerDlg::OnHciInquiryCnf(void **ppMsg)
509. {
510.     HCI_TInquiryCnf      *ptInquiryCnf;
511.     int count;
512.     CRemoteDevice device;
513.
514.     ptInquiryCnf =(HCI_TInquiryCnf *) *ppMsg;
515.
516.     count = m_DevicesFound.GetSize();
517.     m_RemoteNameCounter = 0;
518.
519.     if (count > 0)
520.     {
521.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
522.         AfxMessageBox("remote name");
523.
524.         HCI_ReqRemoteName(10,
525.                             device.tAddress,
526.                             device.tPageScanPeriodMode,
527.                             device.tPageScanMode,
528.                             device.tClockOffset );
529.
530.     }
531.     else
532.     {
533.         AfxMessageBox("No device found");
534.     }
535.
536. }
537. void CRadioChatServerDlg::OnHciRemoteNameCnf(void **ppMsg)
538. {
539.     HCI_TRemoteNameCnf    *ptRemoteNameCnf;
540.     CRemoteDevice device;
541.     char sName[248];

```

```

542.     int count;
543.
544.     ptRemoteNameCnf =(HCI_TRemoteNameCnf *) *ppMsg;
545.
546.     sprintf(sName, "%s", &ptRemoteNameCnf->tName);
547.     m_DevicesFound[m_RemoteNameCounter].SetName( (CString) sName);
548.
549.     m_RemoteNameCounter++;
550.
551.     count = m_DevicesFound.GetSize();
552.
553.     if (count > m_RemoteNameCounter)
554.     {
555.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
556.         HCI_ReqRemoteName(10,
557.                             device.tAddress,
558.                             device.tPageScanPeriodMode,
559.                             device.tPageScanMode,
560.                             device.tClockOffset );
561.     }
562.     else
563.     {
564.
565.         ShowAllDevicesFound();
566.     }
567. }
568. void CRadioChatServerDlg::OnHciRemoteNameCnfNeg(void **ppMsg)
569. {
570.     HCI_TRemoteNameCnfNeg      *ptRemoteNameCnfNeg;
571.     CRemoteDevice device;
572.     int count;
573.
574.     ptRemoteNameCnfNeg =(HCI_TRemoteNameCnfNeg *) *ppMsg;
575.
576.     m_DevicesFound[m_RemoteNameCounter].SetName( (CString)_T("UNKNOWN"));
577.
578.     m_RemoteNameCounter++;
579.
580.     count = m_DevicesFound.GetSize();
581.
582.     if (count > m_RemoteNameCounter)
583.     {
584.         device = (CRemoteDevice) m_DevicesFound.GetAt(m_RemoteNameCounter);
585.         HCI_ReqRemoteName(10,
586.                             device.tAddress,
587.                             device.tPageScanPeriodMode,
588.                             device.tPageScanMode,
589.                             device.tClockOffset );
590.     }
591.     else
592.     {
593.         ShowAllDevicesFound();
594.     }
595. }
596. void CRadioChatServerDlg::OnScmConnectCnf(void **ppMsg)
597. {

```

```

598.   SCM_TConnectCnf *tConnectCnf = (SCM_TConnectCnf *)*ppMsg;
599.   AfxMessageBox("connected");
600.   tConnectCnf = tConnectCnf;
601.   m_ConnectionInfo.tAclHandle = tConnectCnf->tHandle;
602.   m_ConnectionInfo.tAddress = tConnectCnf->tAddress;
603.   OnGetServices();
604. }
605. void CRadioChatServerDlg::OnScmConnectCnfNeg(void **ppMsg)
606. {
607.   SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
608.   tConnectCnfNeg = tConnectCnfNeg;
609.   AfxMessageBox("No Connection made");
610. }
611. void CRadioChatServerDlg::OnSdConnectCnf(void **ppMsg)
612. {
613.   SD_TConnectCnf *tConnectCnf = (SD_TConnectCnf *)*ppMsg;
614.   SD_TUuid          *ptSearchPatternList;
615.   uint16             uiMaxRecords;
616.   uint8              ucNrOfUuids;
617.   m_ConnectionInfo.uiSdcHandle = tConnectCnf->uiSdcHandle;
618.   uiMaxRecords = 6;
619.   ucNrOfUuids = 1;
620.
621.   ptSearchPatternList = (SD_TUuid*)VOS_Alloc((uint16)
(ucNrOfUuids * sizeof(SD_TUuid)));
622.   ptSearchPatternList[0].eUuidType = SD_DET_UUID16;
623.   ptSearchPatternList[0].TUuid.uiUuid16 = SRP_SERIAL_GENERIC_SERIALPORT_UUID ;
//SRP_HEADSET_UUID;
624.
625.   SD_ReqServiceSearch (0, m_ConnectionInfo.uiSdcHandle, uiMaxRecords,
ucNrOfUuids, ptSearchPatternList);
626. }
627. void CRadioChatServerDlg::OnSdConnectCnfNeg(void **ppMsg)
628. {
629.   SD_TConnectCnfNeg *tConnectCnfNeg = (SD_TConnectCnfNeg *)*ppMsg;
630.   CString str;
631.   str.Format("Could not connect to SD , Error %d",
tConnectCnfNeg->tHdr.iResult);
632.   MessageBox(str);
633. }
634. void CRadioChatServerDlg::OnSdServiceSearchCnf(void **ppMsg)
635. {
636.   SD_TServiceSearchCnf *tServiceSearchCnf = (SD_TServiceSearchCnf *)*ppMsg;
637.   uint16                uiCurrentServiceRecordCount;
638.   uint32                *pulSRHandles;
639.   uint16                *puiAttributeIDList;
640.   uint8                 ucNrOfAttr;
641.   CService              service;
642.   uiCurrentServiceRecordCount =
tServiceSearchCnf->uiCurrentServiceRecordCount;
643.   pulSRHandles = (uint32*)
VOS_Alloc(((uint16)(uiCurrentServiceRecordCount*sizeof(uint32))));
644.
645.   (void*)memcpy(pulSRHandles,
646. &tServiceSearchCnf->ulServiceRecordHandleList,

```

```

647.         (uiCurrentServiceRecordCount*sizeof(uint32)));
648.
649. m_ConnectionInfo.ulServiceRecordHandle =
tServiceSearchCnf->ulServiceRecordHandleList;
650.     ucNrOfAttr = 1;
651.     puiAttributeIDList =
(uint16*)VOS_Alloc( (uint16) (ucNrOfAttr*sizeof(uint16)));
652.     puiAttributeIDList[0] = BT_SERVICE_NAME(0);
653.     service.m_SDCHandle = m_ConnectionInfo.uiSdcHandle;
654.     service.m_ServiceRecordHandle =
tServiceSearchCnf->ulServiceRecordHandleList;
655.     m_ServicesFound.SetAtGrow(m_ServiceCounter,service);
656.     SD_ReqServiceAttribute(1, m_ConnectionInfo.uiSdcHandle, pulSRHandles[0],
ucNrOfAttr, puiAttributeIDList);
657.     VOS_Free((void*)&puiAttributeIDList);
658.     VOS_Free((void*)&pulSRHandles);
659. }
660. void CRadioChatServerDlg::OnSdServiceSearchCnfNeg(void **ppMsg)
661. {
662.     SD_TServiceSearchCnfNeg *tConnectCnfNeg = (SD_TServiceSearchCnfNeg
*)*ppMsg;
663.     CString str;
664.     tConnectCnfNeg = tConnectCnfNeg;
665.     str.Format("Service Search Confirm Negative, Error %d",tConnectCnfNeg-
>tHdr.iResult);
666.     MessageBox(str);
667. }
668. void CRadioChatServerDlg::OnSdServiceAttributeCnf(void **ppMsg)
669. {
670.     SD_TServiceAttributeCnf *tServiceAttributeCnf = (SD_TServiceAttributeCnf
*)*ppMsg;
671.     CService service;
672.     AfxMessageBox("sdser");
673.     switch (tServiceAttributeCnf->tHdr.uiSeqNr)
674.     {
675.     case 1:
676.         ReceiveServiceName(tServiceAttributeCnf);
677.         AskForServiceRecordHandle();
678.         AfxMessageBox("hai");
679.         break;
680.     case 2:
681.         ReceiveServiceRecordHandle(tServiceAttributeCnf);
682.         AfxMessageBox("SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle)");
683.         SD_ReqDisconnect(0,m_ConnectionInfo.uiSdcHandle);
684.         break;
685.     default:
686.         break;
687.     }
688. }
689. void CRadioChatServerDlg::OnSdServiceAttributeCnfNeg(void **ppMsg)
690. {
691.     SD_TServiceAttributeCnfNeg *tConnectCnfNeg = (SD_TServiceAttributeCnfNeg
*)*ppMsg;
692.     CString str;
693.     tConnectCnfNeg = tConnectCnfNeg;

```



```

694.     str.Format("Service Attribute Confirm negative, error %d",tConnectCnfNeg-
>tHdr.iResult);
695.     MessageBox(str);
696. }
697. void CRadioChatServerDlg::OnSdDisconnectCnf(void **ppMsg)
698. {
699.     ppMsg = ppMsg;
700.     Beep (1000,200);
701.     OnSelservices();
702. }
703. void CRadioChatServerDlg::OnDbmRegisterServiceCnf(void **ppMsg)
704. {
705.
706.     DBM_TRegisterServiceCnf *ptRegisterCnf = (DBM_TRegisterServiceCnf *)
*ppMsg;
707.     m_pSerialPort->WriteProfile(ptRegisterCnf->ulDbmHandle);
708.     COM_ReqRegister(PROFILE_SERIAL,/* use this as sequence number */
709.                     0);
710. }
711. void CRadioChatServerDlg::OnDbmRegisterServiceCnfNeg(void **ppMsg)
712. {
713.
714.     ppMsg = ppMsg;
715.     MessageBox(_T("Could not register to Data Base Manager"));
716.     DestroyWindow();
717. }
718. void CRadioChatServerDlg::OnDbmAddDescriptorCnf(void **ppMsg)
719. {
720.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
721.
722.     tConnectCnfNeg = tConnectCnfNeg;
723.     AfxMessageBox("star chat");
724.     AfxMessageBox("end chat");
725.     Beep (1000,200);
726. }
727. void CRadioChatServerDlg::OnDbmAddDescriptorCnfNeg(void **ppMsg)
728. {
729.     SCM_TConnectCnfNeg *tConnectCnfNeg = (SCM_TConnectCnfNeg *)*ppMsg;
730.     tConnectCnfNeg = tConnectCnfNeg;
731.     MessageBox(_T("Could not register the service to DBM"));
732. }
733.
734. void CRadioChatServerDlg::OnConnectAcceptInd(void **ppMsg)
735. {
736.     SCM_TConnectAcceptInd *ptConnectAcceptInd;
737.     ptConnectAcceptInd = (SCM_TConnectAcceptInd *) *ppMsg;
738.     SCM_RspConnectAccept( (MSG_TMMsg **)ppMsg,
739.                           SCM_POS_RESULT,
740.                           ptConnectAcceptInd->tAddress,
741.                           SCM_SLAVE);
742.
743.     *ppMsg = NULL;
744. }
745. void CRadioChatServerDlg::OnHciInquiryEvt(void **ppMsg)
746. {
747.     HCI_TInquiryEvt     *ptInquiryEvt;

```

```

748. CRemoteDevice device;
749. ptInquiryEvt =(HCI_TInquiryEvt *) *ppMsg;
750. device.tAddress = ptInquiryEvt->tAddress;
751. device.tPageScanMode = ptInquiryEvt->tPageScanMode;
752. device.tPageScanPeriodMode = ptInquiryEvt->tPageScanPeriodMode;
753. device.tClockOffset = ptInquiryEvt->tClockOffset;
754. device.tCod = ptInquiryEvt->tCod;
755. device.tPageScanRepMode = ptInquiryEvt->tPageScanRepMode;
756. AddDevice(device);
757. }
758.
759.
760. void CRadioChatServerDlg::OnScmPincodeInd(void **ppMsg)
761. {
762.     SCM_TPINcodeInd          *ptPincodeInd;
763.
764.     /* request for a PIN-code from the remote side. */
765.
766.     ptPincodeInd =(SCM_TPINcodeInd *) *ppMsg;
767.
768.     /* Reply positive. */
769.     SCM_RspPincode( (MSG_TMsg **)ppMsg,
770.                     SCM_POS_RESULT,
771.                     ptPincodeInd->tAddress,
772.                     _tPincode,
773.                     PINCODE_LENGTH);
774. }
775.
776.
777. void CRadioChatServerDlg::OnScmConnectEvt(void **ppMsg)
778. {
779.     SCM_TConnectEvt *tConnectEvt = (SCM_TConnectEvt *)*ppMsg;
780.
781.     // this event is received because we are also registerd as
782.     SCM_MONITOR_GROUP
783.     tConnectEvt = tConnectEvt;
784.
785.     // remember some connection information
786.     m_ConnectionInfo.tAclHandle = tConnectEvt->tHandle;
787.     m_ConnectionInfo.tAddress = tConnectEvt->tAddress;
788.
789.     // Disable the get devices button on the screen
790.     // because you can not do a inquiry when a connection has been established
791.     // m_Inquiry.EnableWindow(FALSE);
792. }
793.
794. void CRadioChatServerDlg::OnScmDisconnectEvt(void **ppMsg)
795. {
796.
797.     ppMsg = ppMsg;
798.     // connection is removed
799.
800.     // so we will end also this security application
801.     m_ConnectionInfo.tAclHandle = 0;
802.

```

```

803.   OnCloseapplication();
804. }
805.
806. void CRadioChatServerDlg::OnHciStartCnf(void **ppMsg)
807. {
808.   HCI_TStartCnf *ptStartCnf = (HCI_TStartCnf *)*ppMsg;
809.
810.   ptStartCnf = ptStartCnf;
811.
812.   HCI_ReqConfigurePort(0,PORTSETTINGS);
813. }
814.
815.
816. void CRadioChatServerDlg::OnComVersionCnf(void **ppMsg)
817. {
818.   CAboutDlg      Abodlg;
819.
820.   COM_TVersionCnf* ptVersionCnf;
821.   char* cpVerStr = NULL;
822.   int8 iCharCount = 9;
823.   char cpStr[3];
824.
825.   ptVersionCnf = (COM_TVersionCnf *) *ppMsg;
826.   cpVerStr = &ptVersionCnf->cVersion;
827.   do
828.   {
829.     iCharCount++;
830.     cpStr[iCharCount-10] = cpVerStr[iCharCount];
831.   }while(iCharCount <= 11);
832.   cpStr[3] = ((char)0);
833.
834.   //Abodlg.SetVersionText((CString) cpStr);
835.
836.   Abodlg.DoModal();
837.
838. }
839.
840.
841.
842.
843.
844.
845.
846.
847.
848. void CRadioChatServerDlg::AskForServiceName()
849. {
850.   uint16          *puiAttributeIDList;
851.   uint8           ucNrOfAttr;
852.
853.   /* Read PDL attributes */
854.   ucNrOfAttr = 1;
855.   puiAttributeIDList =
(uint16*)VOS_Alloc((uint16) (ucNrOfAttr*sizeof(uint16)));
856.   puiAttributeIDList[0] = BT_SERVICE_NAME(0);
857.

```

```

858.
859.   SD_ReqServiceAttribute(0, m_ConnectionInfo.uiSdcHandle,
m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr, puiAttributeIDList);
860.
861.   VOS_Free((void*)&puiAttributeIDList);
862. }
863.
864.
865. void CRadioChatServerDlg::ReceiveServiceName(SD_TServiceAttributeCnf
*tServiceAttributeCnf)
866. {
867.   CService service;
868.
869.   /* the name of the profile is received */
870.
871.   /* attribute message has the following format */
872.
873.   /*
874.   SD_DET_SEQUENCE8|Lenth|SD_DET_UINT16|byte0|byte1|SD_DET_STRING8
|length of string|"Headset"
875.   */
876.
877.   service = m_ServicesFound.GetAt(m_ServiceCounter);
878.
879.   // Remember necessary items.
880.   service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
881.   service.m_AttributeListByteCount = tServiceAttributeCnf-
>uiAttributeListByteCount;
882.   // service.m_pAttributeData = (uint8*)
VOS_Alloc(service.m_AttributeListByteCount);
883.   (void*)memcpy(service.m_pAttributeData,
884.                  &tServiceAttributeCnf->ucAttributeData,
885.                  service.m_AttributeListByteCount);
886.
887.   // service.m_pServiceName = (char*) VOS_Alloc(service.m_pAttributeData[6]
+ 1); /* +1 for the NULL char */
888.
889.   (void*)memcpy(service.m_pServiceName,
890.                  &service.m_pAttributeData[7],
891.                  service.m_pAttributeData[6]);
892.
893.   service.m_pServiceName[service.m_pAttributeData[6]] = NULL;
894.
895.   m_ServicesFound.SetAt(m_ServiceCounter, service);
896.
897.   m_ServiceCounter++;
898.
899.   /* also remember this in the connectioninfo class. */
900.   m_ConnectionInfo.uiAttributeListByteCount = tServiceAttributeCnf-
>uiAttributeListByteCount;
901.   // m_ConnectionInfo.pucAttributeData = (uint8*)
VOS_Alloc(service.m_AttributeListByteCount);
902.   (void*)memcpy(m_ConnectionInfo.pucAttributeData,
903.                  &tServiceAttributeCnf->ucAttributeData,
904.                  m_ConnectionInfo.uiAttributeListByteCount);
905.

```

```

906. // m_ConnectionInfo.pcServiceName = (int8*)
VOS_Alloc(service.m_pAttributeData[6] + 1); /* +1 for the NULL char */
907.
908. (void*)memcpy(m_ConnectionInfo.pcServiceName,
909.               &service.m_pAttributeData[7],
910.               service.m_pAttributeData[6]);
911.
912. m_ConnectionInfo.pcServiceName[service.m_pAttributeData[6]] = NULL;
913.
914. ShowAllServicesFound();
915.
916. //m_SelService.EnableWindow(TRUE);
917. }
918.
919.
920. void CRadioChatServerDlg::AskForServiceRecordHandle()
921. {
922.     uint16                *puiAttributeIDList;
923.     uint8                 ucNrOfAttr;
924.
925.     /* Read PDL attributes */
926.     ucNrOfAttr = 2;
927.     puiAttributeIDList =
(uint16*)VOS_Alloc((uint16) (ucNrOfAttr*sizeof(uint16)));
928.     puiAttributeIDList[0] = BT_SERVICE_RECORD_HANDLE;
929.     puiAttributeIDList[1] = BT_PROTOCOL_DESCRIPTOR_LIST;
930.
931.
932.
933.
934.
935.
936.
937. SD_ReqServiceAttribute(2, m_ConnectionInfo.uiSdcHandle,
m_ConnectionInfo.ulServiceRecordHandle, ucNrOfAttr, puiAttributeIDList);
938.
939.     VOS_Free((void*)&puiAttributeIDList);
940. }
941.
942.
943. void RadioChatServerDlg::ReceiveServiceRecordHandle
(SD_TServiceAttributeCnf *tServiceAttributeCnf)
944. {
945.     CService service;
946.
947.     service = m_ServicesFound.GetAt(m_ServiceCounter-1);
948.
949.     /* Remember necessary items. */
950.     service.m_SDCHandle = tServiceAttributeCnf->uiSdcHandle;
951.     service.m_AttributeListByteCount =
tServiceAttributeCnf->uiAttributeListByteCount;
952. // service.m_pAttributeData = (uint8*)
VOS_Alloc(service.m_AttributeListByteCount);
953.     (void*)memcpy(service.m_pAttributeData,
954.                   &tServiceAttributeCnf->ucAttributeData,
955.                   service.m_AttributeListByteCount);

```

```

956.
957.
958.
959.   m_ServicesFound.SetAt(m_ServiceCounter-1,service);
960.   m_ServiceCounter++;
961.
962.
963.   /* also remember this in the connectioninfo class. */
964.   m_ConnectionInfo.uiAttributeListByteCount =
965.       tServiceAttributeCnf->uiAttributeListByteCount;
966.   (void*)memcpy(m_ConnectionInfo.pucAttributeData,
967.               &tServiceAttributeCnf->ucAttributeData,
968.               m_ConnectionInfo.uiAttributeListByteCount);
969. }
970.
971.
972.
973.
974.
975.
976.
977. void CRadioChatServerDlg::OnCloseapplication()
978. {
979.
980.     // Application is about to close
981.
982.     // De Register from the component
983.
984.     // First SCM
985.
986.     SCM_ReqDeRegister(1,SCM_SECURITY_HANDLER);
987. }
988.
989.
990. void CRadioChatServerDlg::OnScmDeRegisterCnf(void **ppMsg)
991. {
992.     SCM_TDeRegisterCnf *ptDeRegisterCnf = (SCM_TDeRegisterCnf *) *ppMsg;
993.
994.     switch (ptDeRegisterCnf->tHdr.uiSeqNr)
995.     {
996.     case 1: // DeRegister from SECURITY HANDLER
997.         // unregister now from the monitor group
998.         SCM_ReqDeRegister(2,SCM_MONITOR_GROUP);
999.         break;
1000.    case 2: // DeRegister from MONITOR GROUP
1001.        // unregister from DBM
1002.        if (m_ConnectionInfo.ulDbmHandle > 0)
1003.        {
1004.            DBM_ReqUnRegisterService(3,m_ConnectionInfo.ulDbmHandle);
1005.        }
1006.        else
1007.        {
1008.            if (m_ConnectionInfo.tAclHandle>0)
1009.            {
1010.                // Close the ACL Connection

```

```

1011.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
1012.     }
1013.     else
1014.     {
1015.         //Close the window program
1016.         DestroyWindow();
1017.     }
1018. }
1019.     break;
1020. default:
1021.     break;
1022. }
1023. }
1024.
1025.
1026. void CRadioChatServerDlg::OnScmDeRegisterCnfNeg(void **ppMsg)
1027. {
1028.     ppMsg = ppMsg;
1029.
1030.     MessageBox(_T("Could not unregister from SCM"));
1031.
1032.     DestroyWindow();
1033. }
1034.
1035.
1036. void CRadioChatServerDlg::OnDbmUnRegisterServiceCnf(void **ppMsg)
1037. {
1038.     ppMsg = ppMsg;
1039.
1040.     // Close the application
1041.
1042.     if (m_ConnectionInfo.tAclHandle>0)
1043.     {
1044.         SCM_ReqDisconnect(0,m_ConnectionInfo.tAclHandle);
1045.     }
1046.     else
1047.     {
1048.         DestroyWindow();
1049.     }
1050. }
1051.
1052.
1053. void CRadioChatServerDlg::OnDbmUnRegisterServiceCnfNeg(void **ppMsg)
1054. {
1055.     ppMsg = ppMsg;
1056.     // let the user know
1057.     MessageBox(_T("Not possible to UnRegister from DBM"));
1058.
1059.     DestroyWindow();
1060. }
1061.
1062.
1063. void CRadioChatServerDlg::OnScmDisconnectCnf(void **ppMsg)
1064. {
1065.
1066.     ppMsg = ppMsg;

```

```

1067. // connection is removed
1068.
1069. // so we will end also this security application
1070. m_ConnectionInfo.tAclHandle = 0;
1071.
1072. DestroyWindow();
1073. }
1074.
1075.
1076. void CRadioChatServerDlg::OnScmDisconnectCnfNeg(void **ppMsg)
1077. {
1078.
1079.     ppMsg = ppMsg;
1080.     // connection could not be removed removed
1081.
1082.     MessageBox(_T("Could not remove ACL connection"));
1083.
1084.     DestroyWindow();
1085. }
1086.
1087.
1088. BOOL CRadioChatServerDlg::DestroyWindow()
1089. {
1090.     // TODO: Add your specialized code here and/or call the base class
1091.
1092.     return CDialog::DestroyWindow();
1093. }
1094. void CRadioChatServerDlg::ShowAllDevicesFound()
1095. {
1096.     CRemoteDevice device;
1097.     int iFound,i;
1098.     iFound = m_DevicesFound.GetSize();
1099.     for (i=0; i < iFound; i++)
1100.     {
1101.         device = m_DevicesFound.GetAt(i);
1102.         AfxMessageBox("device1");
1103.         hdevice1=m_tree.InsertItem(device.GetAddress(), hPA, TVI_SORT);
1104.         OnSelDevice();
1105.     }
1106. }
1107. void CRadioChatServerDlg::AddService(CString sService)
1108. {
1109.     CService service(sService);
1110.     m_ServicesFound.Add(service);
1111. }
1112. void CRadioChatServerDlg::AddService(CService service)
1113. {
1114.     m_ServicesFound.Add(service);
1115. }
1116. void CRadioChatServerDlg::ShowAllServicesFound()
1117. {
1118.     CService service;
1119.     int iFound,i;
1120.     iFound = m_ServicesFound.GetSize();
1121.     for (i=0; i < iFound; i++)
1122.     {

```



```

1123.     service = m_ServicesFound.GetAt(i);
1124.     m_tree.InsertItem(service.GetService(),hdevice1,TVI_LAST);
1125.     }
1126. }
1127. void CRadioChatServerDlg::AddDevice(CRemoteDevice device)
1128. {
1129.     m_DevicesFound.Add(device);
1130. }
1131. void CRadioChatServerDlg::OnInquiry()
1132. {
1133.
1134.     HCI_TLap    tLap = {0x9E,0x8B,0x33};
1135.     HCI_TInquiryLength    tInquiryLength = 2;
1136.     HCI_TNrOfResponses    tNrOfResponses = 0;
1137.     HCI_ReqInquiry(1,tLap,tInquiryLength,tNrOfResponses);
1138. }
1139. void CRadioChatServerDlg::OnSelDevice()
1140. {
1141.
1142.     CRemoteDevice device;
1143.     device = m_DevicesFound.GetAt(0);
1144.     m_ConnectionInfo.tAddress = device.tAddress;
1145.     SCM_ReqConnect(0,
1146.                   device.tAddress,
1147.                   SCM_DM1,
1148.                   SCM_R1,
1149.                   SCM_MANDATORY_PAGE_SCAN_MODE,
1150.                   0,
1151.                   SCM_NOT_ACCEPT_ROLE_SWITCH);
1152. }
1153.
1154. void CRadioChatServerDlg::OnSelServices()
1155. {
1156.
1157.     CService service;
1158.     service = m_ServicesFound.GetAt(0);
1159.     m_ConnectionInfo.ulServiceRecordHandle = service.m_ServiceRecordHandle;
1160.     DBM_ReqRegisterService(0, DBM_StackDB);
1161.
1162. }
1163.
1164. void CRadioChatServerDlg::OnGetservices()
1165. {
1166.
1167.     m_ServiceCounter = 0;
1168.     SD_ReqConnect(0,SD_DEFAULT_MFS,m_ConnectionInfo.tAclHandle);
1169. }
1170. void CRadioChatServerDlg::OnComConnectCnf(void **ppMsg)
1171. {
1172.     COM_TConnectCnf *ptConnectCnf = (COM_TConnectCnf *) *ppMsg;
1173.     m_ConnectionInfo.uiRFCCommHandle = ptConnectCnf->uiHandle;
1174.     MessageBox(_T(" RFCOMM connection"));
1175.     Beep (1000,200);
1176.     Sleep(100);
1177.     Beep (1000,200);
1178. }

```

```

1179. void CRadioChatServerDlg::OnComConnectCnfNeg(void **ppMsg)
1180. {
1181.     COM_TConnectCnfNeg *ptConnectCnfNeg = (COM_TConnectCnfNeg *) *ppMsg;
1182.     ptConnectCnfNeg = ptConnectCnfNeg;
1183.     m_ConnectionInfo.uiRFCCommHandle = 0;
1184.     MessageBox(_T("Could not create a RFCOMM connection"));
1185. }
1186.
1187. void CRadioChatServerDlg::OnButton2()
1188. {
1189.     SCM_ReqRegister(0, SCM_MONITOR_GROUP);
1190. }
1191. void CRadioChatServerDlg::OnScmRegisterCnf(void **ppMsg) //2
1192. {
1193.     ppMsg = ppMsg;
1194.     SDS_ReqStart(0);
1195. }
1196. void CRadioChatServerDlg::OnScmRegisterCnfNeg(void **ppMsg) //2
1197. {
1198.     SCM_TRegisterCnfNeg *ptMsg = (SCM_TRegisterCnfNeg *) *ppMsg;
1199.     ptMsg = ptMsg;
1200.     MessageBox(_T("Not possible to register to SCM"));
1201.     DestroyWindow();
1202. }
1203. void CRadioChatServerDlg::OnSdsStartCnf(void **ppMsg)
1204. {
1205.     CString sAddress;
1206.     SDS_TStartCnf *ptStartCnf;
1207.     ptStartCnf = (SDS_TStartCnf *) *ppMsg;
1208.     m_pSerialPort = new CRS232(PROFILE_SERIAL);
1209. }
1210. void CRadioChatServerDlg::OnComRegisterCnf(void **ppMsg)
1211. {
1212.     COM_TRegisterCnf *ptRegisterCnf = (COM_TRegisterCnf *) *ppMsg;
1213.     m_RFServerChannel = ptRegisterCnf->ucServerChannel;
1214.     COM_ReqFillPdl(PROFILE_SERIAL,
1215.                    ptRegisterCnf->ucServerChannel,
1216.                    (uint16)m_pSerialPort->GetSRPHandle());
1217. }
1218. void CRadioChatServerDlg::OnComRegisterCnfNeg(void **ppMsg)
1219. {
1220.     ppMsg = ppMsg;
1221.     MessageBox(_T("Not possible to register to RFCOM"));
1222.     DestroyWindow();
1223. }
1224. void CRadioChatServerDlg::OnComFillPdlCnf(void **ppMsg)
1225. {
1226.     COM_TFillPdlCnf *ptFillPdlCnf = (COM_TFillPdlCnf *) *ppMsg;
1227.     char *pcName = NULL;
1228.     CString sName;
1229.     ptFillPdlCnf = ptFillPdlCnf;
1230.     m_pSerialPort->GetProfileName(&pcName);
1231.     Beep(1000,100);
1232. }
1233. }
1234. void CRadioChatServerDlg::OnComFillPdlCnfNeg(void **ppMsg)

```

```

1235. {
1236.     ppMsg = ppMsg;
1237.     MessageBox(_T("Not possible to fill PDL for RF COM"));
1238.     DestroyWindow();
1239.
1240. }
1241. void CRadioChatServerDlg::OnComConnectInd(void **ppMsg)
1242. {
1243.     COM_TConnectInd *tConnectInd = (COM_TConnectInd *)*ppMsg;
1244.
1245.     m_RFCommHandle = tConnectInd->uiHandle;
1246.     AfxMessageBox("In Comm Connection Ind");
1247.     COM_RspConnect((MSG_TMsg **)ppMsg, COM_POS_RESULT,
tConnectInd->uiMaxFrameSize);
1248.     *ppMsg = NULL;
1249.
1250. }
1251. BOOL CRadioChatServerDlg::PreTranslateMessage(MSG* pMsg)
1252. {
1253.     if (pMsg->message == WM_KEYDOWN)
1254.     {
1255.         if (pMsg->wParam == VK_RETURN)
1256.         {
1257.             HandleReturn();
1258.             return FALSE;
1259.         }
1260.     }
1261.
1262.     return CDialog::PreTranslateMessage(pMsg);
1263. }
1264.
1265. void CRadioChatServerDlg::HandleReturn()
1266. {
1267.     CHAR sChatStr[80];
1268.     uint8 *pucData;
1269.     uint16 iCount, i;
1270.     iCount = (uint16) m_InputChat.GetWindowText(sChatStr, 80);
1271.     if (iCount > 0)
1272.     {
1273.         pucData = COM_DataAlloc((uint16)(iCount + 1));
1274.         for (i=0; i < iCount; i++)
1275.         {
1276.             pucData[i] = sChatStr[i];
1277.         }
1278.         pucData[i] = 0;
1279.         COM_DataSend(0, pucData, m_RFCommHandle, (uint16)(iCount + 1));
1280.         m_ChatArea.InsertString(index, (CString)sChatStr);
1281.         m_InputChat.SetWindowText(_T(""));
1282.         index++;
1283.     }
1284. }
1285.
1286. void CRadioChatServerDlg::OnComDataInd(void **ppMsg)
1287. {
1288.     COM_TDataInd *tDataInd = (COM_TDataInd *)*ppMsg;
1289.     uint8 *pucData;

```

```

1290.  CHAR  sData[80];
1291.  uint16 uiLength;
1292.  uint16 uiHandle;
1293.  int i;
1294.  pucData = COM_DataExtract((MSG_TDataMsg *)*ppMsg,
1295.                           &uiLength,
1296.                           &uiHandle);
1297.
1298.  COM_RspData(tDataInd->tHdr.ucSeqNr, COM_POS_RESULT, uiHandle);
1299.  for (i=0; i < uiLength; i++)
1300.  {
1301.      sData[i] = pucData[i];
1302.  }
1303.  m_ChatArea.InsertString(index, (CString)sData);
1304.  index++;
1305. }

```

Code Description

To the greatest extent possible, the code of file transfer application has been reused for this example. The differences are explained below.

- ◆ Lines 1–1250: Explained in `RadioFileServerDlg.cpp` file.
- ◆ Lines 1251–1263: As soon as the Enter key is pressed, the function `HandleReturn` will be invoked to send the data to the remote device.
- ◆ Lines 1265–1284: The data entered in the chat Input is retrieved and sent to the remote BLUETOOTH peer device. The data is displayed in the chat area list box.
- ◆ Lines 1286–1305: When `COM_DATA_IND` event is fired by the BLUETOOTH module, the command to read the incoming data from the client is issued. The response is sent to the client to indicate that the data has been received. The data that has been read is added to the chat area in the dialog box.

Code Output

When the application is built, the screen in Figure 9-10 appears. The screen has one edit box to enter messages and a list box to show the exchange of messages between the server and the clients.

Summary

In this chapter, we discussed the programming aspects of Bluetooth. Using Ericsson's Bluetooth PC Reference Stack and Bluetooth module, two desktop PCs become Bluetooth enabled. The API calls provided in the Reference stack enable us to access the services of different layers. The HCI programming example illustrated how the API calls can be used for the stack to communicate with the Bluetooth module to obtain the Bluetooth device address, version number, packet sizes supported for ACL and SCO links, how to do loopback testing and also how to obtain the remote Bluetooth device address. The SDP programming example has been used to demonstrate how to register services on a Bluetooth device, which can be accessed by other devices in the piconet. Then, we studied how to create a full-fledged application for file transfer using RFCOMM, SDP, and HCI programming. We also discussed how the file transfer application could be extended to create a chat application.

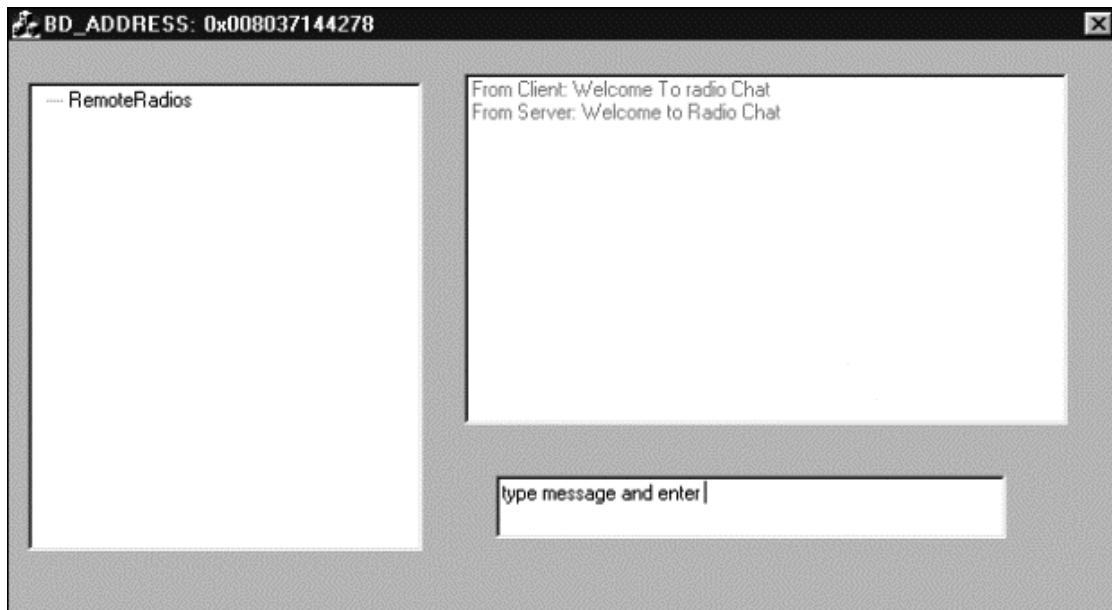


Figure 9-10: Output of the Server Module

This application can be easily extended to demonstrate WAP with Bluetooth applications. Reusing the file transfer code to the maximum possible extent, we also presented chat application implementation. The code given here covers the most important API calls for creating data applications using the Ericsson's PC Reference stack. All these applications are built using the API calls and the sample code given in this Bluetooth SDK.

For Bluetooth enabling of a device, the protocol stack needs to be ported to that device. With a thorough understanding of the protocol implementation discussed in this chapter, if the source code for the stack is available, one can easily port the code on to any platform.

Chapter 10

An Overview of 3G

The need to provide communication facilities for people on the move led to the development of cellular mobile communication systems. Even though voice continues to be the killer application on mobile communication systems, the demand for mobile data services is increasing rapidly. According to market projections, there will be one billion wireless Internet Access Devices by the year 2003. Development of sophisticated mobile devices and high data rate mobile networks are paving the way for exciting times — wireless network multi-media services that provide “anywhere, anytime” communication will soon be a reality. To provide these services will be a technological challenge because the operators have already installed billions of dollars worth of equipment, which presently supports only low data rates. The challenge is to develop systems that can support high data rates without throwing away the existing infrastructure.

In this chapter, we study the various “generations” of wireless networks. The first generation wireless networks were analog systems. The second generation systems, which are currently operational, are digital but support low data rates. We study the architecture of the most popular second generation wireless system — Global System for Mobile Communications (GSM), which was standardized in Europe but adapted by many other countries. Then, we study the 2.5 Generation (2.5G) and third generation (3G) networks, which support higher data rates to provide multimedia applications. We briefly review the underlying telecommunication technologies but focus more on content development to use the 3G technologies. To provide the 3G services, we need sophisticated mobile devices for subscribers to access the Internet content, advanced languages, and tools for content development. We review these aspects in this chapter. In the next chapter we will study programming aspects to develop sophisticated applications over 3G networks.

This chapter gives the basic theory of 3G technologies, thus we won’t discuss any programming aspects of 3G. You must understand these technologies if you want to take up content development for 3G networks.

Principles of Cellular Mobile Communications

During the initial days of mobile communications, mobile systems were structured like TV broadcasting systems. A base station with a very powerful transmitter located at the highest point in a service locality (for example, a city) catered to an area of about 50 km radius. The mobile terminals were generally car-mounted with a transmitter, receiver, and an antenna. For communication by one mobile terminal, one channel is used. A channel consists of two frequencies: one frequency for communication from the base station to the mobile terminal (called downlink) and one frequency for communication from the mobile terminal to the base station (called uplink). Each base station is assigned a number of channels, based on the subscriber density in the region.

These traditional mobile systems can be called single cell systems, as the entire coverage area is only one cell. The disadvantages of this type of system are:

- ♦ Very powerful transmitters are required at the base station and the mobile terminals — high power transmitters are costly.

- ◆ The capacity of the system is very low, as only a fixed number of channels are available for a service area (because of the limited radio spectrum).
- ◆ The number of subscribers who can make calls simultaneously is limited.
- ◆ The size of the mobile terminal is large because of the high-power transmitter.
- ◆ Expansion of the system to cater to a higher number of subscribers is very difficult.

To overcome all the above limitations, multi-cell systems have been developed.

Multi-Cell Wireless Networks

Bell Labs developed the concept of multi-cell systems in the early 1970s, and the Nordic countries were the first to introduce commercial multi-cell mobile systems in 1981. In a multi-cell system, the service area is divided into “cells” as shown in Figure 10-1. A *cell* is the basic geographic unit in a mobile system. Each cell is represented by a hexagon. Each cell will have a base station, with a low power transmitter. The size of the cell may vary, depending on the terrain: natural terrain (such as mountains, lakes, and so on) or man-made terrain (such as buildings and such). Each cell is allocated some channels and all the mobiles, when they are in that cell, use one of these channels for communication. The main attraction of this approach is that it requires of very low power transmitters at the base stations, as well as for the mobile phones.

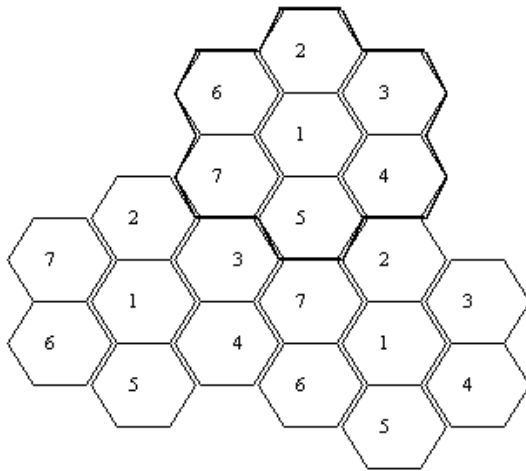


Figure 10-1: Multi-cell system with service area divided into cells (seven-cell cluster)

In a multi-cell system, two adjacent cells cannot use the same channel because there will be interference. So, if a mobile subscriber moves from one cell to another cell while the call is in progress, there are two options: either the call has to be dropped or the mobile terminal has to switch to the one of the channels used in the new cell. Because dropping a call is not acceptable, the other option is chosen. When the mobile terminal is at the edge of one cell, the signal strength goes down, and the mobile terminal monitors the signal strengths of the channels in the adjacent cells and switches to the channel for which signal strength is high. The call will not be dropped, but conversation can continue using the new channel. This process is called *handover*. Certainly, handover introduces complexity in cellular mobile communication, but it has many advantages:

- ◆ Because of the low power required at each base station, as well as the mobile terminals, low cost systems can be developed. The size of the mobile terminals also are smaller.

- ◆ Depending on the distance between the mobile terminal and the base station, variable power levels can be used for communication, thereby reducing the power requirements and hence, the battery requirements of the mobile terminals.
- ◆ Based on the number of channels allocated for each cell, there is a limit on the number of simultaneous calls. If the subscriber capacity or traffic increases in a cell over a period of time, a cell can be split and new base stations can be installed.
- ◆ The cell size is not fixed; cells can be of different sizes. In urban areas with high subscriber density, cell size can be small (as small as 500 meters), and in rural areas cell size can be large (as large as 30 kilometers).

Cellular System Design Issues

When we take on the complicated task of designing a cellular communications system, we must address many issues. Some important issues are discussed in this section.

Radio Engineering

Based on the terrain of the service area (keeping in view the hills, lakes, high-rise buildings) and the likely subscriber density in different areas, the service area has to be divided into different cells. The hexagonal cell is only a theoretical representation; in practice, there may be overlaps of the cells and some small regions may not be covered by the radio. A radio survey is carried out to find out the best location for installing the base stations and to ensure maximum possible coverage.

Frequency Allocation and Frequency Reuse

For each base station located in a cell, a number of channels (pairs of frequencies) have to be assigned. The number of channels is based on the subscriber capacity in that locality and the maximum number of simultaneous calls that should be supported. A mobile operator has to obtain the frequency allocation from a national authority. Though adjacent cells cannot use the same channels, the same channels can be reused provided there is a minimum separation distance between the cells using the same channels. The concept of clusters is of importance here. *Cluster* is a group of cells; no channels are reused within a cluster. In frequency reuse, each cell is assigned a group of radio channels and the same channels can be reused in another cluster of cells. In Figure 10-1, a seven-cell cluster is shown. Cells denoted by “1” in all the three clusters can use the same set of channels; cells denoted by “2” in all three clusters can use the same set of channels, and so on.

PSTN Connectivity

The mobile network is connected to the Public Switched Telephone Network (PSTN) to enable normal telephone subscribers to be contacted by the mobile subscribers and vice versa. The trunk capacity between the mobile switching system and the PSTN switch has to be decided based on the traffic considerations.

Cell Splitting

For economic reasons, at the outset of development, the cellular service provider does not design the cellular system with small cells. The service provider may have large cells to start with, and as the subscriber capacity increases, the cells will be split, more base stations will be installed, and the frequency reuse pattern is reworked out.

Shadow Regions

Because of the terrain, some areas may not have radio coverage; such regions are called *shadow regions*. In shadow regions, mobile calls are dropped. The cellular operator has to ensure that no shadow regions exist, or at least that their area is minimized.

Traffic Analysis

As soon as the mobile communication system is operational, the operator has to monitor the traffic continuously to check whether calls do not materialize because of network congestion. In case of network congestion, capacities have to be enhanced.

First Generation Wireless Networks

In the 1980s, wireless networks for mobile communications systems were deployed in various countries. These wireless networks were based on proprietary protocols developed by various equipment manufacturers. These systems were known as First Generation (1G) systems. Some of the 1G systems deployed in North America and Europe were:

- ◆ Nordic Mobile Telephony (NMT 450) system operating in 450 MHz band since 1981
- ◆ Advanced Mobile Phone System (AMPS) operating in 800/900 MHz band since 1983
- ◆ Total Access Communication System (TACS) operational since 1985
- ◆ Nordic Mobile Telephony (NMT 900) operating in 900 MHz band since 1986

The drawbacks of these analog cellular systems are: low calling capacity (about 55 calls/cell), limited spectrum, poor data communication support, and minimal privacy. These systems were hardly used for data communication. These systems are no longer operational in North America and Europe, but are in operation in some developing countries in Africa and Asia.

Second Generation Wireless Networks

To overcome the problems associated with the 1G systems, digital systems were developed. These systems, which are presently operational, are known as second generation (2G) wireless networks. The 2G networks can be broadly categorized as Time Division Multiple Access (TDMA) systems and Code Division Multiple Access (CDMA) systems. In TDMA systems, multiple subscribers share a radio channel. Each subscriber is allocated a time slot, in which data is transmitted. In CDMA systems, the entire system bandwidth is made available to each user, but a large bandwidth is required to transmit information and all users can transmit simultaneously. To avoid interference between the data of the subscribers, each subscriber transforms the data using a special code. The 2G systems that are still operational are:

- ◆ Interim Standard (IS) 54/136-based systems operational in North America, which use TDMA technology.
- ◆ IS 95A-based systems operational in North America, which use CDMA technology.
- ◆ Global System for Mobile Communication (GSM) systems, which use TDMA technology. GSM systems are operational in Europe, Asia, and Africa.
- ◆ Personal Digital Cellular (PDC) system and Personal Handy System (PHS) are operational in Japan.

Out of all the previous systems, the GSM system has the largest installation base. The next (or third) generation (3G) systems evolve from the GSM systems in most of the countries.

Global System for Mobile Communications

To facilitate roaming from one country to another within Europe while using the same mobile terminal, in 1983 the European Telecommunications Standards Institute (ETSI) formed the *Groupe Speciale Mobile* (GSM); this body was formed to develop standards for mobile communication systems. Because the standard has been adapted widely by many countries in Asia, Africa, and Middle East in addition to Europe, the acronym GSM now means Global System for Mobile Communications.

The Memorandum of Understanding for GSM was signed by 17 European operators and manufacturers in 1987. The first trial version of GSM was developed in 1991, and the commercial systems were launched in 1992. During the initial days of GSM, many were skeptical about its commercial viability because of the complex protocol architecture. The acronym GSM used to be jokingly referred to as “God, Send Mobiles” because of the complexity involved in developing handsets. Thanks to advances in microelectronics, GSM handsets now go into pockets.

Prominent features of GSM

The most prominent features of GSM are:

- ◆ GSM is based on digital technology; thus security can be easily built into the system and GSM has all the advantages of the digital communication systems, such as better noise immunity and better data capability.
- ◆ Because the interfaces are standardized, network elements manufactured by different equipment vendors can work with one another, thereby paving way for competition; both network operator and the subscriber benefit from this.
- ◆ A higher calling capacity per cell (about 125 calls per cell) as compared to analog systems (about 55 calls per cell)
- ◆ Support for international roaming
- ◆ In addition to voice services, data services are also supported.

GSM specifications

The broad specifications of the GSM system are as follows:

- ◆ **Frequency band:** 900 MHz band (890 – 915 MHz for uplink and 935 – 960 MHz for downlink). As the 900 MHz band got congested, 1800 MHz band has been allocated with 1710 – 1785 MHz for uplink and 1805 – 1880 MHz for downlink. The systems operating in 1800 MHz band are referred to as DCS 1800 (Digital Cellular System 1800).
- ◆ **Duplex distance** (distance between uplink and downlink frequencies): 45 MHz
- ◆ **Channel spacing** (between adjacent carrier frequencies): 200 kHz
- ◆ **Modulation:** Gaussian Minimum Shift Keying (GMSK). The GMSK is a special form of Frequency Shift Keying (FSK). 1s and 0s are represented by shifting the RF carrier plus or minus 67.708 kHz. FSK modulation, where the bit rate is exactly four times the frequency shift, is called the Minimum Shift Keying (MSK). As the modulation spectrum is reduced by applying a Gaussian pre-modulation filter to avoid spreading of energy into adjacent channels, the modulation is called Gaussian MSK (GMSK).
- ◆ **Transmit data rate (over the air bit rate):** 270.833 Kbps, which is exactly four times the RF frequency shift.
- ◆ **Access method:** Time Division Multiple Access (TDMA) with eight time slots. In TDMA system, the same frequency is shared by eight subscribers. Each subscriber is allocated a small time slot during which he can transmit his data. In this time slot he will send his data and then wait for the next time slot, which he will get after the seven others finish transmitting their data in their time slots. In other words, each subscriber will transmit his data in bursts, in time slots which he gets periodically. The subscriber gets the time slot again after the others finish their transmissions.
- ◆ **Speech coding:** To conserve radio spectrum, speech is not transmitted at 64 Kbps data rate (as in normal telephone networks) but at 13 Kbps. Because of low data rate, the quality of the speech is low compared to the voice quality in a normal telephone network.
- ◆ **Signaling:** Unlike telephone networks, where the signaling information (digits dialed, various tones, such as ring-back tone, and so forth) is sent in the same channel as the voice, in GSM a separate signaling network is used for carrying signaling information. So, the radio channels are

used only for carrying out voice traffic, hence the radio spectrum is used efficiently. The signaling used in GSM is called Signaling System 7 (SS7), which is based on standards developed by the International Telecommunications Union-Telecommunications Sector (ITU-T).

GSM services

GSM services are divided into telephony services (referred to as teleservices) and data services (referred to as bearer services). In addition to the normal telephony services, the following services are also supported:

- ◆ Facsimile (fax) transmission through a special interface provided to the handsets.
- ◆ Short Message Service (SMS) to transmit a maximum of 160 alphanumeric characters. If the handset is turned off or is out of the coverage area, the message will be stored in a message center and sent to the handset after it is turned on or when it is within the coverage area.
- ◆ Cell broadcast can transmit maximum of 93 characters to all the handsets in a particular cell. This service is used to transmit information regarding traffic congestion, accident information, and so forth.
- ◆ Voice mail
- ◆ Fax mail

The GSM system also supports the following supplementary services:

- ◆ Call forwarding, to forward a call to another mobile handset or a land line
- ◆ Blocking outgoing calls
- ◆ Blocking incoming calls. All incoming or only incoming calls when roaming outside an operator's region can be blocked.
- ◆ Advice of charge, which gives an estimate of the call charges based on time
- ◆ Call hold, to interrupt a call and then reestablish it again
- ◆ Call waiting, to notify an incoming call when a conversation is in progress
- ◆ Multi-party service to provide conferencing facility
- ◆ Calling Line Identification Presentation (CLIP) to display the telephone number of the calling party
- ◆ Closed User Group (CUG), which emulates the function of a Private Branch Exchange (PBX). A pre-defined group of mobile handsets form the equivalent of PBX.

GSM system architecture

The GSM architecture is shown in Figure 10-2.

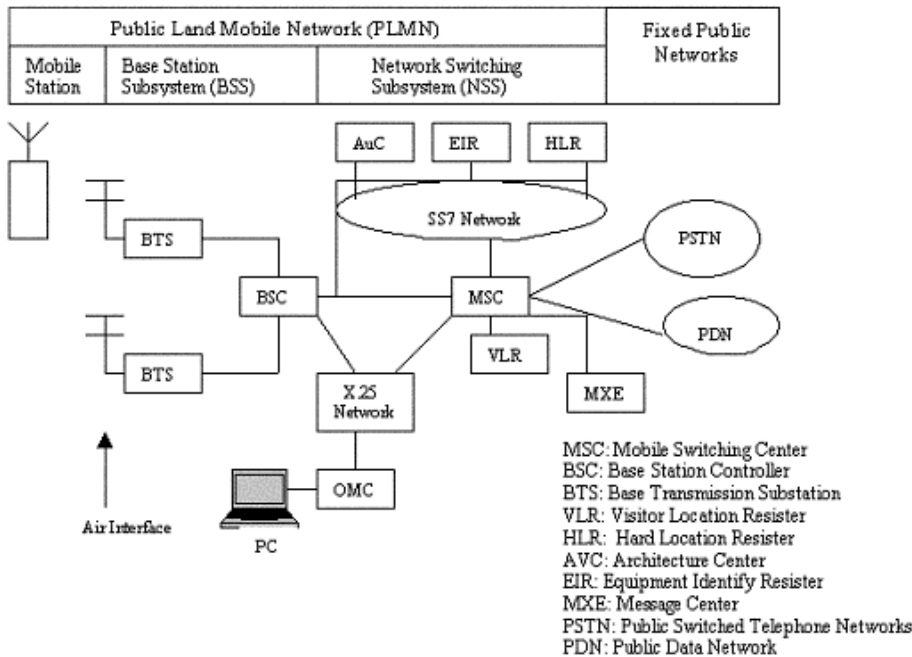


Figure 10-2: GSM architecture

A mobile communications service provider operates in a given geographic region. The mobile network of the entire region is known as Public Land Mobile Network (PLMN). The PLMN is in the administrative control of one operator. The PLMN consists of mobile stations (MS), Base Station Subsystems (BSS), and Network Switching Subsystem (NSS). The MS can be hand-held or car-mounted. The BSS consists of Base Station Controller (BSC) and Base Transceiver Subsystem (BTS). The MSC consists of Mobile Switching Center (MSC), Home Location Register (HLR), Equipment Identity Register (EIR), Authentication Center (AuC), and Visitor Location Register (VLR). In addition to these elements, there is also Operation and Maintenance Center (OMC), which provides the man-machine interface to carry out administrative functionality, such as subscriber management, network management, billing, and so forth.

The PLMN is connected to the Public Switched Telephone Network (PSTN) or a Public Data Network (PDN). The PSTN is the technical word for the telephone network that we all use; the PDN is a data network, such as the Internet. The functions of each element of the GSM system are described in the following section.

Also, known as mobile handset, or hand phone, this is the subscriber terminal. Nowadays, mobile terminals come with many features, such as voice dialing, whereby one can use his voice to dial a number, and powerful batteries that provide at least six hours of talk time and four to five days of standby time. The power transmitted by the MS is in the range 0.8 watt – 20 watts.

A unique mobile phone number identifies the MS. Each MS is also uniquely identified by IMSI (International Mobile Subscriber Identity). The MS contains an SIM (Subscriber Identity Module). The SIM is a smart card inserted in the handset. A Personal Identity Number (PIN) protects the SIM. The PIN is checked locally and not transmitted over the radio link. SIM contains IMSI. To identify handset hardware uniquely, manufacturers use IMEI (International Mobile Equipment Identity) to provide a number to the handset.

The BSS acts as a relay between the NSS and the mobile stations. The BSS consists of BSC and BTSs. The service area is arranged into cells, and each cell has a BTS. Each cell can vary from 350 meters to 35

kilometers depending on the terrain and the subscriber density. Multiple BSSs can be controlled by one BSC.

The BSC handles radio management functions. The BSC arranges new radio link connections to the mobile stations when handover is required. It is connected to the MSC through land lines, normally two Mbps links. It uses standard Pulse Code Modulation (PCM), a coding technique through which speech is coded at 64 Kbps data rate to carry voice in digital format between the BSC and MSC. To reduce radio bandwidth, the GSM uses low bit rate coding of speech at 13 Kbps. BSC does the transcoding — conversion of 13 Kbps speech to PCM and vice versa. Each BSC controls a number of BTSs, typically up to 40.

The BTS is the radio interface between the MS and the BSC. Communication between the MS and the BTS occurs through one channel consisting of a pair of frequencies — one for uplink and one for downlink.

The frequency allocation for GSM in the 900 MHz band is depicted in Figure 10-3. Carriers are separated by 200 kHz. If channel 2 is given to a particular cell, two frequencies are allocated — one uplink frequency and one downlink frequency. The maximum power transmitted by the BTS is in the range 0.25 watt – 320 watts. The BTS uses TDMA for multiple access with eight slots per channel. The TDMA frame format is shown in Figure 10-4. This is a very simplified format — TDMA slots are also used to carry out all the signaling between the BSC and MS.

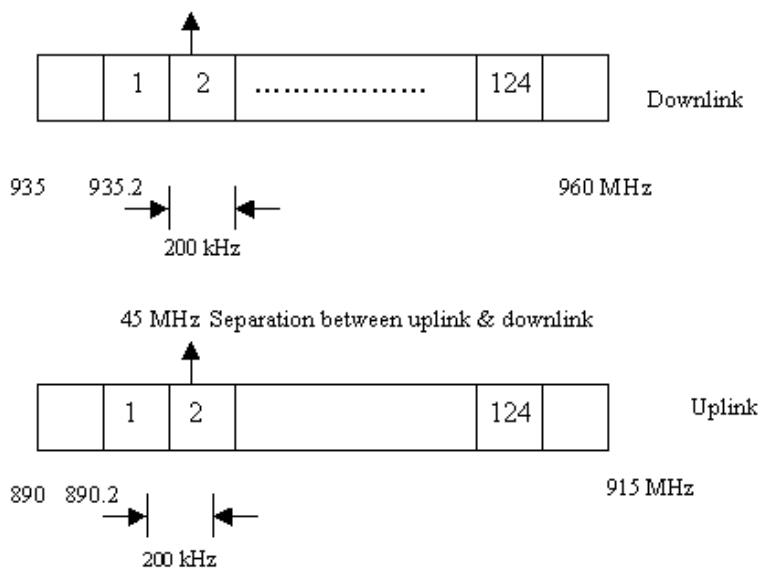


Figure 10-3: Frequency allocation for GSM

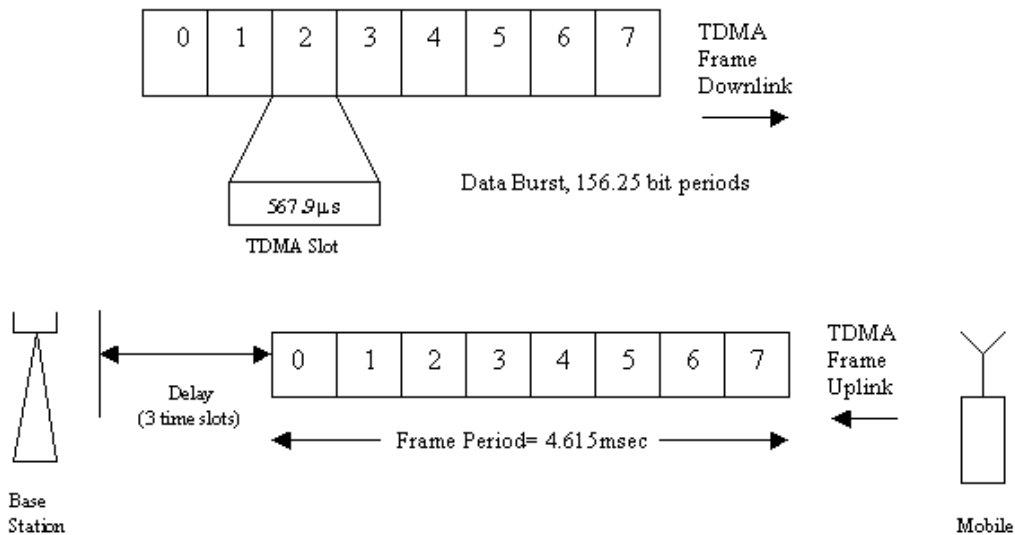


Figure 10-4: TDMA Frame Format in GSM

Each data bit is of 3.692 microseconds duration. Each time slot has a time period equal to 156.25 of the data bits. There are eight time slots per frame, thus each frame period is 4.615 milliseconds. Twenty-six or 51 frames are grouped together to make a multi-frame. A super-frame consists of 51 or 26 multi-frames. These complex frame and multiframe structures are used to transmit control information, to carry out synchronization, and of course, to carry speech data.

The TCH (Traffic channel) carries the bi-directional speech data between the mobile station and the base station. Each base station produces a BCH (Broadcast Channel), which acts as a beacon signal to find service and decode network information. The BCH occupies time slot zero. Each cell is given a number of frequency pairs (channels) denoted by ARFCN (Absolute Radio Frequency Channel Numbers). If a cell has one ARFCN, there will be one BCH and seven time slots for TCH. If there are two ARFCNs in one cell, there will be one BCH and 15 time slots for the TCH.

The HLR is a centralized database to manage the subscriber data. It is a stand-alone system connected to the GSM network subsystems with Signaling System No. 7 (SS7) signaling. This database contains

- ◆ Subscriber information
- ◆ Subscriber rights and privileges (what types of calls are permitted)
- ◆ Location information
- ◆ Activity status

The HLR retains permanently the location of any subscriber. When an MS receives a call, the HLR is consulted and the database translates the mobile phone number to an IMSI number. The HLR reroutes incoming calls to the MSC target or another telephone number when call forwarding is requested.

The AuC contains security functions such as IMSI, encryption key, and an algorithm to be used for encryption. The AuC provides the data to verify the identity of each user and to provide conversation/data confidentiality. The HLR/AuC administered by the Man Machine Interface (MMI) is derived from the OMC.

The VLR contains information about all the mobile subscribers currently located in the MSC service area. The VLR is generally integrated into MSC. When a mobile station roams into a new MSC service

area, the VLR connected to that MSC gets the data about the mobile station from the HLR and stores it. The VLR is responsible for the information about the current location of the user.

The EIR contains information about the mobile equipment. Each MS is uniquely identified by IMEI. If a mobile handset is lost, the subscriber informs the customer-support center and this information is stored in the EIR. If the lost mobile is used for making a call, the EIR will not permit the call. As EIR and AuC both provide security, EIR and AuC can be combined into one machine.

The MSC provides the complete switching functionality for the entire network; hence all call control functions are built in the MSC. To facilitate normal telephone subscribers of PSTN to receive/make calls to the mobile subscribers, the MSC is connected to the PSTN through trunk lines. Similarly, Public Data Networks (PDNs) can be connected to the MSC. The MSC is also connected to the OMC through which the configuration of the network, entry, and modification of the subscriber data, traffic analysis, billing, and other network management functions can be carried out.

The OMC is used to carry out network management activities, such as fault diagnosis of various network elements, traffic analysis, billing, performance management, configuration management (such as adding a new BSC, cell splitting, and so on), as well as managing subscriber information.

The communication between the MSC and the databases (HLR, EIR, AuC) is through an SS7 network, because only signaling information is exchanged between these entities. The communication between the OMC and the MSC/BSC is through a packet-switching network based on X.25 standards.

The GSM system can contain the following network elements.

Message Center is a node that provides voice, data, and fax messaging. It handles the Short Messaging Service (SMS), cell broadcast, voice mail, fax mail, and e-mail messaging. Separate servers are required to handle these messaging systems, which are connected to the MSC.

When a PLMN contains a number of MSCs, one MSC is designated as a Gateway MSC to interconnect with other networks such as PSTN and PDN. If the PLMN contains only one MSC, that MSC itself can act as a Gateway MSC.

GSM network areas

In a GSM network, the following areas are defined:

- ◆ **Cell:** Cell is the basic service area: one BTS covers one cell. Each cell is given a Cell Global Identity (CGI), a number that uniquely identifies the cell.
- ◆ **Location Area:** A group of cells form a Location Area. This is the area that is paged when a subscriber gets an incoming call. Each Location Area is assigned a Location Area Identity (LAI). Each Location Area is served by one or more BSCs.
- ◆ **MSC/VLR Service Area:** The area covered by one MSC is called the MSC/VLR service area.
- ◆ **PLMN:** The area covered by one network operator is called PLMN. A PLMN can contain one or more MSCs.

GSM operation

The operation of the GSM system can be understood by studying the sequence of events that takes place when a call is initiated from the Mobile Station.

When a mobile subscriber makes a call to a PSTN telephone subscriber, the following sequence of events takes place:

1. The MSC/VLR receives the message of a call request.
2. The MSC/VLR checks if the mobile station is authorized to access the network. If so, the mobile station is activated. If the mobile station is not authorized, service will be denied.

3. MSC/VLR analyzes the number and initiates a call setup with the PSTN.
4. MSC/VLR asks the corresponding BSC to allocate a traffic channel (a radio channel and a time slot).
5. The BSC allocates the traffic channel and passes the information to the mobile station.
6. The called party answers the call and the conversation takes place.
7. The mobile station keeps on taking measurements of the radio channels in the present cell and neighboring cells and passes the information to the BSC. The BSC decides if handover is required; if so, a new traffic channel is allocated to the mobile station and the handover is performed. If handover is not required, the mobile station continues to transmit in the same frequency.

Call to a mobile station

When a PSTN subscriber calls a mobile station, the sequence of events is as follows:

1. The Gateway MSC receives the call and queries the HLR for the information needed to route the call to the serving MSC/VLR.
2. The GMSC routes the call to the MSC/VLR.
3. The MSC checks the VLR for the location area of the MS.
4. The MSC contacts the MS via the BSC through a broadcast message, that is, through a paging request.
5. The MS responds to the page request.
6. The BSC allocates a traffic channel and sends a message to the MS to tune to the channel. The MS generates a ringing signal and, after the subscriber answers, the speech connection is established.
7. Handover, if required, takes place, as discussed in the earlier case.

Note, that the MS codes the speech at 13 Kbps for transmission over the radio channel in the given time slot. The BSC converts (or transcodes) the speech to 64 Kbps and sends it over a land link or radio link to the MSC. The MSC then forwards the speech data to the PSTN. In the reverse direction, the speech is received at 64 Kbps rate at the BSC and the BSC does the transcoding to 13 Kbps for radio transmission.

In its original form, GSM supports 9.6 Kbps data, which can be transmitted in one TDMA time slot. Over the last few years, many enhancements were done to the GSM standards (GSM Phase 2 and GSM Phase 2+) to provide higher data rates for data applications. Through these enhancements, the GSM systems can evolve to 2.5G systems, which we study later in the chapter.

Wireless Internet Access through 2G Systems

At present, most of us access the Internet through our desktops or our corporate LANs through wired connections (dial-up or leased lines). Our desktops have high processing capability, large primary storage (Random Access Memory), and secondary storage. This enables us to run a browser, such as Internet Explorer or Netscape Navigator, which requires huge resources. The monitors are capable of displaying high-resolution color graphics. Also, the wired connection can support high data rates, anywhere between 64 Kbps to 2 Mbps. Hence, accessing the Internet is a lot of fun, with a lot of multimedia content, fast file downloads, and fast and easy navigation.

The only drawback of this access method is that we are restricted to being in one place — there is no mobility. If we could access Internet services through wireless devices, such as mobile phones, laptops, or palmtops, it would be of immense value. However, to achieve this objective, we face lots of challenges — the mobile phone has a small display and small memory, mobile networks support very low data rates, and long delays are common. However, the need for mobile access to the Internet is growing rapidly. In the future, the number of wireless devices will probably exceed the number of wired devices accessing the Internet. This market demand is paving the way for exciting developments in wireless Internet services and technologies.

If widespread wireless access to the Internet becomes a reality, users could have mobile access to the Internet anytime from anywhere. However, providing Internet services over these mobile devices is a pretty challenging task, as we see in the next section

Challenges in wireless access to the Internet

Wireless access to the Internet (Web service, in particular) presents many problems at present. These problems are:

- ◆ There are a variety of protocols for the wireless systems, such as the TDMA, CDMA, GSM, PDC, and so on. So, the protocols for wireless access to the Internet need to be independent of the underlying cellular network protocols.
- ◆ Present wireless networks support very low data rates, ranging from 300 bps to 14.4 Kbps. Thus, accessing a Web page with multimedia components would take ages! As compared to the wired networks, the delay is higher in wireless networks. The time taken for a data packet to reach the server and be acknowledgement (known as the round trip delay) is very high in wireless networks.
- ◆ Wireless devices (mobile phones, pagers, and so forth) have limited capabilities:
 - Small screen, generally 4 lines with 8 to 12 characters per line
 - Screen with low resolution and no support for color
 - Low power
 - Keypad with a very limited functionality
- ◆ Wireless devices vary widely — different platform technologies, different memory sizes, and so on.

Due to these problems, developing applications and protocols for wireless access of Internet services is a real technical challenge. In the late 1990s, the software industry created a lot of hype about wireless access to the Internet, but it has since realized that lots of work needs to be done to enhance the speeds of wireless networks, develop efficient protocols, and also create useful and appealing applications. To provide wireless Internet access through 2G systems, the Wireless Application Protocol (WAP) has been developed. The need for WAP arose because a “light-weight” protocol is needed — a protocol with less overhead as compared to the Transmission Control Protocol/Internet Protocol (TCP/IP) used in the Internet. As we have seen in earlier chapters, the WAP protocol stack can be used on any type of digital wireless networks, such as the TDMA, CDMA, PDC, GSM. The mobile device (the handset) has to run a small browser (micro-browser), which interprets the WML content and presents it to the user.

WAP limitations and future work

WAP is the first step in the process of standardization for achieving wireless Internet access. Many critics of the WAP have already dismissed it as technology that makes you “Wait and Pay!” In its present form, WAP does have some limitations:

- ◆ For WAP to be deployed on a large scale, WAP content needs to be available. At present, portals with WAP content are very few. Unless there is a good number of mobile portals, the WAP service will not pick up; unless the WAP service picks up, the number of mobile portals will not increase.
- ◆ WAP is a connection-oriented protocol, meaning that a connection must be established with the server, and then the data can be transferred in a session.
- ◆ WAP is still considered a heavy-weight protocol because of the overhead involved in sending packets of data.
- ◆ WAP enabled mobile phones are costly at present.
- ◆ WAP content needs to be developed in WML and WMLScript, whereas most of the Internet content presently is in HTML. The conversion from HTML to WML using automated tools is very inefficient as many tags supported in HTML are not available in WML.

- ◆ When Short Messaging Service (SMS) is used as the bearer for WAP, the message length is limited to only 160 characters.

WAP has been revised by the WAP Forum, and WAP 2.0 specifications have been released in July 2001 to overcome these limitations. In WAP 2.0, XHTML (Extensible HyperText Markup Language) will be used for content development. As XHTML is derived from the XML and is very similar to the HTML, the content-related problems are likely to be solved. Compatibility with WML is provided, and hence the existing WAP applications will continue to run without any changes.

i-mode

As the rest of the world has been struggling to provide wireless Internet access through WAP on 2G networks, in Japan, wireless Internet access has become very popular; this has been made possible through the i-mode technology of Japan's largest cellular operator — NTT DoCoMo. i-mode became an extremely popular service in a short span of time because of the content, efficient protocols used, and high data rates, in addition to the low price the subscribers pay for accessing the service. i-mode is a proprietary protocol, and the tool kits are available with Japanese language support only.

The system architecture for offering i-mode mobile Internet services is shown in Figure 10-5. An i-mode server is connected to the Internet and to the DoCoMo packet network, based on PDC (Personal Digital Cellular) standard of Japan. The communication between the Internet and the i-mode server is through TCP/IP and HTTP protocols.

The i-mode terminal has a micro-browser to access Internet services. Information providers, such as banks can connect to the i-mode server to provide services, such as mobile banking. Compact HTML (cHTML) is used to transfer the data from the i-mode server to the mobile terminal, and the micro-browser interprets the cHTML content and presents it to the user. cHTML is derived from HTML to provide the content on the wireless network.

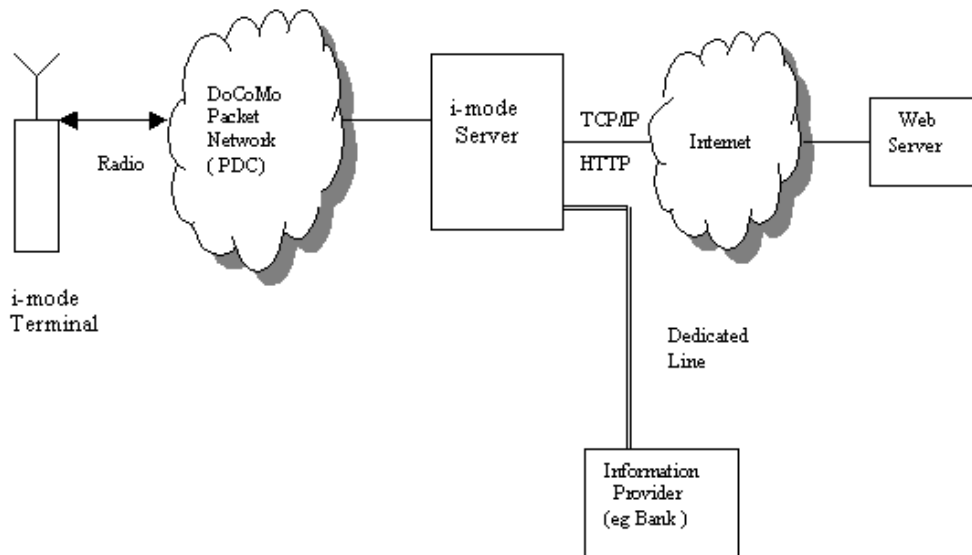


Figure 10-5: i-mode system architecture

i-mode supports a number of services, such as:

- ◆ Short e-mail messages
- ◆ Weather information

- ◆ Mobile banking
- ◆ Travel information including traffic information and maps of tourist spots
- ◆ Shopping information
- ◆ Part-time job listings
- ◆ Games and horoscopes
- ◆ Navigation information for drivers
- ◆ Graphic and comic strip downloads

Because i-mode works on the packet mode and not the circuit-switched mode, it is always connected and is also faster. The tariff is based on the number of packets (one packet = 128 bytes) transmitted.

With 17 million subscribers and 776 information providers as of January 2001, i-mode is the most popular wireless Internet access service in the world. The popularity of i-mode service is due to its fast access, low tariff, and also because of the large content available to the users. The protocols used in i-mode are lightweight, as compared to WAP; thus i-mode is now being considered for deployment in many other countries. NTT DoCoMo is now promoting i-mode in the U.S. and Europe as well. The WAP Forum and NTT DoCoMo are to work jointly to provide efficient protocols and attractive content to the end users.

2G wireless devices

The wireless devices that access the Internet content through 2G networks have limited capability, in terms of processing power (only 8-bit or 16-bit micro-controllers). Other limitations include small black and white displays (2 to 4 lines with 8 to 12 characters), small keypads, which makes typing text difficult; thus the browser that runs on these devices also is of limited capability. Consequently, the services provided to the users also have limitations. High resolution graphics cannot be transmitted, and animation is not possible. The content that can be downloaded on to the handsets should be text-based and very focused, such as stock quotes, weather information, or small text messages, such as astrological predictions, sports information, or news. Composing a message on the handset is also time consuming because the keypad has limited functionality. Navigation is tough because only few soft keys are available on the handset. But still, WAP and i-mode provide good utility in obtaining focused information from the mobile portals.

2G Internet content

Today, most of the Internet content is in HTML format, which cannot be accessed through 2G wireless devices. The content has to be in WML in the case of WAP, or cHTML in the case of i-mode. The precursor to WAP is Handheld Device Markup Language (HDML); content is also available in this language. There are also examples of proprietary content creation approaches, such as Palm's web-clipping. However, many of these languages share the same predicament: content cannot be made available to everyone irrespective of the characteristics of the wireless device.

Presently, "content transcoding" is a big business — it involves transferring the content from one markup language to another. Automatic tools, such as HTML-to-WML conversion utilities are available, but they don't do a good job because WML supports very few tags, and many tags supported in HTML are not supported by WML. Scripting languages, such as WMLScript present more difficulties, because conversion from, say, JavaScript to WMLScript is also extremely difficult.

One of the main reasons for the delayed development of widespread wireless Internet access is this content-related problem. Later in this chapter, we see how the next generation of wireless devices can overcome this problem.

2.5G Wireless Networks

The 2.5G wireless networks, which evolve from the 2G systems, will support data rates in the range 64 to 144 Kbps, and these networks will evolve into 3G systems. The various standards for 2.5G are:

- ◆ High Speed Circuit Switched Data (HSCSD) based on GSM
- ◆ General Packet Radio Service (GPRS) based on GSM
- ◆ IS 95B Systems based on CDMA

HSCSD

The HSCSD system is shown in Figure 10-6. In the GSM network, when a call is established, one time slot is allocated to the subscriber. In one time slot, we can push 9.6 Kbps data. The HSCSD is based on multi-slotting — a user is allocated two to eight time slots. If all eight time slots are allocated to one user, it gives 8×9.6 Kbps, that is, 76.8 Kbps of theoretical data rate. Using this approach, download speeds up to 43.2 Kbps have been achieved in practical systems. This data rate is sufficient to support good resolution graphics, animation, and low speed video. The HSCSD is an add-on feature to the GSM network with software only for upgrades. This is a connection-oriented service, such as a normal telephone call, where a data call has to be established for Internet access. Because the protocols used in the GSM network and the PSTN/Internet are different, an Inter Working Function (IWF) is required, as shown in Figure 10-6.

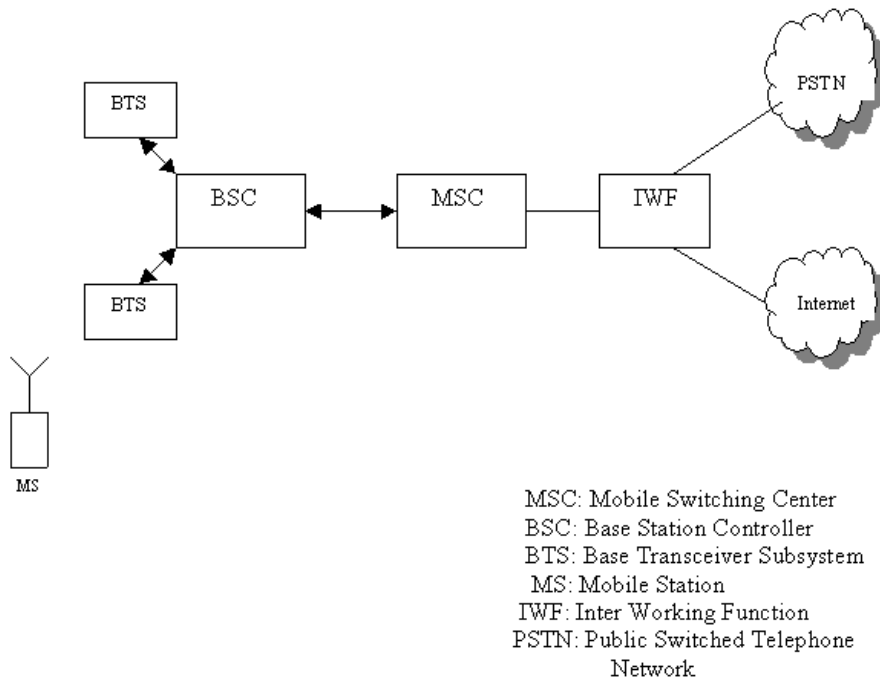


Figure 10-6: High speed circuit switchboard data service

The IWF carries out the necessary protocol translation between PSTN/Internet and the GSM. Because the HSCSD provides a connection-oriented service, it is well suited for applications, such as video conferencing, where a circuit has to be established before the conferencing takes place. Of course, the quality of the video will not be very high because of the limitation of the data rate.

General Packet Radio Service (GPRS)

GPRS uses the GSM infrastructure to provide packet switching, such as is needed for the Internet. Similar to the HSCSD, multiple time slots are allotted to the user, but the data is transmitted using the Internet Protocol (IP). A mobile device connected to the GPRS is like a node on the LAN, or a node connected to the Internet through the Digital Subscriber Line (DSL) — it is always online and there is no need to make a call. The mobile device is assigned an IP address whenever it is connected to the Internet for carrying out any data transfer. As in the case of the HSCSD, data rates up to 78.8 Kbps can be achieved in the GPRS. Using efficient coding techniques, we can push data up to 14.4 Kbps in one slot, and the data rates can be increased to 172 Kbps. Video transmission using MPEG-4 standard (a low bit rate video coding standard developed by Motion Picture Experts Group) has been demonstrated over the GPRS network.

The block diagram of the GPRS system is shown in Figure 10-7. Two network elements — Gateway GPRS Support Node (GGSN) and Serving GPRS Support Node (SGSN) — are added to an existing GSM infrastructure. GGSN is the gateway between a GPRS wireless network and the IP-based data network, such as the Internet. GGSN carries out the necessary protocol conversion between the packet network (for example, Internet) and the GPRS network. The GGSN also carries out the address conversion because the addressing formats of the mobile network and the data network will be different. The Serving GPRS Support Node (SGSN) performs the authentication of the users and keeps track of the mobile users. Location registers in the SGSN (HLR and VLR) store the location information, such as current cell, as well as user profiles of the subscribers registered with the GPRS. It is responsible for mobility management, i.e., attaching and detaching the mobile devices to the network, and managing the links for data communication between the Internet and the mobile devices. The SGSN also routes the packets to/from the mobile device to the GGSN by establishing the links.

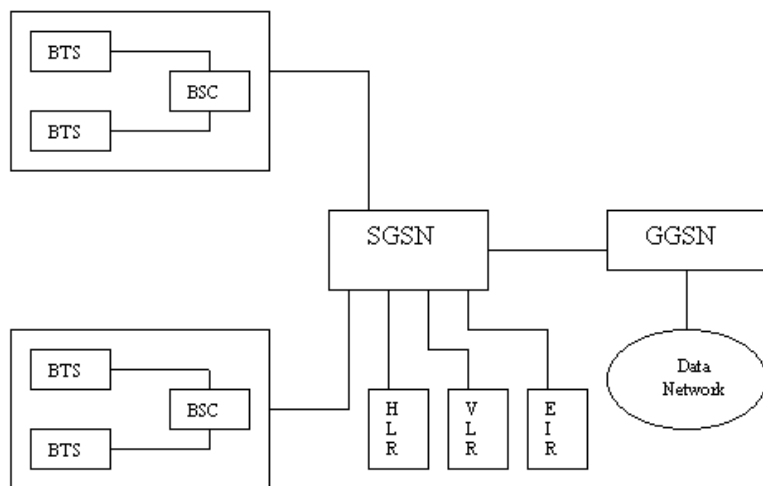


Figure 10-7: GPRS architecture

CDMA 95B

The CDMA 95B networks will evolve from the networks based on the IS 95A standards. The IS 95A-based networks evolved from the AMPS networks and support data rates up to 14.4 Kbps. The IS 95B-based networks support 76.8/115.2 Kbps using packet switching. Qualcomm Corporation, which pioneered the CDMA technology, is the most prominent manufacturer of these CDMA-based systems.

All these 2.5G systems are only a stepping-stone to the 3G systems. The data rates supported by 2.5G systems can be used for graphics and audio applications, but for high resolution video streaming applications, such as video mail and video conferencing, these data rates are not sufficient.

Third Generation Wireless Networks

The 3G wireless networks will be capable of supporting very high data rates in the range of 384 Kbps – 2.048 Mbps to provide multimedia services to mobile users. With these data rates, applications, such as two-way video conferencing, high quality audio (music) downloading, high-resolution graphics, and animation can be supported.

To develop high-speed wireless networks, ITU initiated the standardization process in 1986. The Future Public Land Mobile Telecommunication Systems (FPLMTS) working group developed the initial set of standards. FPLMTS is now known as IMT 2000 (International Mobile Telecommunications in 2000). UMTS (Universal Mobile Telecommunications Systems) is the European version of IMT 2000. The work was challenging considering the many different technologies (some of which are in vogue, some of which are not), huge infrastructures put in place by different operators, lack of availability of spectrum in the same bands in different countries, and an increasing demand for higher and higher data rates by the end users. Even today, no common spectrum is available worldwide for 3G systems, nor can a single technology be standardized.

The broad objectives of 3G systems are:

- ◆ Support 2 Mbps for handheld devices, 384 Kbps for walking mobile devices, and 144 Kbps for car-borne mobile devices
- ◆ Support for global roaming
- ◆ The 3G systems should work in all radio environments: urban areas, suburban areas, hilly and mountainous regions, and indoor environments. To achieve this, the cell size may vary considerably. In addition to the regular cells in the GSM networks, the 3G systems should support micro-cells (cells of a few meters radius) and pico-cells (cells of a few feet radius).
- ◆ Asymmetric and symmetric services should be supported. In asymmetric services, the uplink (from handset to base station) data rates can be lower and downlink (base station to handset) data rates can be higher.

The following services should be supported:

- ◆ Computer data with Internet access, mail, file transfer, mobile computing
- ◆ Telecom services, such as telephony, video telephony, video and audio conferencing
- ◆ Audio/video on demand, tele-shopping, TV and radio broadcast.

3G Standardization Activities

Based on a call for proposals by the ITU Radio Communications Standardization Sector (ITU-R) Task Group 8/1, various proposals were submitted by the following organizations:

- ◆ ETSI Special Mobile Group (SMG) of Europe
- ◆ Research Institute of Telecommunications Transmission (RITT) of China
- ◆ Association of Radio Industry and Business (ARIB) and Telecommunications Technology Committee of Japan
- ◆ Telecommunication Technologies Association (TTA) of Korea
- ◆ Telecommunications Industries Association (TIA) and T1 of USA

The objective was to develop a single international standard IMT2000, which facilitates global roaming. The proposals submitted by the previously mentioned organizations were nowhere near achieving this objective — they were all based on different technologies. Subsequently, two international bodies were established; the 3GPP (3G Partnership Program) and the 3GPP2 to harmonize and develop standards for 3G systems based on the above proposals. Harmonization is a tall order — the access scheme (TDMA versus CDMA), spectrum, signaling protocols, and such — all differ from system to system. So instead of a single technology, the 3G standards have mainly two types of systems, both based on the CDMA technology. One system is referred to as CDMA2000 and the other as the W-CDMA. Both the systems meet the objective of 3G system data rates: 144 Kbps for high mobility users, 384 Kbps for limited mobility users, and 2 Mbps for static users.

The evolution of 3G is shown in Figure 10-8. The GSM network will be upgraded to the GPRS, which in turn will be upgraded to the W-CDMA network. However, a new spectrum is required for the 3G services. The IS 95A network based on CDMA will be upgraded to IS 95B network, which in turn will be upgraded to CDMA2000 network in 3G. The existing spectrum will be used for providing the 3G services. The PDC system of Japan will be upgraded to the W-CDMA-based system for providing the 3G services.

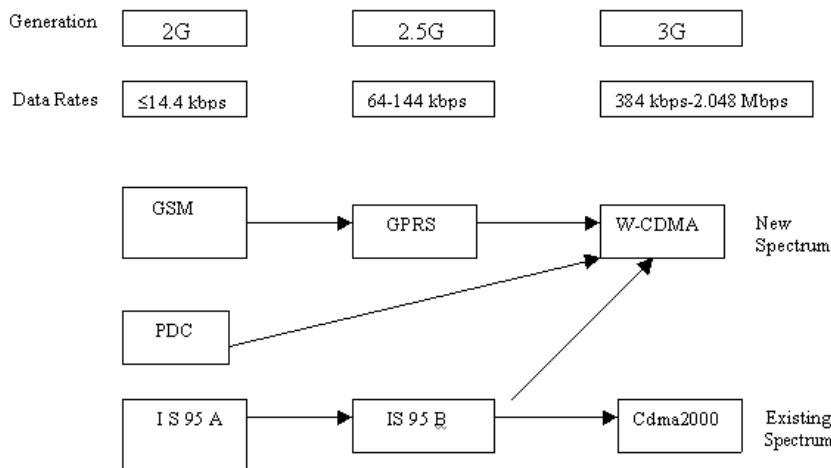


Figure 10-8: Evolution of 3G

CDMA2000

The CDMA2000 network will evolve from the existing CDMA systems, such as IS 95A and IS 95B, which are also based on CDMA. User data rates ranging from 9.6 Kbps to 2 Mbps are supported in this system. The signaling protocols used in the network are same as those used in the IS 95 standards viz., IS 41. This system uses the same frequency band as the IS 95-based systems and thus, the existing spectrum is used. The RF channel bandwidths required for this system are 1.25, 5, 10, 15 and 20 MHz.

W-CDMA

This network will evolve from the GSM networks. However, new frequency bands are proposed for this network and hence new spectrum needs to be allocated. Though there is a backward compatibility to the GSM network, if a user of the W-CDMA network roams into an old GSM network, a dual-band handset will be needed because of the differences in the frequency of operation. Signaling protocols used in the GSM network are also used in W-CDMA networks. The RF channel bandwidths required for this system are 1.25, 5, 10, and 20 MHz.

3G Spectrum Needs

To achieve global roaming, ideally all the 3G systems in different countries should use the same frequency bands. However, this is not possible because 3G systems evolve from the existing systems, which use different frequency bands in different countries. Thus, international roaming is still a problem. For the European region, the frequency bands 1900-1980 MHz, 2010-2025 MHz, and 2100-2170 MHz have been allocated for IMT 2000. The existing Personal Communication Services (PCS) bands in the region of 1850-1990 MHz will be used for 3G networks in the USA. Unfortunately, no common spectrum is available worldwide for 3G mobile services. The 3GPP and the 3GPP2 have been merged together to form the Global 3G (G3G) forum. This forum worked out the modalities to achieve international roaming by providing the necessary network elements for protocol conversions, handover from one type of network to another type of network, and for interfacing to the legacy wireless networks, such as the existing 2G networks.

3G Wireless Devices

To support the applications making use of the high data rates, wireless mobile devices should have the following characteristics:

- ♦ **High performance:** The mobile device should have high processing capability, at least 10 times the processing capability of today's mobile devices. As users would like to have the capabilities, such as voice dialing through speech recognition, the processor should be capable of handling the input/output operations quickly. In addition, the devices should have high primary memory (RAM) and also secondary storage devices.
- ♦ **Low power consumption:** As the processing power increases, the battery drain also goes up. Better battery technologies are required to be incorporated in the 3G devices.
- ♦ **Small size and weight:** Size and weight continue to be the most important features. Small size and less weight with high performance can be achieved only by large scale integration of the electronic circuitry using the system-on-a-chip concept. This will be the greatest challenge in developing 3G devices.
- ♦ **Integrated peripherals:** To support applications such as video, a video camera needs to be integrated into the mobile device. Also, a high resolution color graphics display is needed. The input devices should be user-friendly. Also a full-fledged keyboard must be integrated. Imagine a multi-media PC that can go into your pocket — that is the 3G mobile device!
- ♦ **Operating system:** To take care of input/output management, memory management, and process management, an operating system needs to be running on the mobile device. Mobile operating systems, such as Win CE, Palm OS, OS/9, and Java OS will be ported on the mobile devices. To run various applications above the operating system, Sun Microsystems developed J2ME (Java 2 Micro Edition), which has a virtual machine, called the KVM (Kilobytes Virtual Machine). The KVM occupies very small memory (less than 256 KB) and, hence, it is ideally suited for mobile devices. The Java compatible software applications can be downloaded from the Internet on to the mobile device just the way we download the applications from the Internet on to our desktops today.

However, it is naïve to assume that all 3G devices will have all the preceding characteristics. A large mobile population continues to use the normal handsets — some will continue to have the WAP-enabled phones and some will have mobile phones, which are compatible only with the GPRS networks. The important point to be noted here is that each of these devices have different processing capabilities and have different platform technologies. All these devices vary in terms of processing power, primary and secondary storage capacity, display size and resolution, battery capacity, input device capability, and so on. Not all devices have full-fledged operating systems residing on them.

Because of the variety of the wireless access devices, the mechanism for accessing the content from the Internet also varies:

- ◆ Mobile phones, two way pagers, and the like will have limited processing capability, limited memory, and very small displays. They need to access the content using protocols, such as WAP and the content has to be in a format, such as the WML.
- ◆ Handheld computers will have higher processing power, more memory, and a color display with larger size. They can run a mobile operating system, such as WinCE, PalmOS or JavaOS. They can run a browser with better features than a micro-browser and interpret mark up languages, such as XHTML or XML. Alternatively, these devices can run a KVM that can download Java code, and hence they can run Java-based applications.
- ◆ Laptop computers will be Able to run a full-fledged desktop operating system and access Internet content just as we access the content from the desktops.
- ◆ Mobile devices that can support high data rates will have a large display, a built-in camera, and video-transmitting functions to support video conferencing.
- ◆ Devices will be capable of handling multi-call services for simultaneous handling of both voice and data services.

Content for 3G Devices

Creating content that can be accessed by mobile devices of different capabilities is going to be the biggest challenge for 3G content developers. Creating content in many markup languages that cater to different devices will be a gigantic task. For some time to come, transcoding of content, to convert content of one language to another, will continue to be a big business. However, in the long run, it is likely that the content creation will be based on languages derived from XML, such as XHTML, and applications will be based on Java. With the advent of J2ME and Sun Microsystems' Wireless Tool Kit, the 3G programming roadmap isn't so blurry. However, it needs to be mentioned that for 3G to succeed, the key lies with content creation. The end user is not as much interested in the CDMA or TDMA as in what applications are supported and how effectively the 3G networks can be used for daily routine — for shopping, for education, for business, and, of course, for entertainment. In the next section, we discuss the various applications supported by the 3G networks.

3G Applications

In the present 2G networks, Web browsing is very constrained because of the limited capabilities of mobile devices. However, these applications continue to be popular as they provide very focused information and can be accessed quickly. Some applications that are now becoming popular are listed in the following section:.

- ◆ **Mobile e-mail:** To read, reply to, forward and create e-mails, This service is an enhanced version of the Short Messaging Service (SMS) of the cellular mobile systems.
- ◆ **M-Commerce:** To place orders for small items (books, bouquets, and so on), to buy and sell shares, to carry out bank transactions, to book flight, train tickets.
- ◆ **Entertainment services:** To obtain sports information, daily horoscope, weather information, travel information, sports scores, and the like.
- ◆ **News:** To obtain the latest business, political, and sports news.
- ◆ **Electronic business cards:** The information contained on a business card will be available in a mobile device (name, designation, e-mail address, phone numbers). This information can be automatically transferred to another mobile device. Thus, there is no need to exchange paper business cards.
- ◆ **Electronic wallet:** The SIM card in the mobile device can act as a wallet. One can carry out a mobile commerce transaction, but there is no need to give credit card information. The bill amount can be a part of the telephone bill. Alternatively, another smart card can be placed in the mobile phone, which is the electronic wallet.

- ◆ **Advertisements:** Advertisements can be sent to the mobile devices from the server. This is a revenue generating source for the operators and service providers.

In the future, as the capability of mobile devices improves and the access speed increases, multimedia services can be supported. These include features such as animation and real time video. Some typical applications are listed here:

- ◆ **Videophone:** Facilitating seeing the video of the people in conversation
- ◆ **Video conferencing:** Point-to-point or point-to-multipoint conferencing through video and audio
- ◆ **Video mail:** Sending video clippings in the mail messages
- ◆ **Web based learning:** Participating in real-time Web casting of lectures. Web-based learning provides an integrated solution for hearing lectures online or offline. It also provides the capability to create virtual learning communities and discussion groups and provides bibliographic database access.
- ◆ **Telemedicine:** Facilitating remote medical diagnostics. A patient can send the diagnostic reports to a specialist and obtain second opinion. Telemedicine can be effectively used in remote/rural areas where there might be only a paramedic staff available.
- ◆ **Mobile video player:** Downloading a video (movie) while on the move
- ◆ **Advanced car navigation:** Downloading the digital maps while on the move and obtaining navigational information and location-based services

To summarize, whatever we are doing through our desktops, we will be able to do from anywhere, anytime.

Location-Based Services

When a mobile device accesses a server to obtain the content, the server will know the approximate location of the mobile device. Information pertaining to that location can be transferred to the mobile device. On entering a new city in a car, one can get information about the hotels or hospitals in that locality from the server. These location-based services are now catching up very fast. This can be achieved in two ways:

- ◆ By integrating a Global Positioning Satellite (GPS) receiver with the mobile device, a GPS receiver gives the longitude and latitude of the location of the mobile device, which can be transferred to the server to obtain the information pertaining to that location.
- ◆ The mobile system (the BSC of the GSM network) can calculate the approximate distance between the mobile device and the base station. Based on this distance, the BSC will forward the location information of the mobile device to the server. The server will, in turn, send the information pertaining to that location to the mobile device, for instance, the names of the nearby hotels.

To exploit the potential of the location-based services and at the same time to ensure interoperability between mobile positioning systems developed by different organizations, the Location Interoperability Forum (LIF) was founded by Ericsson, Motorola, and Nokia in September 2000.

Location-based services will be of immense use in public safety and emergency services. New exciting personalized services can also be developed using this technology — you can find a friend in a specific locality as soon as you enter a particular location. These services will be made available on both the 2G and the 3G systems and terminals.

A 3G Example: NTT DoCoMo's FOMA

In most countries, the 3G systems will be introduced only in 2002 or later. Japan was the first country to introduce 3G system on experimental basis, in May 2001. The system is based on W-CDMA. The architecture of the system is shown in Figure 10-9. The system supports both packet-switching and

circuit-switching operations. The normal data services, such as e-mail, file transfer, and text chat are carried out through the packet-switching operation. Because these services are similar to the i-mode services being offered today, an i-mode server will be connected to the W-CDMA network, which in turn is connected to the Internet. Various content providers will be connected to the network to provide content — such as normal Web servers, music distribution servers, image distribution servers. For all these data services, which use packet switching, the speeds supported are 64 Kbps for uplink and 384 Kbps for downlink. For applications, such as video conferencing, circuit switching is used at a data rate of 64 Kbps. So, for video conferencing, a user has to make a call, just like a voice call, and then carry out video conferencing and disconnect the call. This service is known as FOMA (Freedom of Mobile Multimedia Access).

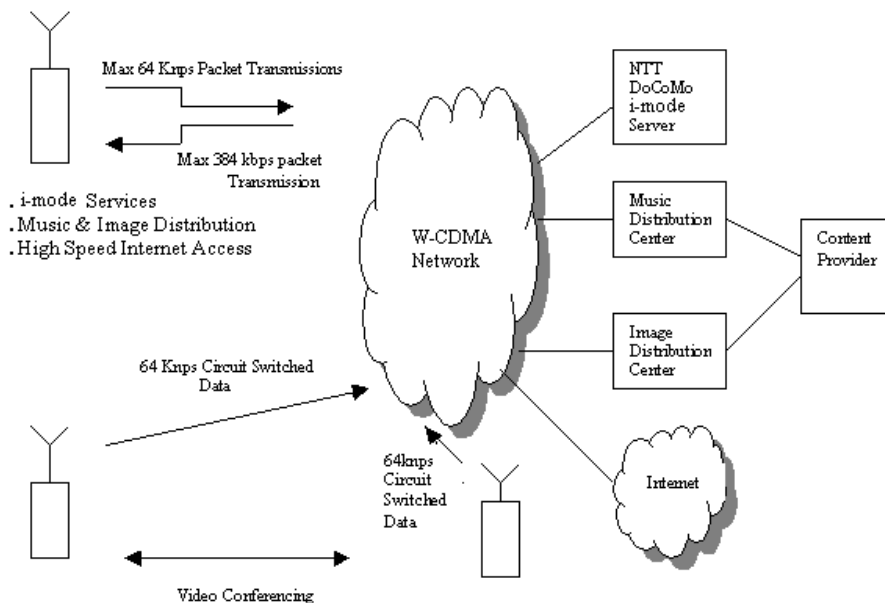


Figure 10-9: NTT DoCoMo's FOMA

Hopefully, by the end the first decade of the 21st century, service such as FOMA will be available throughout the world and one will be able to communicate with anyone, anywhere — making the utopia of a global village a reality.

Summary

This chapter gives an overview of the evolution of wireless networks. The first-generation wireless networks for mobile communications were analog, and the second-generation wireless networks were digital systems. Out of the various 2G systems, the GSM systems were the most widely deployed systems. To provide wireless Internet access over the 2G systems, special protocols, such as Wireless Application Protocol and i-mode were developed. Using these protocols, Web content can be accessed through mobile phones at data rates up to 14.4 Kbps. Hence, content is generally limited to text and low-resolution graphics. To increase the data rates for Internet access, the 2.5G systems were proposed which are evolving from the 2G systems. The 2.5G systems support data rates up to 144 Kbps and, hence, high-resolution graphics and limited multimedia applications can be supported. The 2.5G systems are based on the TDMA technology and the GPRS and CDMA technology. To increase the speeds further, the 3G system standardization activity has been initiated. The 3G systems can be broadly categorized as the CDMA2000 systems and the W-CDMA systems. 3G systems support data rates in the range 384 Kbps to 2.048 Mbps to provide full-fledged multimedia applications. Using 3G systems, users can access a wide

variety of services that include text, graphics, multi-party audio, and video conferencing. In addition, location-based services are likely to become very popular with the advent of the 3G systems.

Wireless Internet access holds a great promise — the wireless subscribers are likely to grow from 400 million in 2000 to 1800 million by 2010. By 2010, 60 percent of the traffic is expected to be multimedia. Therefore, the operators and equipment manufacturers can reap rich benefits. More than that, the users will benefit greatly — a wide variety of services will be available to them anywhere, anytime at a very low cost. In order for the 3G systems to become popular, research needs to focus on wireless device technology and content development technology, in addition to the network technologies. Handheld devices are also evolving rapidly. The mobile handsets are no longer low power devices: handheld devices with mobile operating systems, such as the Win CE, Palm OS, OS/9, and so forth are available with higher memory and high-resolution color display to present the Internet content. These handheld devices can run small Java Virtual Machines called KVM and can interpret Java code. So, what we can do today on desktops can soon be done through the handheld devices, thereby making “anywhere, anytime” communication a reality.

Chapter 11

Advanced 3G Programming

The objective of the 3G wireless networks is to provide to end users applications that support high data rates. The end user is not concerned with the underlying technologies, but he/she is interested in using the network for personal, business, or leisure activities. So, the thrust for operators is to develop the content that can provide appealing applications. Unlike the 2G networks, 3G networks can support animation, audio, and video applications, which require high bandwidth. In this chapter, we focus on creating the content to provide multimedia applications.

We study 3G programming using different languages. These include: Wireless Markup Language (WML), which has some presence as a legacy language for a few more years; Extensible HyperText Markup Language (XHTML), which is the markup language standardized in the next version of Wireless Application Protocol (WAP 2.0); Extensible Markup Language (XML), which is now more widely used for developing Internet content; and Java, the network programming language that provides true platform independence and multimedia support. First, we study the various issues involved in 3G programming, and then we discuss the implementation of real-world examples, which illustrate the ease with which content can be developed for 3G networks.

3G Application Development Issues

NTT DoCoMo was the first operator to introduce 3G services, in May 2001. In all other countries, the 3G services are likely to be introduced on a large scale between 2002 and 2004. Until the market for 3G services matures, the subscribers of wireless data services will have wireless devices of different capabilities. The subscribers can be divided into the following categories:

- ◆ Subscribers with mobile phones that support low data rates (up to 14.4 Kbps). These subscribers have WAP-enabled handsets that understand only WML content.
- ◆ Subscribers with mobile phones that support data rates up to 64 Kbps. These subscribers have handsets that support GPRS and WAP. Consequently these handsets are capable of handling both circuit-switched data and packet-switched data. They also support, to some extent, audio and video streaming applications in addition to animation.
- ◆ Subscribers with high-end wireless devices that run a mobile operating system (such as Palm OS and Win CE) that can support high data rates (384 Kbps upwards). These devices, because of higher memory capacities, can run a sophisticated browser and interpret the content written in XHTML or XML. These devices can also run a Kilobytes Virtual Machine (KVM) and can interpret the Java code.
- ◆ Subscribers with laptops that can run a desktop operating system, but who would like to use the wireless network from fixed locations. These devices are capable of handling data rates up to 2 Mbps and can support full-fledged multimedia with video conferencing.

To cater to all these categories of subscribers is the challenge of 3G programming. As a consequence, we have different markup languages and different tools to develop 3G applications. We briefly review these languages and tools in the next section.

The wired Internet provides access to information through wired devices at very high speeds. The same content needs to be made available through wireless devices. Due to the limited capabilities of wireless devices, many markup languages have been proposed to develop content that can be presented through a browser that has minimal processing requirements. In this section, we study the limitations of WML and the new languages for 3G application development namely, XHTML, XML, and Java — the language for networked applications.

WML

The revolution of wireless Internet access started with WAP, which used WML as the markup language. A good number of sites are presently available that provide WML content. Also, WAP-enabled phones will be cheaper than the other high-end 3G mobile devices. So, WAP will continue to be present for some time, and WML content, which has already been developed, will continue to be available. However, WML has the following limitations:

- ◆ The graphics capability of WML is very limited. We need to create wireless bitmap (WBMP) images, which can be displayed one at a time.
- ◆ Because WAP is meant for low-speed, wireless networks, the WBMP files provide very low-resolution graphics. To enable an image to be transmitted over wireless networks with low data rates, utilities are available that convert images into the WBMP format. These utilities reduce the data rate required to transmit the image and also reduce the resolution of the image.
- ◆ To create animation, we need to follow a roundabout approach; we need to create a WML deck that contains a number of WML cards. Each card is displayed for a few milliseconds and is followed by the next one. The amount of time a card is displayed is controlled through the “ontimer” tag, which specifies the time in milliseconds. In the next section, we see how animation can be achieved using this approach.
- ◆ WML does not support tags for playing audio and video files directly.
- ◆ Because most of Internet content is in HTML, conversion of HTML to WML is not efficient. Many tags supported in HTML are not available in WML. Hence, to provide content to mobile devices using WML is a considerable task.

For these reasons, WML for 3G-applications development is not an attractive choice. WAP 2.0 was released in July 2001 using XHTML as the markup language — of course, with backward compatibility for WML.

However, it needs to be mentioned here that WML is still a good choice for the legacy 2G networks, particularly for obtaining focused information such as stock quotes, news, and simple database access, as discussed in the previous chapters.

XML

You may be wondering, if most Internet content is available in HTML, why not use HTML for 3G programming? Though HTML has been standardized, the browsers that interpret HTML content create a lot of problems. Some of these problems are as follows:

- ◆ The HTML-content creators did not follow the syntax strictly (for example, many HTML programmers do not use close tags). Still, to present the content properly on the browser, the browser developers use all the necessary processing power to interpret the content properly.
- ◆ Because the browser runs on a desktop with high processing power, the browser itself is a complex piece of software with high memory requirements. To access content from mobile devices with limited capability, the browser needs to be much simpler. If the browser has to be simpler, we need a markup language that has strict syntax.

- ◆ As users wanted more and more features for accessing the vast amount of information on the Web, new tags were added for content creation; and browsers became proprietary. This resulted in the content not being displayed uniformly on all browsers.

To overcome these problems, XML has been standardized — XML is a meta-language and hence we can define our own tags using XML. WML is derived from XML. The word “extensible” is mainly to indicate the following feature: New tags can be created as long as the documents are well-formed. A well-formed document has the following features:

- ◆ **All elements must have opening and closing tags.** Omitting closing tags is not allowed. In HTML, if we do not close the body tag (`</body>`), some browsers (such as Internet Explorer) may not complain; but this is not allowed in XML.
- ◆ **All elements must be nested properly.** For instance, in WML, a `<wml>` tag can contain the `<card>` tag. Unless the card tag is closed with `</card>`, you can’t close the wml document with `</wml>`.
- ◆ **There must be a single root element, which contains all other elements.** For example, the root for a WML document is `<wml>` under which all other elements are used.

Because XML is a very powerful meta-language, we need a special parser to parse XML documents. Popular parsers for Java are SAX and DOM, which can be downloaded from Sun Microsystems’ Web site: www.java.sun.com. Only the wireless devices with high processing capability can handle such parsers. However, the most widely used browsers, such as Internet Explorer and Netscape Navigator, can interpret XML documents.

XHTML

The advantages of XML and HTML are combined to form XHTML. XHTML uses the vocabulary of HTML and the syntax of XML. The tags are identical to HTML but as the syntax is that of XML. XHTML documents can be interpreted by any XML user agent (desktop, palmtop, or mobile phone). The advantages of XHTML are:

- ◆ Because most Internet content is in HTML, you don’t need to rewrite a large amount of code. You only need to ensure that the syntax is followed strictly.
- ◆ Because most of today’s browsers interpret HTML, they can also interpret XHTML, so you don’t have to change or get new browser software.
- ◆ Different browsers can display the content in the same fashion.
- ◆ Developing browsers for wireless devices is not very complex because of the strict syntax of XHTML. The computing power and memory requirements are fewer than those for HTML or XML.

The other good news is that you hardly need to expend much effort to learn XHTML if you know HTML. The following are important differences between HTML 4.0 and XHTML:

- ◆ A DTD declaration has to be provided at the top of the file.

```
<!DOCTYPE PUBLIC "-//W3C/DTD XHTML 1.0 Strict/EN" "">
```
- ◆ A reference to the XML namespace has to be included in the html element.

```
<html xmlns=http://www.w3.org/TR/xhtml1>
```
- ◆ Tag names and attribute names must be in small letters.
- ◆ All attribute values must be enclosed in quotation marks.
- ◆ All tags have to be closed.
- ◆ Tags must nest properly.
- ◆ `<head>` and `<body>` elements should not be left out.

- ◆ The first element in the head must be the <title> element.

These are the basic differences between HTML and XHTML. We illustrate XHTML programming with examples in the next section.

Java

Java is the programming language for network computing. The main advantage of Java is its platform independence. In addition, the Java code can move from machine to machine, thereby providing the capability for network computing. The Java applet can be downloaded from the server and executed in a client. The Java programming language provides dynamic content creation capability through applets.

Using Java for Internet computing with desktop as the client is now taken for granted. However, the Java Virtual Machine (JVM) must be running on the client. JVM can interpret the Java code that is downloaded from a server on the Internet. Now for the problem: The JVM (for the Java Standard Edition that runs on the desktop) requires large memory and high processing capability. These are available on desktops but not on mobile devices. To overcome this problem, Sun Microsystems developed J2ME (Java 2 Micro Edition), which has a Virtual Machine that occupies only a few Kilobytes. This virtual machine is called KVM (K stands for Kilobytes). The KVM can run on a wireless device, and the wireless device can download Java compatible code and execute it. These mobile devices are referred to as Mobile Information Devices (MIDs), and the applications that run on these devices are called MIDlets. Just the way Java revolutionized Internet access through desktops in the 1990s, it is likely to revolutionize Internet access through wireless devices in the first decade of the 21st century. We will study how to create wireless applications using the wireless tool kit of Sun Microsystems later in this chapter.

Implementation of Real-World 3G Applications

In this section, we discuss the implementation of 3G applications. We will discuss how the content can be created using WML, XHTML, XML, and Java.

Animation Using WML

In this example, we create an animated image using WML that can be displayed on WAP-enabled devices. If you are using a tool kit's phone emulator, you cannot simulate the speed of the network. So, on an actual WAP-enabled mobile phone, the animation will be much slower than on a phone emulator.

Because WML does not support animated images to be given directly as attributes for the image tag, we need to create a series of images and use the timer tag to display one image after another. In this example, you first need to create six images and convert them into a WBMP format from a BMP format by using a tool for that purpose. These WBMP files are stored in different files with filenames IMG001.wbmp, IMG002.wbmp, and so on. The code is given in Listing 11-1. Note that the images are displayed in the following order: 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1. The details of the code follow.

Listing 11-1: WML code for animated image using timer

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version="1.0"?>
2. <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">
3. <wml>
4. <card id="card1" title="Animation" ontimer="#card2">
5. <timer value="1"/>
6. <p align="center">
7. 

```

```

8. </p>
9. </card>
10. <card id="card2" title="Animation" ontimer="#card3">
11. <timer value="1"/>
12. <p align="center">
13. 
14. </p>
15. </card>
16. <card id="card3" title="Animation" ontimer="#card4">
17. <timer value="1"/>
18. <p align="center">
19. 
20. </p>
21. </card>
22. <card id="card4" title="Animation" ontimer="#card5">
23. <timer value="1"/>
24. <p align="center">
25. 
26. </p>
27. </card>
28. <card id="card5" title="Animation" ontimer="#card6">
29. <timer value="1"/>
30. <p align="center">
31. 
32. </p>
33. </card>
34. <card id="card6" title="Animation" ontimer="#card7">
35. <timer value="1"/>
36. <p align="center">
37. 
38. </p>
39. </card>
40. <card id="card7" title="Animation" ontimer="#card8">
41. <timer value="1"/>
42. <p align="center">
43. 
44. </p>
45. </card>
46. <card id="card8" title="Animation" ontimer="#card9">
47. <timer value="1"/>
48. <p align="center">
49. 
50. </p>
51. </card>
52. <card id="card9" title="Animation" ontimer="#card10">
53. <timer value="1"/>
54. <p align="center">
55. 
56. </p>
57. </card>
58. <card id="card10" title="Animation" ontimer="#card11">
59. <timer value="1"/>
60. <p align="center">
61. 
62. </p>
63. </card>

```

```

64. <card id="card11" title="Animation" ontimer="#card1">
65. <timer value="1"/>
66. <p align="center">
67. 
68. </p>
69. </card>
70. </wml>

```

Code description

- ◆ Line 1: Indicates the XML version being used by the tool kit
- ◆ Line 2: Indicates the document-type definition for creating the WML page
- ◆ Line 3: Start tag of the WML page
- ◆ Line 4: Start tag for the WML card. This card has the ID "card1" and title "Animation". The ontimer event is used in this tag to perform a task when a stipulated amount of time is completed. When the timer expires, card2 is displayed.
- ◆ Line 5: The timer tag — the tag for initiating the timer and setting the timer. The time is in milliseconds. In this line only, the timer tag is closed.
- ◆ Line 6: The para tag — the tag in which the actual content is put. It contains an attribute align, which aligns the content.
- ◆ Line 7: The image tag, the tag for inserting the image. This tag has the src as an attribute to pass the image (IMG001.wbmp); alt is another compulsory attribute, which contains the alternative text for the image. It is useful because when the image is not displayed, the alt tag says what the image is. Here, alt is blank.
- ◆ Line 8: Close of para tag
- ◆ Line 9: Close of card tag
- ◆ Lines 10–15: Another card with card id as "card2". The content of the card is same as the previous card with changes in the card id and the image passed. The image to be displayed is IMG002.wbmp.
- ◆ Lines 16–21: Another card with card id as "card3". The content of this card is same as the above card with changes in the card id and the image passed. The image to be displayed is IMG003.wbmp.
- ◆ Lines 22–27: Another card with card id as "card4". The image is IMG004.wbmp.
- ◆ Lines 28–33: Another card with card id as "card5". The image is IMG005.wbmp.
- ◆ Lines 34–39: Another card with card id as "card6". The image is IMG006.wbmp.
- ◆ Lines 40–45: Another card with card id as "card7". The image is IMG005.wbmp.
- ◆ Lines 46–51: Another card with card id as "card8". The image is IMG004.wbmp.
- ◆ Lines 52–57: Another card with card id as "card9". The image is IMG003.wbmp.
- ◆ Lines 58–63: Another card with card id as "card10". The image is IMG002.wbmp.
- ◆ Lines 64–69: Another card with card id as "card11". The image is IMG001.wbmp. Note that on expiration of timer, the card1 will be displayed.
- ◆ Line 70: Closing of the wml deck

Code output

After entering the code in the WML environment of the Nokia tool kit, save the file (animation.wml). Click the Compile button and then the Show button. The output of the code is shown in the simulator. You will see a moving ball above the welcome message. Some screen shots are shown in Figure 11-1.

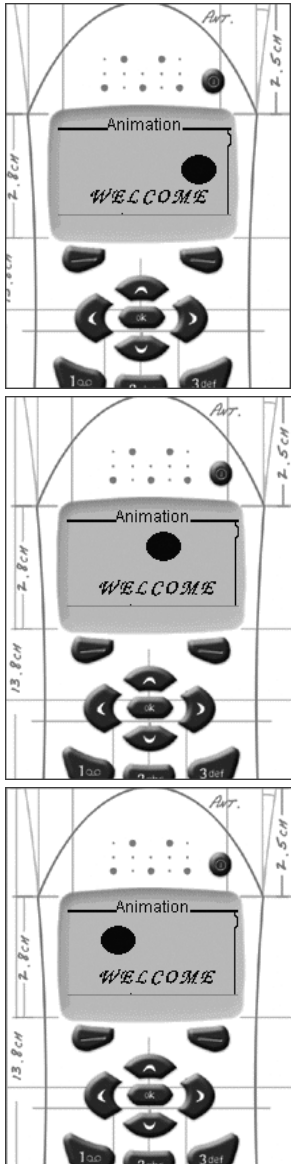


Figure 11-1: Snapshots of the animated image displayed on the WAP phone

Now you can appreciate the complexity involved in creating animation through WML. Because of the support for display of only one low-resolution image at a time, we had to create a series of images and use the timer to control the display. However, on a high-speed network, you can still view animation well. However, content creation is pretty tough. Precisely for this reason, XHTML is adapted as the standard for content creation in WAP version 2.0.

In the next set of examples, we will see the power of XHTML to create content. As you can see, XHTML has the capability to create applications using a much simpler code.

Animation Using XHTML

The following example illustrates how to create animation in an XHTML document. To run this application, you need to create an animated file in GIF format (you can use any popular software, such as

Adobe Photoshop). In this example, let's assume that a file with the filename `anim.gif` is created. Listing 11-2 gives the XHTML code for displaying an animated file.

As compared to WML, this is a much simpler code because the animation files can be given as arguments to the `img` tag in XHTML, whereas WML allows only WBMP files, which are static images. The code given in Listing 11-2 is explained below:

Listing 11-2: Animation through XHTML

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
5. <head>
6. <title>This is Animation Example</title>
7. </head>
8. <body>
9. <table border="1">
10. <tr><td width="75" height="75"></td></tr>
11. </table>
12. </body>
13. </html>

```

Code description

- ◆ Line 1: Indicates the version of XML being used and the encoding format
- ◆ Lines 2-3: Indicate the document type, the DTD used for content development and the reference location of the DTD
- ◆ Line 4: Start tag with XML name spacing reference location
- ◆ Line 5: Head start tag
- ◆ Line 6: Title of the page with start and end tags for the title
- ◆ Line 7: Close tag for head
- ◆ Line 8: Body start tag, which defines the body of the document
- ◆ Line 9: Table start tag with attribute border. It specifies the border of the table.
- ◆ Line 10: Row tag with `td` (table data cell) tag which is a cell. The `td` tag contains the image (`img`) tag with `src` (source) as attribute for passing the source of the animated GIF image.
- ◆ Line 11: Table close tag
- ◆ Line 12: Body close tag
- ◆ Line 13: HTML close tag, which closes the document.

Code output

After typing the code using any text editor, save the file with the extension `HTM` or `HTML`. Open the browser and call the file from the location where you saved it. You can also open the file by double-clicking the left mouse-button on the file. Images, as shown in Figure 11-2 are displayed — this animation is a falling drop.

Play an Audio File Using XHTML

Here is an example to play an audio file through an XHTML document. You need to create a file `myClip.wav`, which contains a recorded file. This file can be created using an audio blaster card of your

PC. Store the file in the WAV format. Listing 11-3 shows the code for playing an audio file. Details of the code follow.

Listing 11-3: XHTML code to play an audio file

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
5. <head>
6. <title>This is Audio Example</title>
7. </head>
8. <body>
9. <table border="1">
10. <tr><td><embed src="myClip.wav" autostart="false" width="200" height="45"
    /></td></tr>
11. </table>
12. </body>
13. </html>

```



Figure 11-2: Snapshot of the animated image displayed on the browser

Code description

- ◆ Line 1: Indicates the version of XML being used and the encoding format.
- ◆ Lines 2–3: Indicate the document type and the DTD used for content development, the reference location of the DTD.
- ◆ Line 4: HTML start tag with XML name spacing reference location
- ◆ Line 5: Head start tag

- ◆ Line 6: Title with start and close tags for the title
- ◆ Line 7: Head close tag
- ◆ Line 8: Body start tag, which defines the body of the document
- ◆ Line 9: Table start tag with attribute border, which specifies the border of the table.
- ◆ Line 10: Row tag with td (table data cell) tag, which is a cell. The td tag contains the embed tag with src (source), autostart, width, height as attributes to embed the sound file into the document. The src specifies the source of the file (`myClip.wav`); autostart is to specify whether to start on load or not; the width and height are to specify the height and width of the embedded file. If autostart is false, the audio file will not be played as soon as the document is loaded. You have to explicitly start it. If the autostart is true, it will be played immediately on loading the file on to the browser.
- ◆ Line 11: Table close tag
- ◆ Line 12: Body close tag
- ◆ Line 13: HTML close tag, which closes of the document.

Code output

After typing the code in any text editor, save the file with extension HTM or HTML (`audio.htm` or `audio.html`). Open the browser and call the file from the location where you saved it. Alternatively, you can also open the document by double-clicking the file. When you open the file, the audio file control pad is displayed (Figure 11-3).

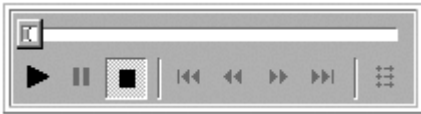


Figure 11-3: Audio file control pad

By clicking the arrow (the play button), you can hear the sound.

You can now feel the power of XHTML. Downloading an audio file (say, a music file) on to a mobile device is extremely simple. After downloaded on to the mobile device, the mobile devices have to pass the audio file to the voice processing module, which will output the file to the speakers.

Play a Video File Using XHTML

This example illustrates how a video clipping can be played through an XHTML document. You need to have a video clipping with the filename `video.mpg` in your system. If you don't have an `.mpg` file in your system, you can download from one of the following sites:

www.bobcam.com/shirley_holmes/ or rmschockey.8m.com/movies.htm. Note that the video file format is MPG as indicated by the extension of the filename. The file format is as per the standards defined by Moving Picture Experts Group (MPEG). MPEG standard is used for transmitting video clippings and also for video conferencing.

Listing 11-4 gives the XHTML code for playing a video clipping. The details of the code follow.

Listing 11-4: XHTML code to play video file

© 2001 Dreamtech Software India Inc.

All rights Reserved

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

```

5. <head>
6. <title>This is Audio Example</title>
7. </head>
8. <body>
9. <table border="1">
10. <tr><td><embed src="video.mpg" type="video/quicktime" width="200"
    height="200" /></td></tr>
11. </table>
12. </body>
13. </html>

```

Code description

- ◆ Line 1: Indicates the version of the XML being used and the encoding format.
- ◆ Lines 2–3: Indicates the document type and the DTD used for content development, the reference location of the DTD.
- ◆ Line 4: HTML start tag with XML name spacing reference location
- ◆ Line 5: Head start tag
- ◆ Line 6: Title of the document, with start and close tags for the title
- ◆ Line 7: Head close tag
- ◆ Line 8: Body start tag, which defines the body of the document
- ◆ Line 9: Table start tag with attribute border, which specifies the border of the table
- ◆ Line 10: Row tag with td (table data cell) tag, which is a cell. The td tag contains the embed tag with src (source), type, width, height as attributes to embed the video into the document. The src is to give the source of the file (in our case, video.mpg); the type is to specify the file type; the width and height are to specify the width and height of the embedded file.
- ◆ Line 11: Table close tag
- ◆ Line 12: Body close tag
- ◆ Line 13: HTML close tag, which closes the document

Code output

After typing the code using any text editor, you can save the file with the extension HTM or HTML (video.htm or video.html). Open the browser and call the file from the location where you saved it. You can also open the document by double-clicking the file. After you open the file, you can see a screen (see Figure 11-4). Then the actual video file starts playing.



Figure 11-4: Video file control pad

Location-Based Services Using XHTML and ASP

In this example, we illustrate how location-based services can be provided to mobile users. If the mobile device is fitted with a GPS (Global Positioning System) receiver, the mobile device can automatically pass the longitude and latitude information of its present location to the server. Otherwise, this information has to be sent by the user to the server by inputting the data.

For this application, you need to create a database of, for example, hotels and hospitals of a location along with the longitude and latitude. After the location information of a mobile device is received, the information on the hotels in that location is sent to the mobile device.

Create a Database

In MS Access or any other database available on your system, create a database named `gps`. The database has one table also named `gps`. The fields in the table are latitude, longitude, restaurants, and hospitals. The latitude is a text field consisting of various latitude values. The longitude is also a text field having longitude values. “restaurants” is a text field having the restaurant names. “hospitals” is also a text field having hospital names.

You also need to write two programs — the first program (`gps.asp`) presents a form to the user to input latitude and longitude, and the second program (`location.asp`) processes the data and presents to the user the restaurants and hospitals in the locality based on the location given by the user.

Listing 11-5 gives the ASP code for `gps.asp`.

Listing 11-5: XHTML code for giving latitude and longitude input

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
5. <head>
6. <title>Location based services</title>
7. </head>
8. <body>
9. <form action="location.asp" method="post">
10. <table border="1">
11. <tr>
12. <td width="25"><font face="arial" size="2">Latitude</font></td>
13. <td width="5"><input type="text" name="latitude" size="2"><font face="arial"
    size="1">degrees</font></input></td>
14. </tr>
15. <tr>
16. <td width="25"><font face="arial" size="2">Longitude</font></td>
17. <td width="5"><input type="text" name="longitude" size="2"><font face="arial"
    size="1">degrees</font></input></td>
18. </tr>
19. <tr>
20. <td align="center">
21. <input type="reset" value="refresh"></input>
22. </td>
23. <td align="center">
24. <input type="submit" value=" submit "></input>
25. </td>
26. </tr>

```

```

27. </table>
28. </form>
29. </body>
30. </html>

```

Code description

- ◆ Line 1: Indicates the XML version that is used and the encoding format, which is UTF-8, meaning ASCII text.
- ◆ Lines 2–3: Indicate the DTD used in which the tags used in the document are defined. It also contains the reference location of the DTD.
- ◆ Line 4: HTML start tag, which also contains the XML name-spacing reference location.
- ◆ Lines 5–7: Head portion of the document with title tag, which specifies the title of the document.
- ◆ Line 8: Start of the body tag
- ◆ Line 9: Form tag, which is the tag to place the elements, such as text boxes, buttons, and so forth. The form tag action attribute is used to perform some action when the form is submitted. The form tag method attribute is to specify the method to be used to perform the task. In this case, after the Submit button is pressed after the form is filled, the data in the form will be posted and `location.asp` is executed.
- ◆ Line 10: Table tag with border attribute. The table tag is to insert the table in the page, and the border attribute is used to add a border to the table. You can make `border="0"` if you do not want a border.
- ◆ Line 11: Table row tag to put a row in the table
- ◆ Line 12: Table data cell tag, which makes the row into divisions; each division is called a cell. The width attribute fixes the width of the division. The font is another attribute, which sets the style and size of the text in the font tag. This line prompts the user to type in **Latitude**.
- ◆ Line 13: Another division tag having the input tag. The input tag is used to put the form components in the page. The input type specified here is `text`, which places a text box. The name of the component is `latitude`, which is used to send the values when the form is submitted. The variable name for the value input by the user is “latitude”.
- ◆ Line 14: Closing of row tag
- ◆ Line 15: Opening of new row
- ◆ Line 16: New division in the row, which prompts the user to give the value of Longitude
- ◆ Line 17: Another division in the row with input type as text with the name longitude. The value input by the user is in the variable “longitude”.
- ◆ Line 18: Closing of the row tag
- ◆ Line 19–26: Code for another row. This row has two divisions each having an input type `reset` and `submit`, respectively. The input type `reset` is used to clear all the values in the text boxes and the input type `submit` is used to submit all the values in the text boxes. The input type `submit` is to submit the form values.
- ◆ Line 27–30: Closing tags for table, form, body and html

The source code for `location.asp` is given in Listing 11-6.

Listing 11-6: XHTML code for displaying the retrieved values from database

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
5. <head>
6. <title>gps locations</title>
7. </head>
8. <body>
9. <%
10. set dbConn=Server.CreateObject("ADODB.Connection")
11. dbConn.Open("dsn=gps")
12. set gps1=dbConn.execute("SELECT * from gps")
13. %>
14. <%
15. While not gps1.eof
16. %>
17. <%
18. if gps1(0)=request.form("latitude") and gps1(1)=request.form("longitude")
19. then
20. <table border="1">
21. <tr>
22. <td widht="15"><font face="arial" size="2">latitude:</font></td>
23. <td width="5"><font face="arial" size="2"><%=gps1("latitude")%></font></td>
24. </tr>
25. <tr>
26. <td widht="15"><font face="arial" size="2">longitude:</font></td>
27. <td widht="5"><font face="arial" size="2"><%=gps1("longitude")%></font></td>
28. </tr>
29. <tr><td widht="15"><font face="arial" size="2">resturants:</font></td>
30. <td widht="5"><font face="arial" size="2"><%=gps1("resturants")%></font></td>
31. </tr>
32. <tr>
33. <td widht="15"><font face="arial" size="2">hospitals</font></td>
34. <td widht="5"><font face="arial" size="2"><%=gps1("hospitals")%></font></td>
35. </tr>
36. </table>
37. <%
38. end if
39. gps1.MoveNext
40. Wend
41. %>
42. <%
43. gps1.close
44. dbConn.close
45. %>
46. </body>
47. </html>

```

Code description

- ◆ Line 1: Indicates the XML version number and the encoding format
- ◆ Lines 2–3: Indicates the DTD used in which the tags used in the document are defined. It also contains the reference location of the DTD.
- ◆ Line 4: HTML start tag, which also contains the name spacing reference location
- ◆ Lines 5–7: The head portion of the document with the title tag, which specifies the title of the document
- ◆ Line 8: Start of the body tag
- ◆ Line 9: Start of the script tag. The script is used to access the database and retrieve the records from the database.
- ◆ Lines 10–13: To create a data object, to open the database using the DSN (Data Source Name), and to declare a variable and assign the record sets retrieved from the database.
- ◆ Lines 14–16: The script while loop, which is used to go through the records step-by-step until the end of the file.
- ◆ Lines 17–19: Script if condition where the `if` condition is used to check whether a condition is satisfied or not. The condition is whether the variables passed in the `gps.asp` are found or not. If the variables are found, further processing will continue. The variables passed contain the latitude and longitude values.
- ◆ Lines 20–22: Table row and data cell tags
- ◆ Line 23: Another division tag with script to write a record retrieved from the database
- ◆ Line 24: Closing of row
- ◆ Lines 25–28: Another row with two divisions. The second division has a script in which to put a record retrieved from the database.
- ◆ Lines 29–31: Another row with two data cells. The second division has a script to display the information about restaurants retrieved from the database.
- ◆ Lines 32–35: Another row with two data cells. The second division has a script to display the information about hospitals retrieved from the database.
- ◆ Line 36: Closing of the table
- ◆ Lines 37–41: Closing of the `if` condition and moving the record set to next record within the `while` loop and closing of the `while` loop
- ◆ Lines 42–45: For closing of the record set and closing of the data object
- ◆ Lines 46–47: Closing of body and HTML tags

Creation of Data Source Name (DSN)/Code output

Creating a DSN will differ depending on your development environment. For Windows 95/98, the following description is valid. Note that you can connect to the database in different ways — here we will discuss ODBC connectivity. To create a DSN, on the desktop, go to Start⇒Settings⇒Control Panel. In the control panel, select the ODBC data sources. When the ODBC data sources are selected, a new window is displayed where user DSN appears. Click the Add button. As the Create a New Data Source Window prompt appears, select the database driver (in this case Access, or if you used another database engine, select the appropriate database driver) and click the Finish button. The ODBC set up window opens. Enter the DSN name, select the database there, and click Finish.

Figure 11-5 shows the display for the user to input the latitude and longitude values. Figure 11-6 shows the display of restaurants and hospitals in that location.

Display an Animated Image in XML Using XSL

We can create the preceding applications using XML through style sheets as well. For illustration, the same examples — animation, audio, and video — are presented here.

The procedure is to create a style sheet and then link it to an XML document. Listing 11-7 gives the style sheet for displaying the animated image. Using a text editor, create the file `animation.xsl` and store it. The details of the code follow.

Listing 11-7: Animation through XSL

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version='1.0'?>
2. <xsl:stylesheet xmlns:xsl="..\New Folder\animation.html">
3. <xsl:template match="/">
4. <html>
5. <head>
6. <title>XML ANIMATION </title>
7. </head>
8. <body>
9. <h1>
10. <CENTER>
11. XML ANIMATION </CENTER></H1>
12. <table align="center" border="0" bgcolor="#ffffff" cellpadding="0"
    cellpadding="0" width="700">
13. <tr>
14. <td width="250" ><IMG ALIGN="center" SRC="C:\WINDOWS\Desktop\graphics
15. \balloons_confetti_lg_clr.gif"></IMG></td>
16. <td width="100" ></td>
17. </tr>
18. </table>
19. </body>
20. </html>
21. </xsl:template>
22. </xsl:stylesheet>

```

Code description

- ◆ Line 1: Indicates the XML version. Note that XML processing instruction starts with `<?` and ends with `?>`.
- ◆ Line 2: Opening tag of the XSL style sheet with XML name spacing
- ◆ Line 3: Style sheet template declaration start tag and setting the root node through the match attribute
- ◆ Line 4: Opening HTML tag
- ◆ Line 5: Opening head tag
- ◆ Line 6: Title tag, represents the title of the page on title bar
- ◆ Line 7: Closing head tag
- ◆ Line 8: Opening body tag
- ◆ Line 9: Header tag to specify the heading
- ◆ Line 10: Tag to align the text at the center of the HTML page
- ◆ Line 11: Heading followed by closing of center tag. The header tag is also closed in this line.

- ◆ Line 12: Tag to open a table. The attributes are to set the border (in this case, no border as the value is 0), background color, cell spacing, padding, and the width.
- ◆ Line 13: Tag to open table row
- ◆ Lines 14–15: Tag td is for opening the table data. The animated image location is specified by src. The full path where the animated image is located has to be specified.
- ◆ Line 16: Closing tag for table data
- ◆ Line 17: Closing tag for table row
- ◆ Line 18: Closing tag for table
- ◆ Line 19: Closing tag for body
- ◆ Line 20: Closing tag for HTML document
- ◆ Line 21: Closing tag for template
- ◆ Line 22: Closing XSL tag

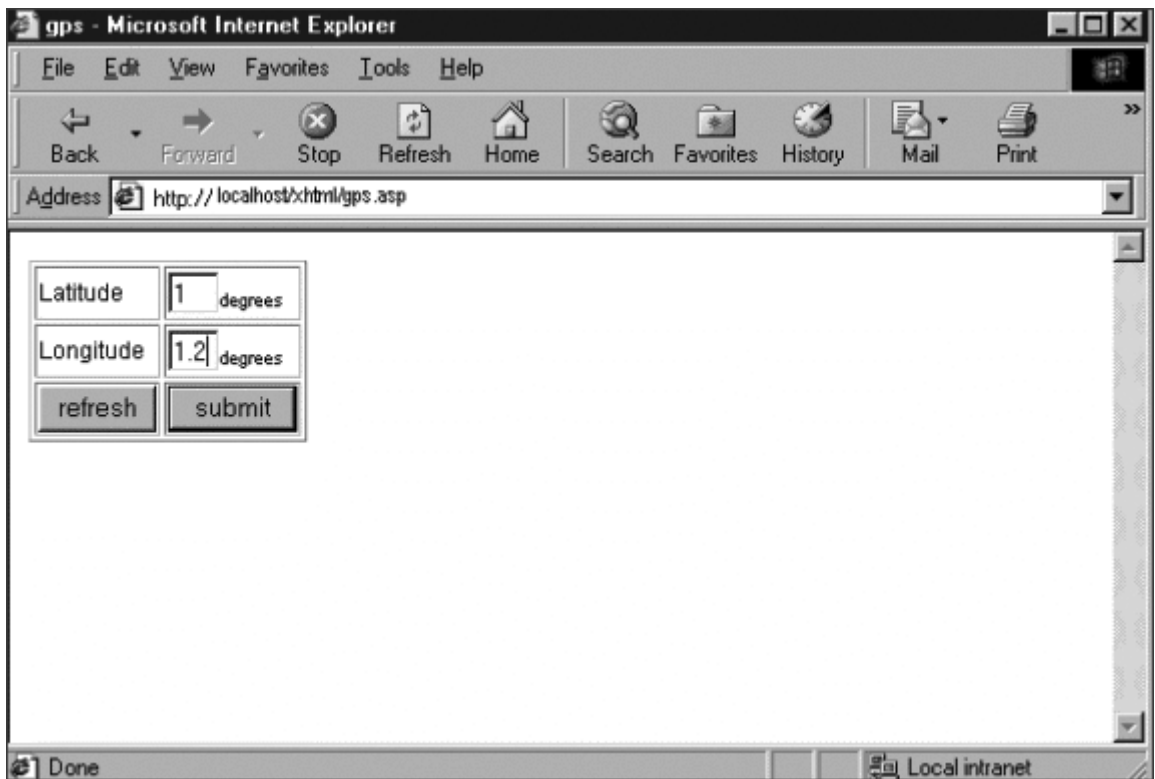


Figure 11-5: Screen to input latitude and longitude

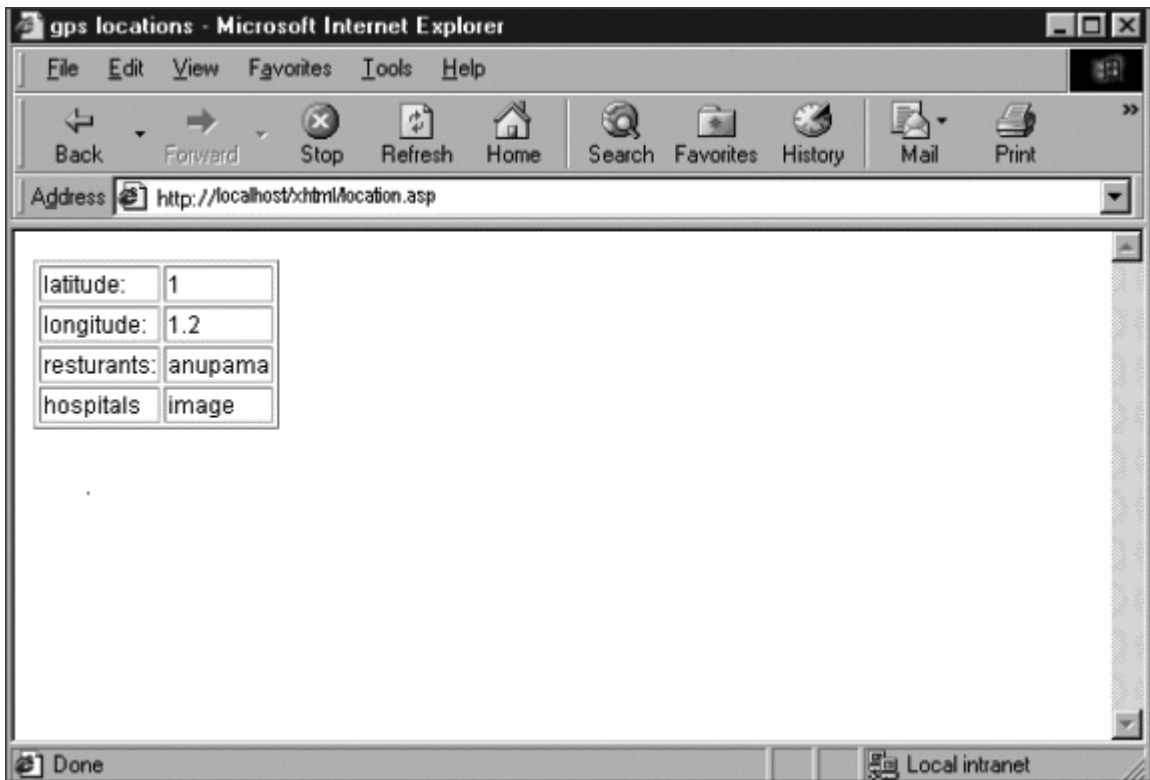


Figure 11-6: Display of restaurants and hospitals in the locality

Listing 11-8 gives the XML code that links the XML file with the style sheet `animation.xsl`.

Listing 11-8: XML code for animation

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version='1.0'?>
2. <xml-stylesheet type="text/xsl" href="animation.xsl"?>
3. <animation>
4. </animation>

```

Code description

- ◆ Line 1: Indicates the XML version number
- ◆ Line 2: This code links the animated file with the xsl style sheet through href attribute.
- ◆ Line 3: This code indicates the opening of the root element.
- ◆ Line 4: This code indicates the closing of the root element.

Code output

Save the file as `animation.xml`. After you run the XML code in Internet Explorer, you can see the animated object as in Figure 11-7.



Figure 11-7: Animation display in the browser

Play an Audio File Using XSL

The listing for playing an audio file using XSL is done on similar lines as the animation. The code is provided in Listing 11-9.

Listing 11-9: Audio through XSL

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version='1.0'?>
2. <xsl:stylesheet xmlns:xsl="..\New Folder\audio.html">
3. <xsl:template match="/">
4. <html>
5. <head>
6. <title>XML AUDIO </title>
7. </head>
8. <body>
9. <h1>
10.     <center> XML AUDIO </center></h1>
11. <table align="center" border="0" bgcolor="#ffffff" cellpadding="0"
12.     cellspacing="0" width="700">
13. <tr>
14. <td width="250" ><bgsound src = "file:///c:/windows/desktop/trial_xml/0.5.wav"
15.     loop="-1"></bgsound></td>
16. <td width="100" ></td>
17. </tr>

```



```

16. </table>
17. </body>
18. </html>
19. </xsl:template>
20. </xsl:stylesheet>

```

Code description

- ◆ Line 1: Indicates the XML version
- ◆ Line 2: Opening tag of the XSL style sheet with XML name spacing
- ◆ Line 3: Style sheet template declaration start tag and setting the root node through the match attribute
- ◆ Line 4: Opening HTML tag
- ◆ Line 5: Opening head tag
- ◆ Line 6: Opening title tag, represents the title of the HTML page on the title bar. The title tag is also closed in the same line.
- ◆ Line 7: Closing head tag
- ◆ Line 8: Opening body tag
- ◆ Line 9: Opening heading tag
- ◆ Line 10: To align the text at center of the HTML page. The heading is also specified here, and the center tag and the heading tag are closed.
- ◆ Line 11: Tag for creating a new table. The attributes are alignment, border, background color, cell spacing and padding as well as the width.
- ◆ Line 12: Tag for opening table row
- ◆ Line 13: Tag for opening table data. The tag bgsnd is to play the sound in background. The source of the sound is specified by src whose value is the complete pathname of the file. The bgsnd tag is also closed in this line.
- ◆ Line 14: Closing table data tag
- ◆ Line 15: Closing table row tag
- ◆ Line 16: Closing table tag
- ◆ Line 17: Closing body tag
- ◆ Line 18: Closing HTML tag
- ◆ Line 19: Closing template tag
- ◆ Line 20: Closing XSL tag

Save this file with the filename `audio.xml`. Create an XML document with Listing 11-10.

Listing 11-10: XML code for audio

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. <?xml version='1.0'?>
2. <xml-stylesheet type="text/xsl" href="audio.xml"?>
3. <audio>
4. </audio>

```

Code description

- ◆ Line 1: Indicates XML version number

- ◆ Line 2: To link the animated file with xsl sheet
- ◆ Line 3: Opening of the root element
- ◆ Line 4: Closing of the root element

Save the file with extension .xml.

Code output

Run the XML code in Internet Explorer. You can hear the audio through the speakers on your multimedia PC.

Play a Video Using XSL

To play video with XSL, you follow the same procedure as for animation. You need to have a video clipping in MPG format to test this application. Listing 11-11 gives the XSL code for creating a style sheet.

Listing 11-11: Playing video through XSL

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. <?xml version='1.0'?>
2. <xsl:stylesheet xmlns:xsl="..\New Folder\video.html">
3. <xsl:template match="/">
4. <html>
5. <head>
6. <title> XML VIDEO </title>
7. </head>
8. <body>
9. <h1>
10. <center>
11. XML VIDEO </center></h1>
12. <table align="center" border="0" bgcolor="#ffffff" cellpadding="0"
    cellspacing="0" width="700">
13. <tr>
14. <td width="250" ></img> </td>
15. <td width="100" ></td>
16. </tr>
17. </table>
18. </body>
19. </html>
20. </xsl:template>
21. </xsl:stylesheet>

```

Code description

- ◆ Line 1: Indicates the XML version number.
- ◆ Line 2: Opening tag of the XSL style sheet with XML name spacing
- ◆ Line 3: Style sheet template declaration start tag and setting the root node through a match attribute
- ◆ Line 4: Opening HTML tag
- ◆ Line 5: Opening head tag
- ◆ Line 6: Opening title tag, which represents the title of the page on the title bar. After the opening tag, the title is given followed by closing tag for title.

- ◆ Line 7: Closing head tag
- ◆ Line 8: Opening body tag
- ◆ Line 9: Opening tag for heading
- ◆ Line 10: Tag to center align the text of the HTML page
- ◆ Line 11: The heading is followed by closing center tag and closing heading tag.
- ◆ Line 12: Opening tag for table row
- ◆ Line 13: Opening tag for table row
- ◆ Line 14: Opening table data and the image tag. The image tag contains the attribute dynsrc, which specifies the source of the video by the complete path name. The image and table data tags are also closed in this line.
- ◆ Line 15: Closing table data tag
- ◆ Line 16: Closing table row tag
- ◆ Line 17: Closing table tag
- ◆ Line 18: Closing body tag
- ◆ Line 19: Closing HTML tag
- ◆ Line 20: Closing template tag
- ◆ Line 21: Closing XSL tag

Save the code with extension XSL. The style sheet has to be linked to an XML document, which is shown in Listing 11-12.

Listing 11-12: XML code for video

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. <?xml version='1.0'?>
2. <xml-stylesheet type="text/xsl" href="video.xsl"?>
3. <video>
4. </video>
```

Code description

- ◆ Line 1: Indicates the version number of XML
- ◆ Line 2: To link the file with the XSL sheet
- ◆ Line 3: Opening of the root element
- ◆ Line 4: Closing of the root element

Code output

Save the file with a XML extension. Run the XML code in Internet Explorer, and you can now see the video file.

Development of a Mobile Advertising Application Using the Wireless Tool Kit

As discussed in an earlier chapter, you can use the wireless tool kit to create applications for mobile devices by writing Java programs. Recall that a Java application created for running on Mobile Information Devices (MIDs) is called a MIDlet. You can run the MIDlets either through the KToolBar or through Forte for Java Development environment. Forte for Java is an Integrated Development

environment, which provides facilities to edit, compile, build, and execute MIDlets. Recall that to run a MIDlet, invoke the Forte for Java and then right-click on the MIDlet Suite icon RunAll and click the Execute menu item, as shown in Figure 11-8. The tool kit provides four types of phone emulators; you can use the DefaultColorPhone or DefaultGrayPhone to test the applications.



Figure 11-8: Running a MIDlet

Listing 11-13 gives the listing of the MIDlet to create a mobile advertising application. The application has to display a set of items (cosmetics, jewelry, books, clothing). When the user selects an item, it has to display the new products that have arrived (such as earrings, necklaces, and bracelets if the user selects jewelry). The code details follows.

Listing 11-13: MIDlet for mobile advertising

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. import javax.microedition.lcdui.*;
2. import javax.microedition.midlet.*;
3. public class MAdvertising extends MIDlet
4. implements CommandListener {
5. Display display = null;
6. List menu = null;
7. List choose = null;
8. List choose1 = null;
9. List choose2 = null;
```

```

10. List choose3 = null;
11. Ticker ticker = new Ticker("Mobile Components");
12. static final Command backCommand = new Command("Back",
13.     Command.BACK, 0);
14. static final Command mainMenuCommand = new Command("Main",
15.     Command.SCREEN, 1);
16. static final Command exitCommand = new Command("Exit",
17.     Command.STOP, 2);
18. String currentMenu = null;
19. public MADvertising() { }
20. public void startApp()
21.     throws MIDletStateChangeException {
22.     display = Display.getDisplay(this);
23.     menu = new List("Advertising Products",
24.         Choice.IMPLICIT);
25.     menu.append("Cosmetics", null);
26.     menu.append("Jewelry", null);
27.     menu.append("Books", null);
28.     menu.append("Clothing", null);
29.     menu.addCommand(exitCommand);
30.     menu.setCommandListener(this);
31.     menu.setTicker(ticker);
32.     mainMenu();
33. }
34. public void pauseApp() {
35.     display = null;
36.     choose = null;
37.     choose1 = null;
38.     choose2 = null;
39.     choose3 = null;
40.     menu = null;
41.     ticker = null;
42. }
43. public void destroyApp(boolean unconditional) {
44.     notifyDestroyed();
45. }
46. //Main menu
47. void mainMenu() {
48.     display.setCurrent(menu);
49.     currentMenu = "Main";
50. }
51. public void testCosmetics() {
52.     choose = new List("Cosmetic Products", Choice.MULTIPLE);
53.     choose.setTicker(new Ticker("Cosmetics..."));
54.     choose.addCommand(backCommand);
55.     choose.setCommandListener(this);
56.     choose.append("Nail Polish", null);
57.     choose.append("Body Lotion", null);
58.     choose.append("Body Spray", null);
59.     display.setCurrent(choose);
60.     currentMenu = "Cosmetics";
61. }
62. public void testJewelry() {
63.     choose1 = new List("Jewelry", Choice.MULTIPLE);
64.     choose1.setTicker(new Ticker("Jewelry..."));
65.     choose1.addCommand(backCommand);

```

```

65.     choose1.setCommandListener(this);
66.     choose1.append("Earrings", null);
67.     choose1.append("Necklace", null);
68.     choose1.append("Bracelets", null);
69.     display.setCurrent(choose1);
70.     currentMenu = "Jewelry";
71. }
72. public void testBooks() {
73.     choose2 = new List("Books", Choice.MULTIPLE);
74.     choose2.setTicker(new Ticker("Books..."));
75.     choose2.addCommand(backCommand);
76.     choose2.setCommandListener(this);
77.     choose2.append("Java", null);
78.     choose2.append("VB", null);
79.     choose2.append("VC", null);
80.     display.setCurrent(choose2);
81.     currentMenu = "Books";
82. }
83. public void testFashion() {
84.     choose3 = new List("Clothing", Choice.MULTIPLE);
85.     choose3.setTicker(new Ticker("Clothing..."));
86.     choose3.addCommand(backCommand);
87.     choose3.setCommandListener(this);
88.     choose3.append("Trousers", null);
89.     choose3.append("T-Shirts", null);
90.     choose3.append("Jeans", null);
91.     display.setCurrent(choose3);
92.     currentMenu = "Clothing";
93. }
94. public void commandAction(Command c, Displayable d) {
95. String label = c.getLabel();
96.     if(label.equals("Exit")) {
97.         destroyApp(true);
98.     } else if (label.equals("Back")) {
99.         if(currentMenu.equals("Cosmetics") ||
100. currentMenu.equals("Jewelry") ||
101. currentMenu.equals("Books") ||
102. currentMenu.equals("Clothing")) {
103.             // go back to menu
104.             mainMenu();
105.         }
106.     } else {
107.         List down = (List)display.getCurrent();
108.         switch(down.getSelectedIndex()) {
109.             case 0: testCosmetics();break;
110.             case 1: testJewelry();break;
111.             case 2: testBooks();break;
112.             case 3: testFashion();break;
113.         }
114.     }
115. }
116. }

```

Code Description

- ◆ Lines 1–2: This code is to import the necessary class libraries.

- ◆ Lines 3–4: To create a class MAdvertising, which extends the MIDlet
- ◆ Lines 5–10: Initialization of menu and the choice list
- ◆ Line 11: Ticker class which displays a horizontal scrolling message “Mobile components”
- ◆ Lines 12–17: Creation of soft keys Back, Main, and Exit
- ◆ Line 18: Initialization of current Menu object
- ◆ Lines 19–32: This code creates the list items for “Advertising Products” namely, Cosmetics, Jewelry, Books, and Clothing
- ◆ Lines 33–41: Re-initialization of the list items
- ◆ Lines 42–44: Code to call the destroyApp object
- ◆ Lines 45–49: To display the main menu items
- ◆ Lines 50–60: If cosmetics are chosen, the various subitems are displayed here. Note that the horizontal scrolling is now changed to “Cosmetics”.
- ◆ Lines 61–71: If Jewelry is chosen, the various sub-items are displayed and horizontal scrolling is changed to “Jewelery”.
- ◆ Lines 72–82: If the item books is chosen in the main menu, the subitems are displayed using this code. The horizontal scrolling changes to “Books”.
- ◆ Lines 83–93: If the item Clothing is chosen, the sub-items are displayed using this code. The horizontal scrolling changes to “Clothing”.
- ◆ Lines 94–97: This code is to check if the soft key Exit is pressed and if so, to exit the application.
- ◆ Lines 98–115: This code keeps track of the various soft keys pressed by the user on the phone emulator. This code checks whether the back button is pressed and, if so, based on the previous actions, the previous screen is displayed. Internally, the history of the various keys pressed is stored in a history stack, and if the back button is pressed, the last entry on the stack is displayed. Otherwise, main menu is displayed.

Code Output

When this MIDlet is executed using the Forte for Java, you will see the screens in Figure 11-9.





Figure 11-9: Mobile adverting displayed on the Wireless tool kit emulator

You can compare the power of MIDlet with that of a WML-based application using this example. The look and feel of the application will be much better using the MIDlet approach because of the power of Java programming language.

Summary

In this chapter, we studied the programming aspects of 3G content development. We illustrated the complexity involved in creating simple animation using WML. For example, in order to animate images with WML, we need to create a number of static, low-resolution images in WMBP format and use a timer to display the images one after the other. Using XHTML, we can obtain animation by downloading an animated GIF file directly through simple code. We also illustrated how the audio and video clippings can be played using XHTML as well as XML and XSL. Using the wireless tool kit of Sun Microsystems, we illustrated the capability of creating applications on wireless devices that run the Java Virtual Machine.

Chapter 12

3G Programming Using BREW

The two main systems for wireless Internet access are the CDMA-based systems and the GSM-based systems. The GSM systems have been widely deployed in Europe and CDMA-based systems have a wide installation base in North America. On other continents, both types of systems are used. Qualcomm Corporation, which pioneered the CDMA technology, has released Binary Runtime Environment for Wireless (BREW) to develop applications that which can be ported on to any mobile device. In this chapter, we will discuss implementation of applications using BREW. An overview of BREW capabilities is given followed by the implementation of applications along with the source code.

BREW Overview

BREW provides the necessary tools to develop applications for deployment on CDMA-based wireless networks. The applications can be developed on the standard PC environment and tested before actual deployment on the network. For deployment on a commercial basis, a development organization has to obtain the necessary certification from Qualcomm.

BREW SDK 1.0 supports images, graphics, and sound files in MIDI (Musical Instrument Digital Interface) and MP3 (MPEG Audio Layer 3) formats. BREW SDK 1.0.1, released in August 2001, has support for position location and application messaging as well as improved emulation capabilities.

BREW SDK can be used for developing wireless applications not only for 3G networks but for the existing wireless networks as well. BREW applications are independent of the underlying air interface. As the data rates supported by the wireless networks become higher and higher, the user experience in running the applications become better due to faster response.

The BREW development environment consists of a Graphical User Interface (GUI) to develop the applications and a BREW Emulator to emulate the mobile device. A number of predefined mobile device emulators with options to change the configuration parameters are available. The advantage of this is that the portability of the application for devices with different capabilities can be checked.

Applications developed using BREW are called *applets*. *Modules* can also be developed, which can be used by several applets. Applets and modules are developed in C or C++ as stand-alone DLLs and are loaded into BREW Emulator at runtime. In this chapter, we use the words *applets* and *applications* interchangeably.

The BREW development kit can be downloaded from <https://brewx.qualcomm.com/developer/sdk/> and installed on your system. The system requirements are Windows NT 4.0 or higher or Windows 2000 with 128 MB RAM. To develop applications, Microsoft Visual Studio 6.0 or higher should also be installed. While developing applications, note that floating-point operations cannot be used. Although the application using floating-point operations may run on the Emulator, the target mobile devices do not support floating-point operations and should be avoided.

After the BREW tool kit is loaded onto your system, you can run the sample applications provided in the kit by launching the Emulator. The BREW Emulator automatically launches the Application Manager showing the icons and applet names on the screen. To run the applications involving sound files, you

need to have a sound card installed along with the necessary driver software. You can check the functionality of the sound card by playing a stored MIDI file. All the sound files are kept in the Music folder in the Application folder. The graphics files are in the Animation folder with a BMP extension and with one-bit-per-pixel resolution.

In the following sections, we will discuss how to create applications using the BREW tool kit. To start with, we will demonstrate how to create a simple application through a step-by-step procedure. After that, we will discuss the code for the following applications:

- ◆ Creating animation
- ◆ Downloading a music file onto a mobile device
- ◆ Mobile advertising
- ◆ Creating a database application.

Using BREW to Develop a New Application

In this section, we discuss how to create a small application that displays a welcome message on the mobile device. This example illustrates the step-by-step procedure to create the application; after you're familiar with the procedure, creating applications more complicated than those discussed in this chapter will be quite an easy task.

To develop a new application by using BREW, the steps are:

1. Create the ClassID file, meaning, BID (Binary Identification), and module information file, MIF, for the module.
2. Place the MIF in the applet directory or in a separate MIF directory.
3. Copy the workspace for one of the sample applications into your work area.
4. Write your application.
5. Place the Module DLLs in a corresponding subdirectory within the applet directory.
6. Launch the BREW Emulator and execute the application.

Each of these steps is explained in detail in the following sections; here you will create a small application to display a welcome message.

Create BID and MIF files

To create a new application in a BREW environment, first a BID file has to be created. A BID file contains the applet or application identification. Every application must have a unique identification. The BID file is used to pass the ClassID information to the application.

The BID file is created using the MIF editor. The MIF editor is launched through Start⇒Programs⇒BREW⇒MIF Editor. When you click the menu, the window in Figure 12-1 appears.

If a new application is being created, a new ClassID has to be generated or an existing BID file can also be used. To select the existing BID file, click the Browse for BID file button. To create a new ClassID, click the New Applet button at the bottom of the window. The window in Figure 12-2 appears after the New Applet button is clicked.

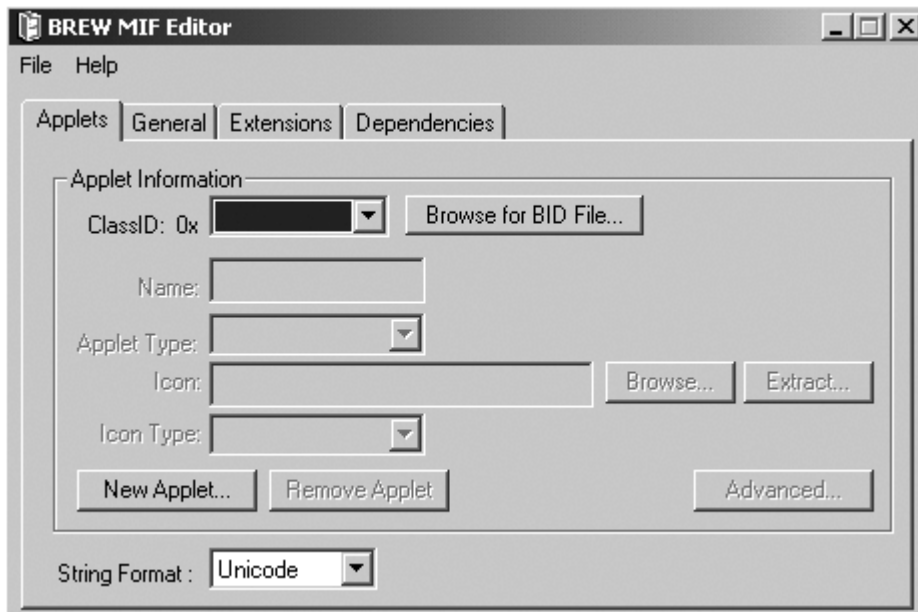


Figure 12-1: The MIF Editor window

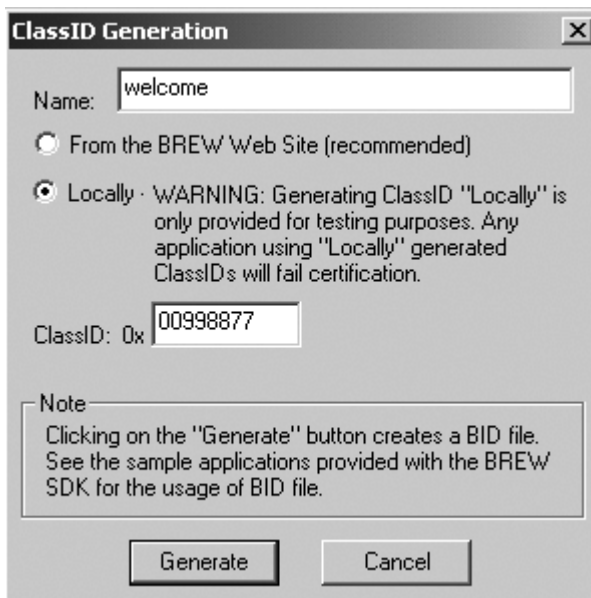


Figure 12-2: The ClassID Generation window

Enter the name of the ClassID in the Name text box and a hexadecimal number in the ClassID text box. By default the option, “From the BREW Web Site” is on. However, to create and test an applet or application locally, select the Locally option. Fill in the details and click the Generate button. A window appears asking where to save the BID file. Save the file in the Applet folder or in the Application folder. When the BID file is saved the window in Figure 12-3 appears.

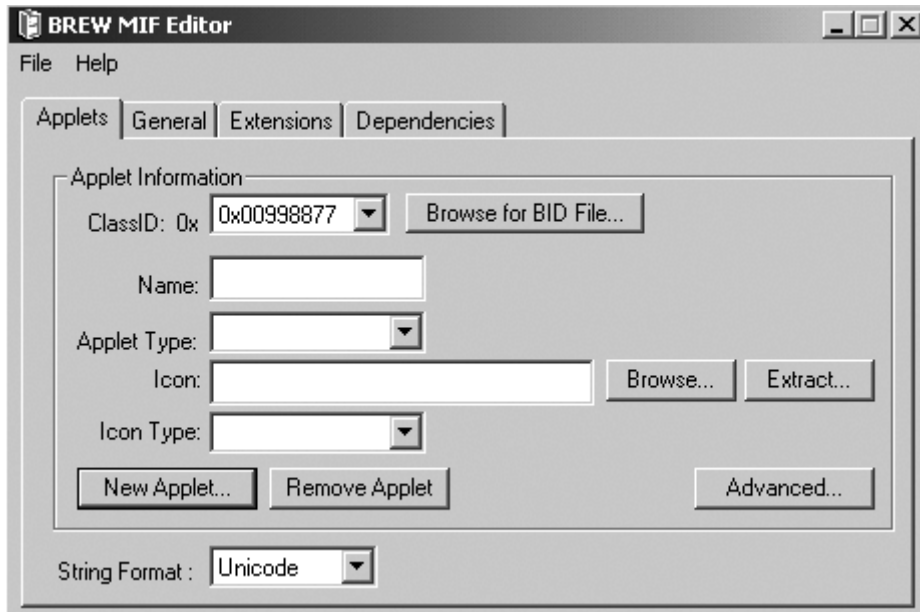


Figure 12-3: The Brew MIF Editor window after entering the ClassID

Enter the Name, Applet Type, Icon, and Icon Type fields. The Name field is compulsory. Click the File Save button and save the file in the Applet directory or in a separate MIF folder. The application manager processes each MIF to obtain the list of applications whose information is present within that MIF. Using the information available from the MIF, the Application Manager loads the applications. By default, a MIF folder is provided along with the BREW SDK. The folder is in the Examples folder. All the applications of MIF files are saved in that MIF folder.

Note that the name of the MIF file and the BID file and the name that is provided in the ClassID generation window should be the same.

Resource Editor

If the application uses some resources, such as string, bitmap, or dialog controls, the Resource Editor has to be used to build the resources. Open the Resource Editor Start⇒Programs⇒BREW⇒Resource Editor. If you click the menu, the window in Figure 12-4 appears. Depending on the requirement of the application, string, bitmap, or dialog control resources can be built. To create a string resource, click the menu Resource⇒New String or right-click the String in the window. The window in Figure 12-5 appears. Now select New String, and the window in Figure 12-6 appears. Enter the Resource ID, Resource Name, String Type, and Value and then click the OK button at the bottom of the window. A String resource is created.

To create a Bitmap resource, click the menu Resource⇒New Bitmap or directly right-click on the Bitmap in the BREW Resource Editor window. The window in Figure 12-7 appears.

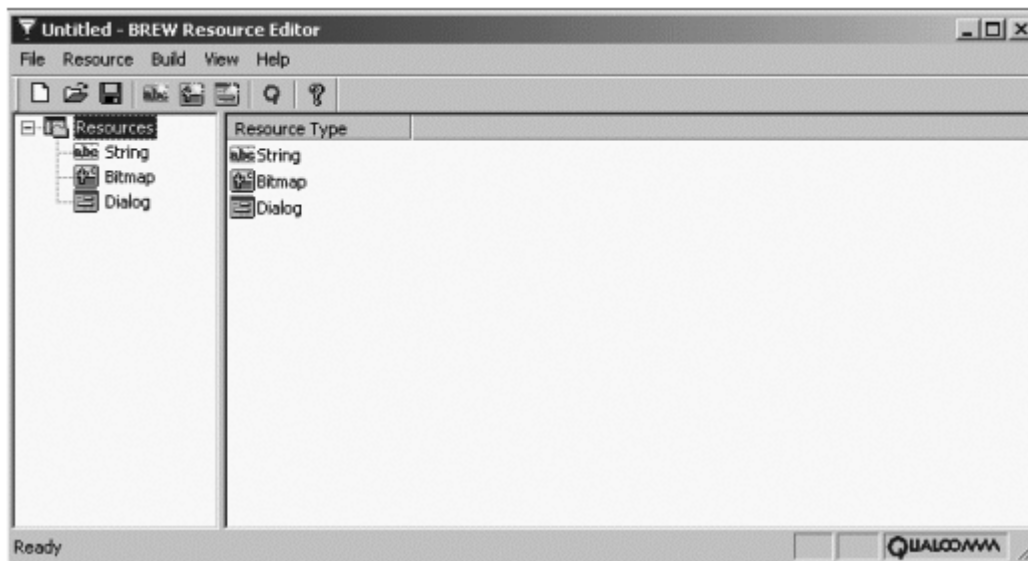


Figure 12-4: The Resource Editor window

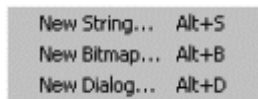


Figure 12-5: The Options window

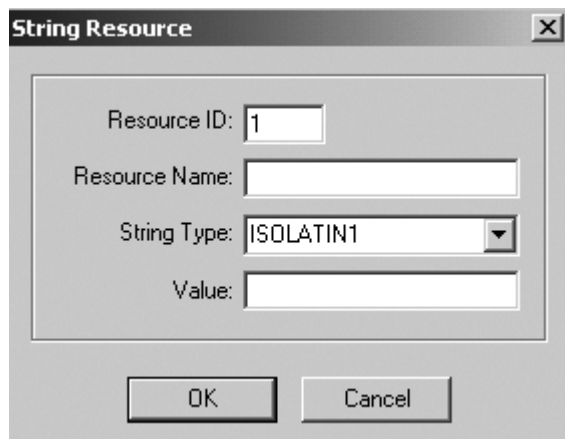


Figure 12-6: The String Resource window

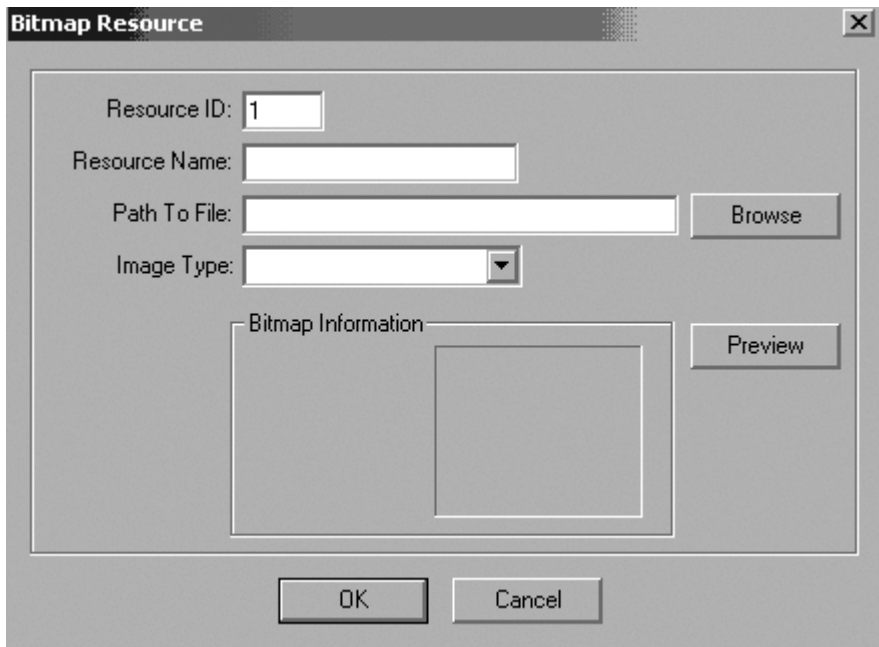


Figure 12-7: The Bitmap Resource window

Enter the Resource ID, Resource Name, Path To File, and Image Type and then click the OK button at the bottom of the window. A Bitmap resource is created.

To create a Dialog resource (we use it in the later examples), click the menu Resource⇒New Dialog or directly right-click on the Dialog in the window. The window in Figure 12-8 appears.

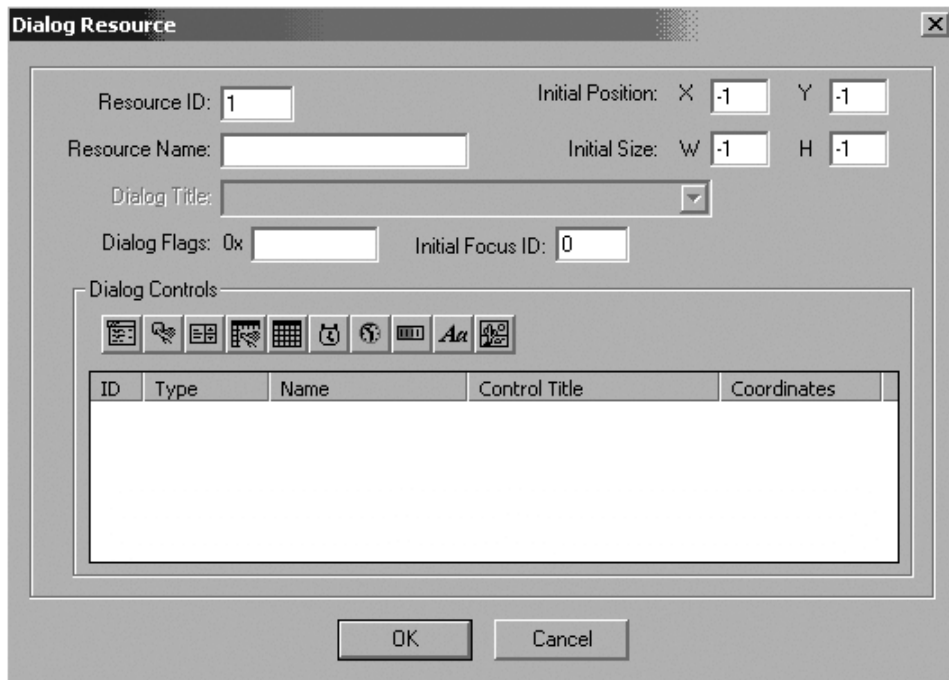


Figure 12-8: The Dialog Resource window

Enter the Resource ID, Resource Name, Dialog Title, Dialog Flags, Initial Focus ID, Initial Position X, Y, and Initial Size W, H and then select the type of control to use. Click the OK button at the bottom of the window after completing the above procedure. A Dialog resource is created.

After creating the resources, the main Resource Editor window appears. After all the Resources are created, click the Build menu. The Build command creates the resource files, the Resource header file and a BAR file. The header file has to be included in the application, and the BAR file is defined as a resource file in the application.

Copy the workspace of a sample application and write a new application. The application is written in the C language, so open the workspace that you copied and modify the C file in the workspace. (See Listing 12-1.)

Listing 12-1: Welcome.c

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. #include "AEEModGen.h"
2. #include "AEEAppGen.h"
3. #include "AEEShell.h"
4. #include "AEEDisp.h"
5. #include "welcome.bid"
6. static boolean eventHandle(IApplet * pi, AEEEvent aee,
    uint16 ui, uint32 dui);
7. int AEELsCreateInstance(AEECLSID ClsId, IShell * ish, IModule * po, void **
    obj)
8. {
9. *obj = NULL;
10. if(ClsId == AEECLSID_WELCOME){
11. if(AEEApplet_New(sizeof(AEEApplet), ClsId, ish, po, (IApplet**)obj,
12. (AEEHANDLER)eventHandle, NULL)
13. == TRUE)
14. {
15. return (AEE_SUCCESS);
16.
17.
18. return (EFAILED);
19. }
20. static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui)
21. {
22. AEEDeviceInfo di;
23. AECHAR szBuf[] = {'W','e','l','c','o','m','e','\0'};
24. AEEApplet * app = (AEEApplet*)pi;
25. switch (aee)
26. {
27. case EVT_APP_START:
28. ISHELL_GetDeviceInfo (app->m_pIShell, &di);
29. IDISPLAY_ClearScreen (app->m_pIDisplay);
30. IDISPLAY_DrawText(app->m_pIDisplay, AEE_FONT_BOLD, szBuf, -1, 0, 0,
31. 0, IDF_ALIGN_CENTER | IDF_ALIGN_MIDDLE);
32. IDISPLAY_Update (app->m_pIDisplay);
    return(TRUE);
33. case EVT_APP_STOP:
34. return TRUE;

```

```

35.default:
36.break;
37.}
38.return FALSE;
39.}

```

Code Description

- ◆ Line 1: Header file, which contains the module interface definitions.
- ◆ Line 2: Header file, which contains the applet interface definitions.
- ◆ Line 3: Header file, which contains the ishell interface definitions.
- ◆ Line 4: Header file, which contains the idisplay interface definitions. All these header files are provided along with the BREW SDK. The header files are included at the beginning of the application program. The header files provide the definitions of the functions and of the interfaces provided.
- ◆ Line 5: BID file, which is an applet ClassID. This file contains the AEECLSID_WELCOME ClassID for the applet.
- ◆ Line 6: Declaration of the function prototype of `HandleEvent`.
- ◆ Lines 7–19: Code for create instance method. This method is invoked after the application begins. The method verifies the ClassID and then invokes the `AEEApplet_New()` function provided in the `AEEAppletGen.c`. The `create_instance` method returns the `AEE_SUCCESS` status upon successful loading of the applet and `EFAILED` status upon failure.
- ◆ Lines 20–39: Code for the `handle_event` method. The `handle_event` is used to handle all kinds of events generated in the application. The event type is passed to the `aee` parameter. If the `EVT_APP_START` receives the START event, the `IDISPLAY_ClearScreen()` function clears the screen. The `IDISPLAY_DrawText()` function is used to draw the text in the screen.
- ◆ Line 23: A character array is declared, which contains the string data. In this case, the string is `'welcome'`.
- ◆ Line 32: `IDISPLAY_Update()` function is used to update the screen.

When the user presses the end key, the applet receives the `EVT_APP_STOP` event. After the applet receives this event, it releases the memory occupied by the application.

After writing the application code in the visual studio environment, use the `Build` command to build the application. The application produces a DLL; place the DLL in the application sub-directory in the Applet folder.

Code Output

Run the BREW Emulator. To run the Emulator, go to `Start⇒Programs⇒BREW⇒BREW Emulator`. After the Emulator launches, select the application and run it. Figure 12-9 shows the display. If the application is saved in a separate Applet directory or the MIF file is in a separate MIF directory, change the default applet directory or MIF directory, whichever is needed. To change the Applet directory, go to `File⇒Change applet directory` or go to the `Tools⇒Settings` menu.

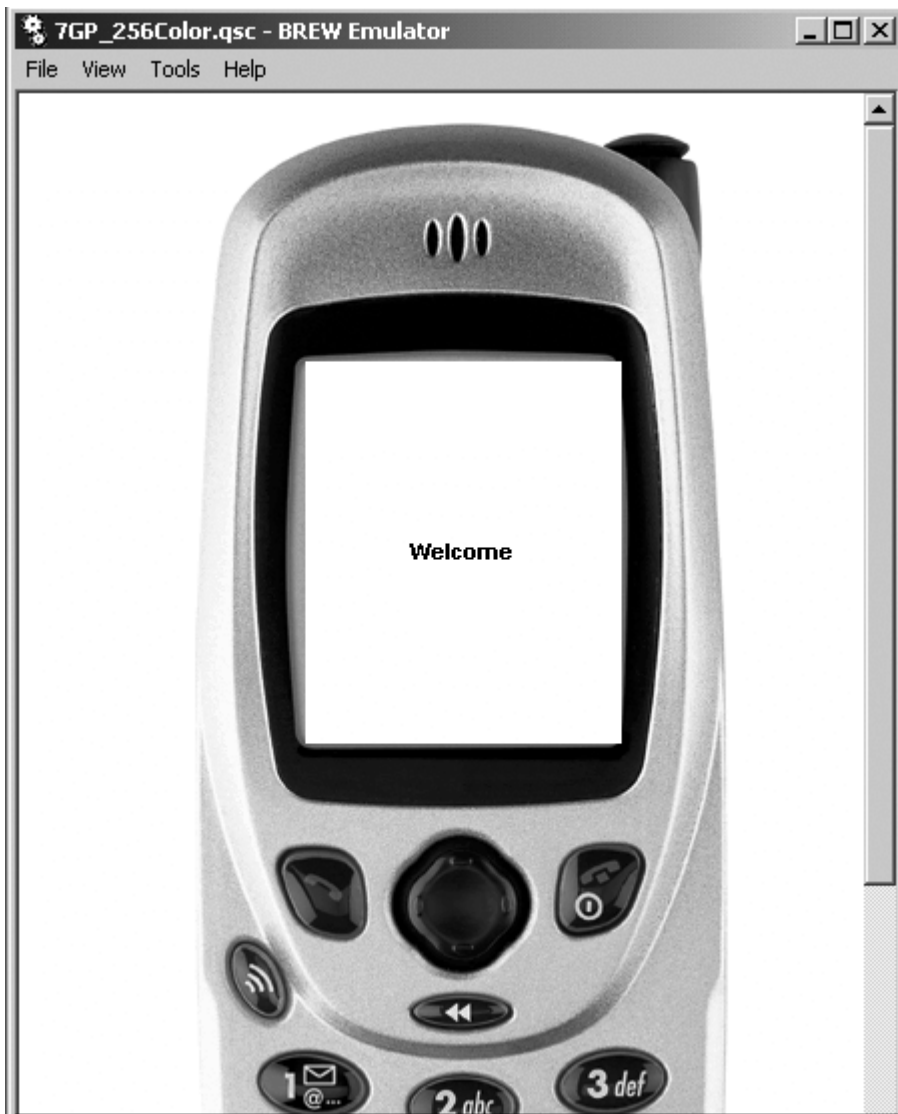


Figure 12-9: The Welcome application on the Emulator

Application: Developing Animation

In this example, we create an application that displays an animated image. The first step to develop the animation application is to create the BID file. To create the BID file go to Start⇒Programs⇒BREW⇒MIF Editor. The MIF Editor is used to create the BID file. The window in Figure 12-10 appears after the MIF Editor menu is clicked. The animation example is a new application, so click the New Applet button at the bottom of the window. The window in Figure 12-11 appears after the New Applet button is clicked.

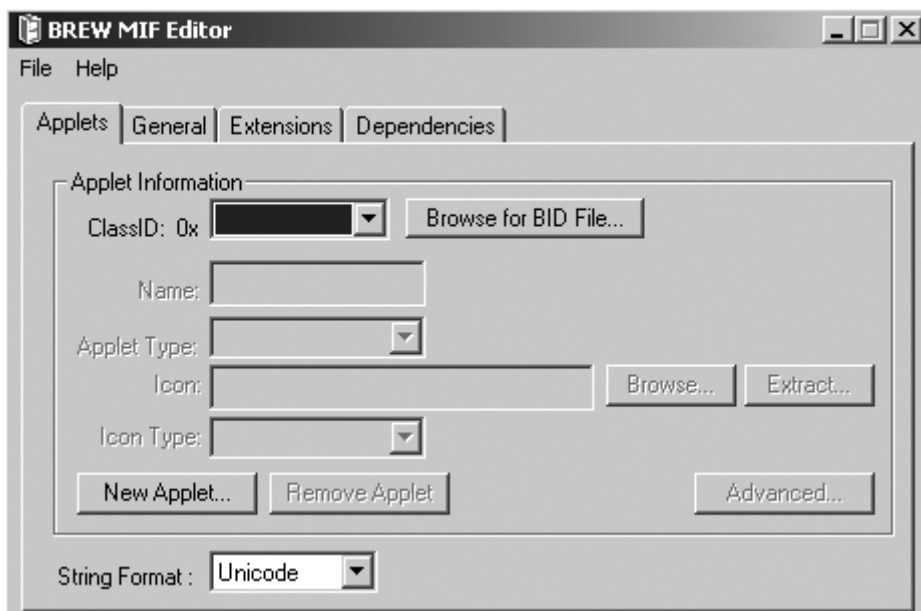


Figure 12-10: MIF Editor window

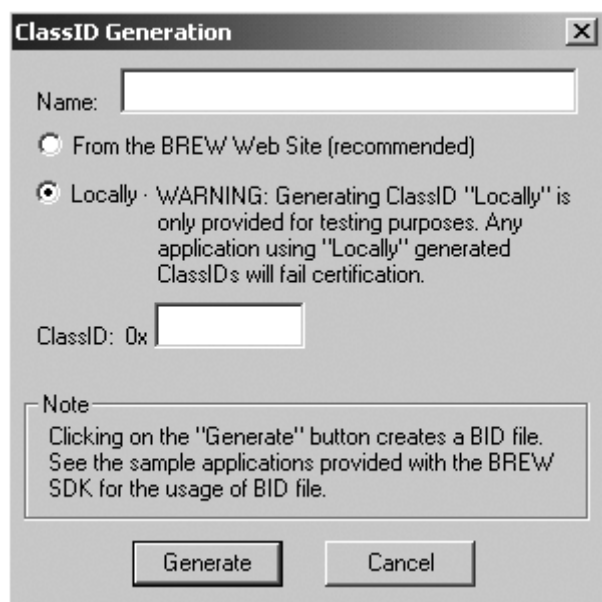


Figure 12-11: The New Applet ClassID Generation window

Enter the name and unique ClassID of the applet. The ClassID name, BID filename, and MIF filename should be the same. Click the Generate button at the bottom of the window. Save the BID file in the application-specific folder or in the Applet directory. The window in Figure 12-12 appears after you click the Generate button.

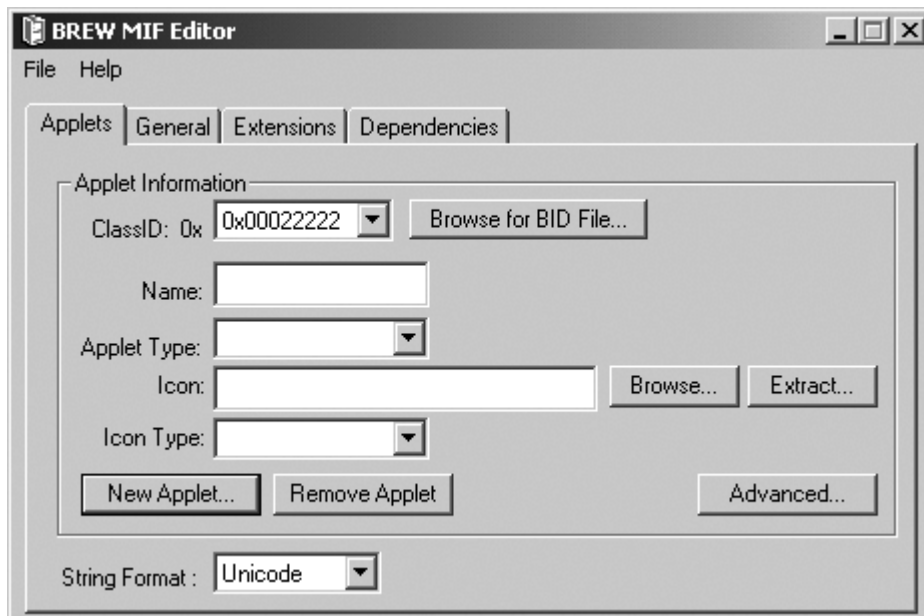


Figure 12-12: The MIF Editor window after entering the ClassID

Enter the Name, Applet Type, Icon, and Icon Type fields. Go to File⇒Save as and save the file in the Applet directory or in a separate MIF folder.

Resource Editor

The application uses bitmap resources. Resource editor has to be used to build the resources. To open the resource editor go to Start⇒Programs⇒BREW⇒Resource Editor. When you click the menu, the window in Figure 12-13 appears.

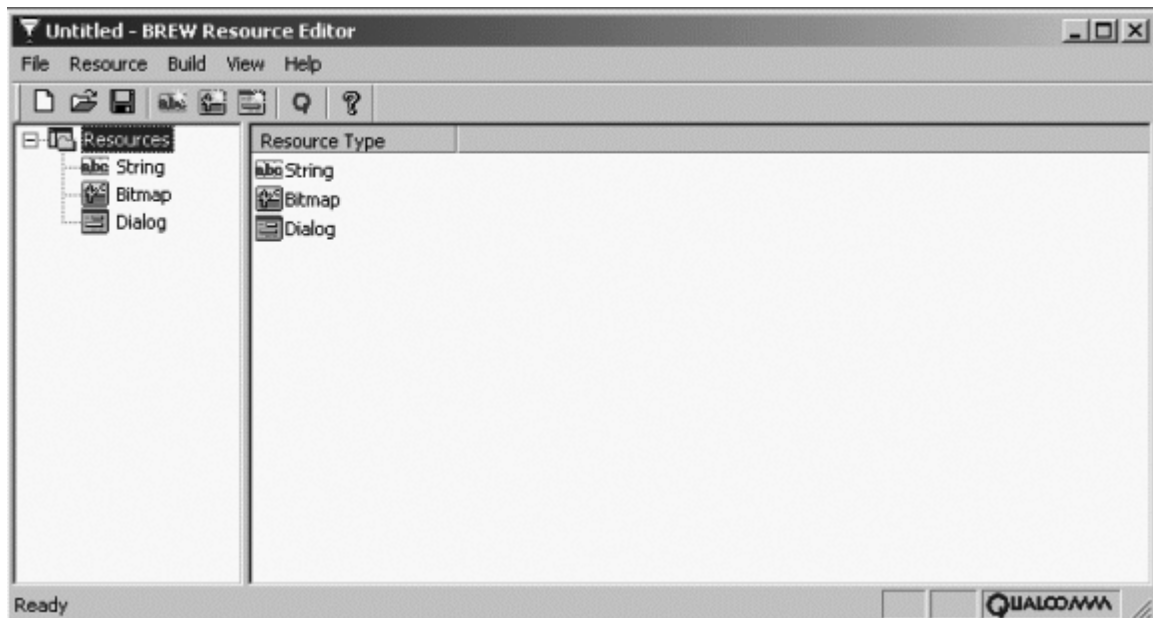


Figure 12-13: The BREW Resource Editor window

To create a Bitmap resource, go to Resource⇒New Bitmap or directly right-click on the Bitmap in the window. The window in Figure 12-14 appears.



Figure 12-14: The Option window

Select New Bitmap. The window in Figure 12-15 appears.

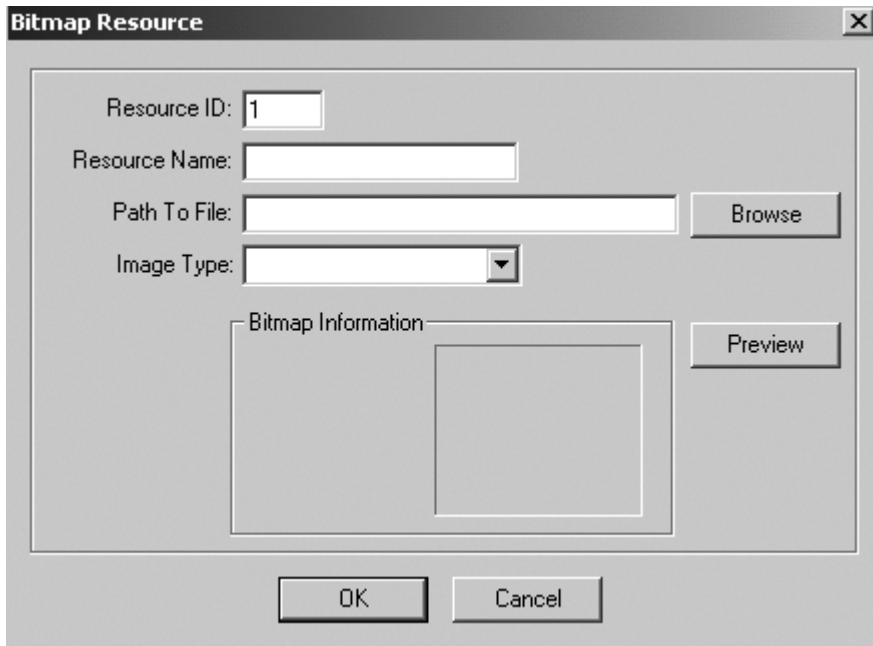


Figure 12-15: The Bitmap Resource window

Enter the Resource ID, Resource Name, Path To File, and Image Type, and click the OK button at the bottom of the window. You can enter the path of the bit-mapped image given in the tool kit. A Bitmap resource is created. After you create the resources, the main Resource Editor window appears. Click the Build menu. The Build command creates the resource files, the Resource header file (animation_res.h), and a BAR (animation.bar) file. The header file has to be included in the application and the BAR file is defined as a resource file in the application.

Copy the workspace of a sample application and write a new application. Open the workspace you copied and modify the C file in the workspace. The animation application code is given in Listing 12-2.

Listing 12-2: Animation.C

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1.#include "AEEAppGen.h"
2.#include "AEEUsageAppIDs.h"
3.#include "AEE.h"
4.#include "AEEShell.h"
5.#include "AEEDisp.h"
6.#include "AEEStdLib.h"
```

```

7.#include "AEEFile.h"
8.#include "AEEMenu.h"
9.#include "AEEGraphics.h"
10.#include "AEEStdLib.h"
11.#include "animation_res.h"
12.#include "animation.bid"
13.typedef struct CIImageApp {
14.AEEApplet          a;
15.IMenuCtl *   ime;
16.IImage *   iim;}
17.CIImageApp;
18.static boolean eventHandle(IApplet * pi, AEEEvent   aee, uint16
    ui, uint32 dui);
19.static boolean initApp(IApplet* app);

20.static void freeApp(IApplet* app);
21.void mainMenu(ima*app);
22.boolean imageUse (ima*app, uint16 ui);
23.#define APP_RES_FILE      "animation.bar"
24.#define IDS_IMAGETITLE    0
25.#define ANIMATION        100
26.int AEEClsCreateInstance(AEECLSID ClsId, IShell* ish, IModule* po,
    void** obj){
27.*obj = NULL;
28.if(ClsId == AEECLSID_ANIMATION){
29.if(AEEApplet_New(sizeof(CIImageApp), ClsId, IShell, po, (IApplet**) obj,
30.(AEEHANDLER)eventHandle, (PFNFREEAPPDATA)freeApp))
31.{ initApp((IApplet*)*obj);
32.return(AEE_SUCCESS);}}
33.return (EFAILED);}
34.static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui){
35.ima* app = (CIImageApp*)pi;
36.if (app == NULL || app->a.m_pIShell == NULL)
37.return FALSE;
38.switch (aee) {
39.case EVT_APP_START:
40.if(ISHELL_CreateInstance(app->a.m_pIShell,
41.AEECLSID_MENUCTL, (void **) &app->ime))
42.return TRUE;
43.mainMenu(app);
44.return(TRUE);
45.case EVT_APP_STOP:
46.if (app->iim != NULL){
47.IIMAGE_Release (app->iim);
48.app->iim = NULL;}
49.IMENUCTL_Release(app->ime);
50.return(TRUE);
51.case EVT_KEY:
52.if (ui == AVK_STAR && app->iim != NULL){
53.IIMAGE_Stop (app->iim);
54.return TRUE;}
55.if ((ui == AVK_UP || ui == AVK_DOWN) &&
56.app->iim != NULL){
57.IIMAGE_Release (app->iim);
58.app->iim = NULL;}

```

```

59.IMENUCTL_SetActive(app->ime, TRUE);
60.if (IMENUCTL_HandleEvent (app->ime,
61.EVT_KEY,Param, 0))
62.return TRUE;
63.else return FALSE;
64.case EVT_COMMAND:
65.switch(ui){
66.case ANIMATION:
67.IMENUCTL_SetActive(app->ime, FALSE);
68.imageUse (app, ui);
69.return TRUE;
70.default:
71.return FALSE;}
72.default:
73.break;}
74.return TRUE;}
75.static boolean initApp(IApplet* pi){
76.ima* app = (CIImageApp*)pi;
77.app->iim = NULL;
78.return TRUE;}

79.static void freeApp(IApplet* pi){
80.ima* app = (CIImageApp*)pi;
81.if (app->iim != NULL){
82.IIMAGE_Release (app->iim);
83.app->iim = NULL;}
84.if (app->ime != NULL){
85.IMENUCTL_Release(app->ime);
86.app->iim = NULL;}}
87.boolean imageUse (CIImageApp* app, uint16 ui){
88.AEEDeviceInfo di;
89.AEERect rect;
90.AEEImageInfo aii;
91.char szResFile[] = APP_RES_FILE;
92.if (app == NULL || app->a.m_pIShell == NULL || app->a.m_pIDisplay == NULL)
93.return FALSE;
94.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
95.rect.x = 0;
96.rect.y = 0 ;
97.rect.dx = di.cxScreen ;
98.rect.dy = di.cyScreen;
99.IDISPLAY_EraseRect (app->a.m_pIDisplay,&rect);
100.switch (ui){
101.case ANIMATION:{
102.app->iim=ISHELL_LoadResImage(app->a.m_pIShell,szResFile,
103.IDB_BROWSER_ICON_ANIM);
104.if (app->iim){
105.IIMAGE_SetParm(app->iim, IPARM_NFRAMES, 4, 0);
106.IIMAGE_GetInfo(app->iim, &aii);
107.IIMAGE_Start(app->iim, 60,60);}
108.IDISPLAY_UpdateEx (app->a.m_pIDisplay,TRUE);}
109.break;
110.default:
111.return FALSE;}
112.return TRUE;}
113.void mainMenu(ima*app){

```

```

114.AEERect qrc;
115.AEEDeviceInfo di;
116.AECHAR szBuf[100];
117.if(app == NULL || app->a.m_pIShell == NULL || app->ime == NULL)
118.return;
119.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
120.qrc.x = 0;
121.qrc.y = 0;
122.qrc.dx = di.cxScreen;
123.qrc.dy = di.cyScreen;
124.IMENUCTL_SetRect(app->ime, &qrc);
125.STR_TO_WSTR("Animation Example", szBuf, sizeof(szBuf));
126.IMENUCTL_SetTitle(app->ime, NULL, 0, szBuf);
127.STR_TO_WSTR("1. Animation", szBuf, sizeof(szBuf));
128.IMENUCTL_AddItem(app->ime, 0, 0, ANIMATION, szBuf, 0);
129.IMENUCTL_SetActive(app->ime,TRUE);}

```

Code Description

- ◆ Line 1: Header file. The AEEAppGen.h file consists of the AEEApplet declaration.
- ◆ Line 2: Header file. The AEEUsageAppIDs.h file contains ClassIDs of usage applications.
- ◆ Line 3: Header file. The AEE.h file contains the Standard AEE Declarations.
- ◆ Lines 4–10: Header files. The AEEShell.h contains AEE Shell Services, AEEDisp.h contains AEE Display Services, AEESTdLib.h contains AEE StdLib Services, AEEFile.h contains AEEFile Services, AEEMenu.h contains Menu Services, AEEGraphics.h contains Graphics Routines, AEESTdLib.h contains AEE stdlib services.
- ◆ Line 11: animation_res.h file is created by using the resource editor. This file contains the resources used by the application.
- ◆ Line 12: animation.bid file. This file contains the ClassID of the applet or application.
- ◆ Lines 13–17: Code for the data structure IImageApp. This structure holds the data members of the applet throughout the life of the applet.
- ◆ Line 18: The handle event function declaration.
- ◆ Lines 19–22: Code for the application-specific functions.
- ◆ Line 23: Defines the resource file.
- ◆ Lines 24–25: Defines the application-specific constants.
- ◆ Lines 26–33: Code for creating the instance method. This function is invoked while the application is being loaded. The module must verify the ClassID and then invoke the AEEApplet_New() function that has been provided in AEEAppGen.c. After invoking AEEApplet_New(), this function can do application specific initialization.
- ◆ Lines 34–74: Code for the handle event method. This method handles all the events of the application. The parameter pi is pointer to the AEEApplet structure. The parameter aee specifies the Event sent to this applet. In this, all the events are handled using the switch.
- ◆ Lines 75–78: The InitAppData method. This function initializes application-specific data and allocates memory for the data.
- ◆ Lines 79–86: Code for the FreeAppData method. This method frees data contained in application data and memory for the application data.
- ◆ Lines 87–112: Code for the imageUse method. In line 88, 89, and 90 variables are declared for device information, rectangle, and image information. Character array is declared in line 91. The application resource file is passed to the array. In line 94, device-specific information is loaded.

Lines 95-98 gives the coordinates for the rectangle. Line 102 is for loading the image resource. Line 105 is for setting the image parameters. Line 108 is for displaying the image.

- ◆ Lines 113–129: Code for the build main menu method. Lines 114–115 are the declaration of variables. Lines 120–123 define the coordinates for drawing the initial rectangle. Line 125 sets a title for the display. Line 127 is for putting a text option; when you select the text the animation is displayed.

Code Output

Figure 12-16 shows the initial screen of the Emulator when the animation example is chosen. Figure 12-17 shows the animation applet is loaded on the Emulator. Figure 12-18 displays the animation screenshot.



Figure 12-16: Emulator with Animation application

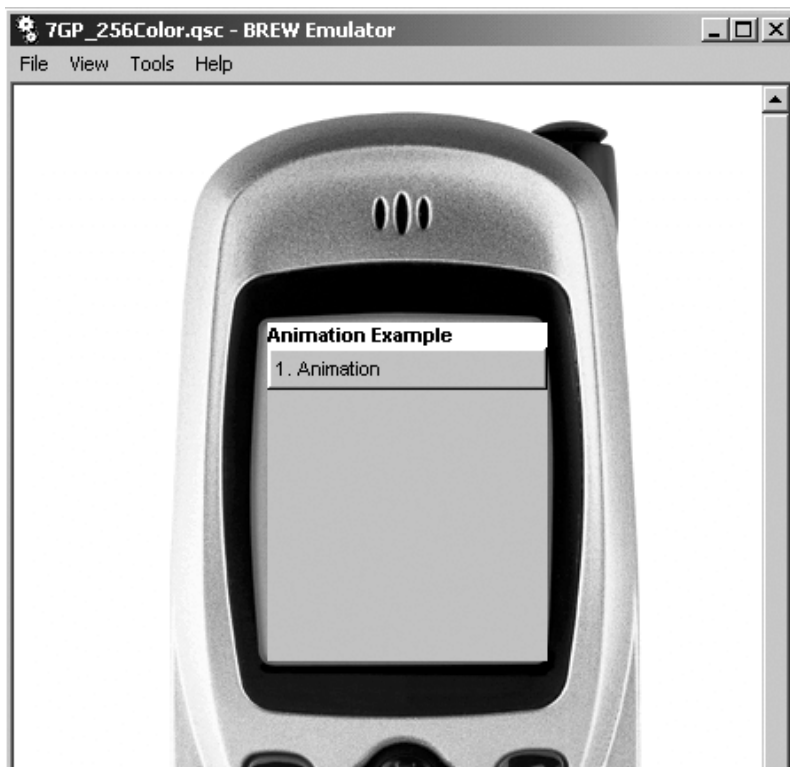


Figure 12-17: Emulator loaded with the Animation displaying the option

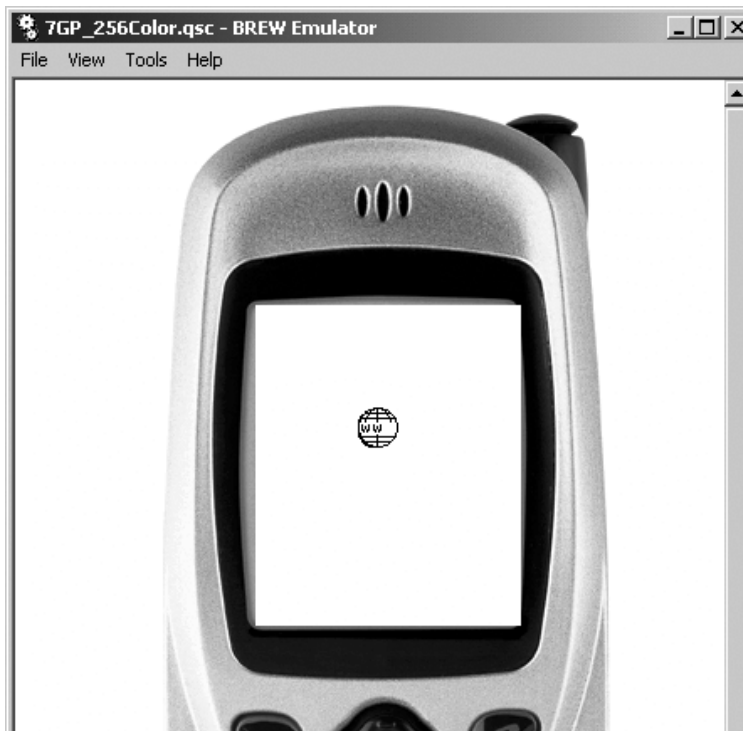


Figure 12-18: The Animation display

Application: Downloading Music onto a Mobile Device

In this example, we illustrate how a music file can be downloaded onto a mobile device. As soon as the 3G services are up and running, you can use your mobile phone to download music from a content provider and listen to it while waiting at an airport lounge. This application demonstrates how to accomplish this.

As usual, the first step is to create the BID file. To create the BID file go to Start⇒Programs⇒BREW⇒MIF Editor. The MIF Editor is used to create the BID file. The window in Figure 12-19 appears after the MIF Editor menu is clicked. This application is a new one, so click the New Applet button at the bottom of the window. The window in Figure 12-20 appears after the New Applet button is clicked.

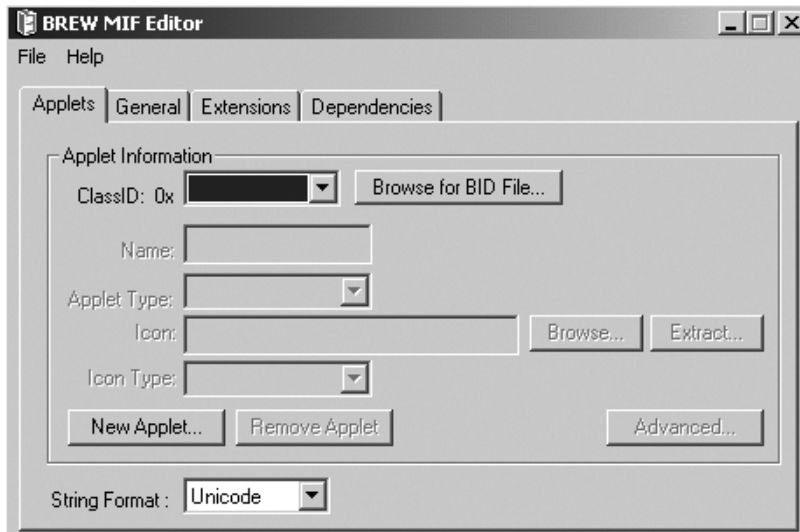


Figure 12-19: The MIF Editor window

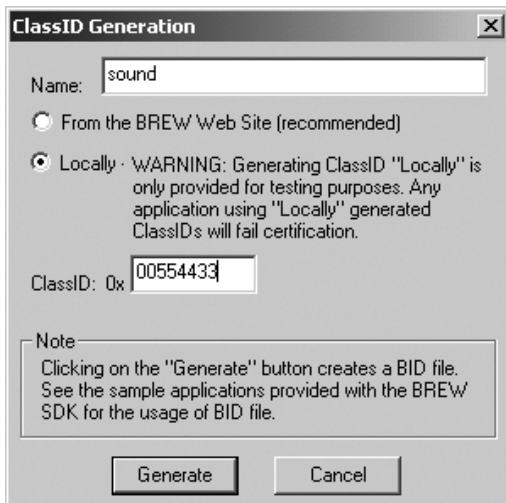


Figure 12-20: The New Applet ClassID Generation window

Enter the name and a unique ClassID of the applet. The window in Figure 12-21 appears after you click the Generate button.

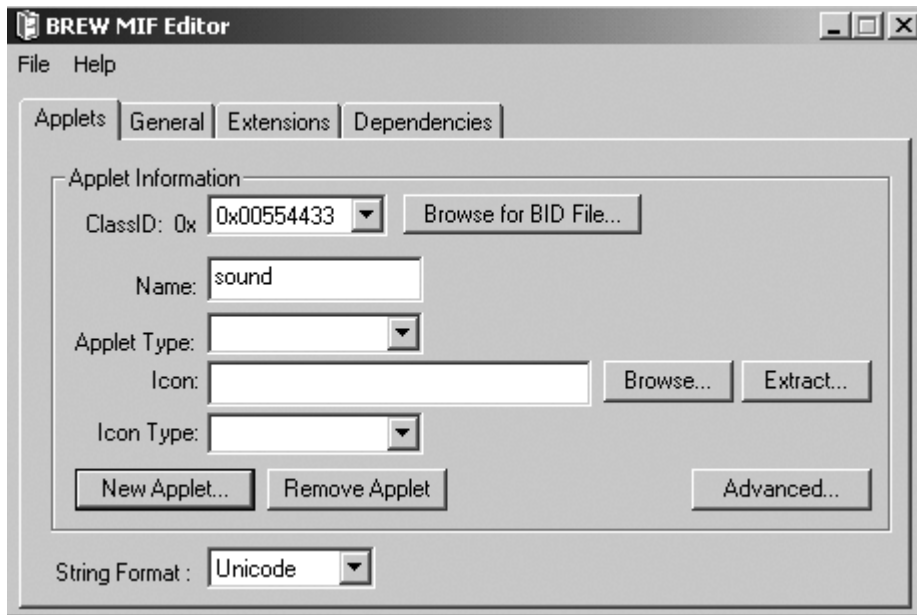


Figure 12-21: The MIF Editor Window after entering the ClassID

Enter the Name, Applet Type, Icon, and Icon Type in the appropriate fields. Click the File Save menu option and save the file in the applet directory or in a separate MIF folder.

As we are creating a new application, we will write the new code by opening the workspace and modifyin the C file. The sound application code is given in Listing 12-3.

Listing 12-3: Sound.C

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1.#include "AEModGen.h"
2.#include "AEEAppGen.h"
3.#include "AEEMenu.h"
4.#include "AEEStdLib.h"
5.#include "AEEUsageAppIDs.h"
6.#include "AEESound.h"
7.#include "AEESoundPlayer.h"
8.#include "AEEFile.h"
9.#include "sound.bid"
10.#define PLAY 103
11.#define STOP 104
12.#define REWIND 105
13.#define FASTFORWARD 106
14.#define PAUSE 107
15.#define RESUME 108
16.#define MIDI_MAX_FILES 10
17.#define TIME_INMILLISECONDS 5000 // 5000 ms
18.#define TEMPFACOR 300
19.#ifndef DIRECTORY_CHAR

```

```

20.#ifdef AEE_SIMULATOR
21.#define DIRECTORY_CHAR '\\\'
22.#define DIRECTORY_STR "\\\"
23.#else
24.#define DIRECTORY_CHAR '/'
25.#define DIRECTORY_STR "/"
26.#endif
27.#endif
28.typedef struct CISoundPlayerApp
29.{AEEApplet      a;
30.IMenuCtl      * ime;
31.ISoundPlayer  * iso;
32.char          * fname;
33.boolean       flag;
34.uint32        ptm;
35.int           lh;
36.AEEDeviceInfo info;
37.}spa;
38.static boolean eventHandle(IApplet * pi, AEEEvent aee,
    uint16 ui, uint32 dui);
39.static boolean initApp(IApplet* app);
40.static void freeApp(IApplet* app);
41.static void playerUsage (spa * app, uint16 ui);
42.static void mainMenu(spa *app);
43.int AEEClsCreateInstance(AEECLSID ClsId,IShell * ish,IModule * po,void
    ** obj)
44.{*obj = NULL;
45.if(ClsId == AEECLSID_SOUND){
46.if(AEEApplet_New(sizeof(spa), ClsId,
    ish,po,(IApplet**)obj,
47.(AEEHANDLER)eventHandle,(PFNFREEAPPDATA)freeApp) == TRUE) {
48.if (initApp((IApplet*)*obj) == TRUE){
49.return(AEE_SUCCESS);}}
50.return (EFAILED);}
51.static boolean eventHandle(IApplet * pi, AEEEvent aee,
    uint16 ui, uint32 dui){
52.spa * app = (spa*)pi;
53.if (app == NULL || app->a.m_pIShell == NULL || app->a.m_pIDisplay == NULL)
54.return FALSE;
55.switch (aee) {
56.case EVT_APP_START:
57.if(ISHELL_CreateInstance(app->a.m_pIShell, AEECLSID_MENUCTL, (void **)
    &app->ime) != SUCCESS){
58.return FALSE;}
59.mainMenu (app);
60.return(TRUE);
61.case EVT_APP_STOP:
62.return(TRUE);
63.case EVT_KEY:
64.if(app->ime != NULL && IMENUCTL_HandleEvent(app->ime, EVT_KEY,
    ui, 0))
65.return TRUE;
66.else
67.return FALSE;
68.case EVT_COMMAND:

```

```

69.switch(ui){
70.case PLAY:
71.case STOP:
72.case REWIND:
73.case FASTFORWARD:
74.case PAUSE:
75.case RESUME:
76.playerUsage (app, ui);
77.return TRUE;
78.default:
79.return FALSE;}
80.default:
81.break;}
82.return FALSE;}
83.static boolean initApp(IApplet* pi){
84.IFileMgr * pIFileMgr = NULL;
85.FileInfo fi;
86.int pnAscent = 0;
87.int pnDescent = 0;
88.char * szStart;
89.spa * app = (spa*)pi;
90.if (!app)
91.return FALSE;
92.app->ime = NULL;
93.app->iso = NULL;
94.app->ptm = 0;
95.app->flag = FALSE;
96.app->fname = NULL;
97.app->lh=IDISPLAY_GetFontMetrics(app->a.m_pIDisplay,
    AEE_FONT_NORMAL, &pnAscent, &pnDescent);
98.ISHELL_GetDeviceInfo(app->a.m_pIShell,&app->info);
99.if ( ISHELL_CreateInstance(app->a.m_pIShell, AEECLSID_FILEMGR, (void
    **)&pIFileMgr))
100.return FALSE;
101.IFILEMGR_EnumInit(pIFileMgr, "Music", FALSE);
102.IFILEMGR_EnumNext(pIFileMgr, &fi);
103.szStart = fi.szName; // STRCHR(fi.szName, (int)DIRECTORY_CHAR) + 1;
104.if ( szStart ){
105.app->fname = MALLOC(STRLEN(szStart) + 1);
106.STRCPY(app->fname, szStart);}
107.IFILEMGR_Release(pIFileMgr);
108.return TRUE;}
109.static void freeApp(IApplet* pi){
110.spa * app = (spa*)pi;
111.if (app->ime != NULL){
112.IMENUCTL_Release (app->ime);
113.app->ime = NULL;}
114.if (app->fname)
115.FREE (app->fname);}
116.static void playerUsage (spa * app, uint16 ui){
117.AEEDeviceInfo *di = NULL;
118.if (app == NULL || app->a.m_pIShell == NULL || app->a.m_pIDisplay == NULL)
119.return;
120.di = &app->info;
121.IDISPLAY_ClearScreen (app->a.m_pIDisplay);

```

```

122.switch (ui){
123.case PLAY:{
124.if (app->iso == NULL){
125.ISHELL_CreateInstance(app->a.m_pIShell, AEECLSID_SOUNDPLAYER, (void
    **)&app->iso);    }
126.ISOUNDPLAYER_Set(app->iso, SDT_FILE, app->fname);
127.ISOUNDPLAYER_Play (app->iso);    }
128.return;
129.case STOP:{    {
130.ISOUNDPLAYER_Stop (app->iso);    }    }
131.return;
132.case REWIND:    {
133.uint32 dwTime;    {
134.dwTime = TIME_INMILLISECONDS;    // in milliseconds
135.ISOUNDPLAYER_Rewind (app->iso, dwTime);    }    }
136.return;
137.case FASTFORWARD:    {
138.uint32 dwTime;    {
139.dwTime = TIME_INMILLISECONDS;    // in milliseconds
140.ISOUNDPLAYER_FastForward (app->iso, dwTime);    }
141.return;
142.case PAUSE:    {    {
143.ISOUNDPLAYER_Pause (app->iso);    }    }
144.return;
145.case RESUME:{    {
146.ISOUNDPLAYER_Resume (app->iso);    }    }
147.return;
148.default:
149.return; }
150.return;}
151.static void mainMenu(spa *app){
152.AEERect qrc;
153.AEEDeviceInfo di;
154.AECHAR szBuf[50];
155.int charHeight = 0;
156.int pnAscent = 0;
157.int pnDescent = 0;
158.if (app == NULL || app->ime == NULL || app->a.m_pIShell ==
    NULL)
159.return;
160.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
161.SETAEERECT (&qrc, 0, 0, di.cxScreen, di.cyScreen);
162.IMENUCTL_SetRect(app->ime, &qrc);
163.STR_TO_WSTR("SoundPlayer Menu:", szBuf, sizeof(szBuf));
164.IMENUCTL_SetTitle(app->ime, NULL, 0, szBuf);
165.STR_TO_WSTR("Play", szBuf, sizeof(szBuf));
166.IMENUCTL_AddItem(app->ime, 0, 0, PLAUI, szBuf, 0);
167.STR_TO_WSTR("Stop", szBuf, sizeof(szBuf));
168.IMENUCTL_AddItem(app->ime, 0, 0, STOP, szBuf, 0);
169.STR_TO_WSTR("Rewind (5 secs)", szBuf, sizeof(szBuf));
170.IMENUCTL_AddItem(app->ime, 0, 0, REWIND, szBuf, 0);
171.STR_TO_WSTR("FastForward (5 secs)", szBuf, sizeof(szBuf));
172.IMENUCTL_AddItem(app->ime, 0, 0, FASTFORWARD, szBuf, 0);
173.STR_TO_WSTR("Pause", szBuf, sizeof(szBuf));
174.IMENUCTL_AddItem(app->ime, 0, 0, PUSE, szBuf, 0);

```

```

175.STR_TO_WSTR("Resume", szBuf, sizeof(szBuf));
176.IMENUCTL_AddItem(app->ime, 0, 0, RESUME, szBuf, 0);
177.IMENUCTL_SetActive(app->ime, TRUE);}

```

Code Description

- ◆ Line 1: Header file. The `AEEModGen.h` file consists of `AEEModule` declaration.
- ◆ Line 2: Header file. The `AEEAppGen.h` file contains `AEEApplet` declaration.
- ◆ Lines 3–8: Header files as in the earlier examples.
- ◆ Line 9: The `BID` file, `sound.bid`.
- ◆ Lines 10–18: Defines the constants used in the application.
- ◆ Lines 19–27: Declaration statements.
- ◆ Lines 28–37: Code for the `IsoundPlayerApp` data structure. This structure holds the data members of the applet throughout the life of the applet.
- ◆ Line 38: The `handle_event` function declaration.
- ◆ Lines 39–42: Code for the application-specific functions.
- ◆ Lines 43–50: The `create_instance` method. This function is invoked while the applet is being loaded. The module must verify the `ClassID` and then invoke the `AEEApplet_New()` function that has been provided in `AEEAppGen.c`. After invoking `AEEApplet_New()`, this function can do applet specific initialization.
- ◆ Lines 51–82: The `handle_event` method. This method handles all the events of the application. The `pi` parameter is Pointer to the `AEEApplet` structure. This structure contains information specific to this applet. It is initialized during the `AEEClsCreateInstance()` function and code specifies the event sent to this applet.
- ◆ Lines 83–108: The `InitAppData` method. This function initializes applet-specific data, allocates memory for the data.
- ◆ Line 109–115: The `FreeAppData` method. This method frees data contained in applet and memory for the data.
- ◆ Lines 116–150: The `playerUsage` method. This function encompasses all the usage examples of all the functions in code blocks switched using the BREW API function `Id` passed into this function. The options are called using the switch case.
- ◆ Lines 151–177: The `build_player_menu` function. This function initially displays some text on the screen. Line 163 displays the text as a title of the screen. Lines 165, 167, 169, 171, 173, and 175 display the options on the screen. After the option is selected, the attached functionality will be performed.

Code Output

Figure 12-22 shows the display when the applet is loaded in the Emulator. The MIDI file will be played when the play option is selected. You can pause, rewind, or fast forward the music file using the given options.

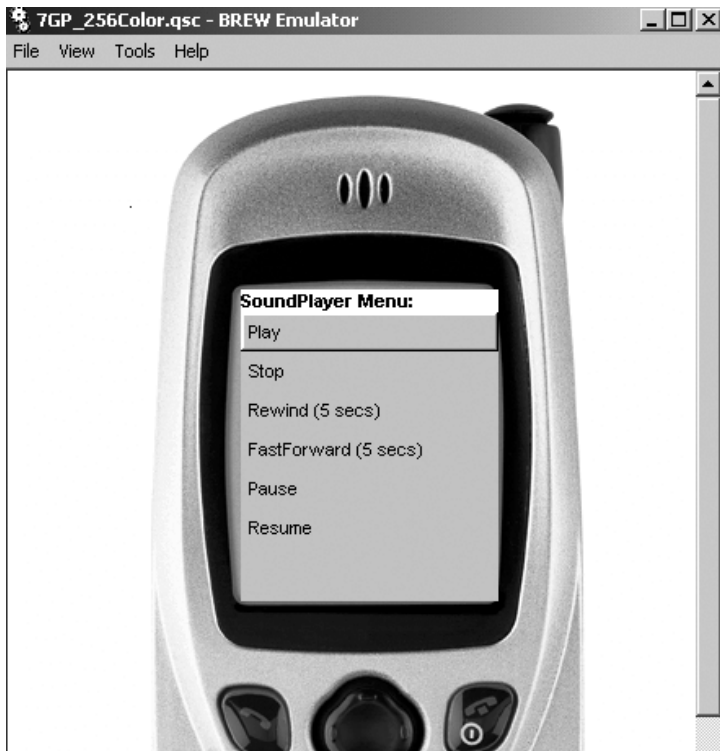


Figure 12-22: Display in the Emulator when Sound Application is loaded

Application: Mobile Advertisements

In this example, we create an application that displays a list of businesses. When a business is selected, the product range of that business is displayed. We follow the same steps as discussed in the earlier examples, however, here we will use the Dialogue Resources to create the dialogues required for our application.

The first step is to create the BID file. Go to Start⇒Programs⇒BREW⇒MIF Editor. The MIF editor is used to create the BID file. The window in Figure 12-23 appears after the MIF Editor menu is clicked.

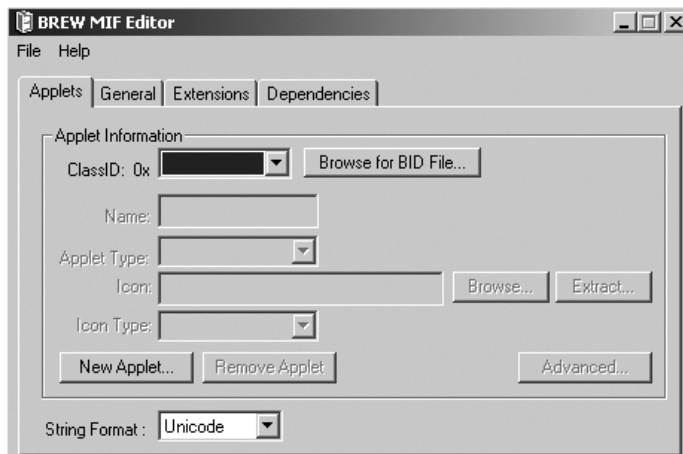


Figure 12-23: The MIF Editor window

The advertisement example is a new applet, so click the New Applet button at the bottom of the window. The window in Figure 12-24 appears after the New Applet button is clicked.

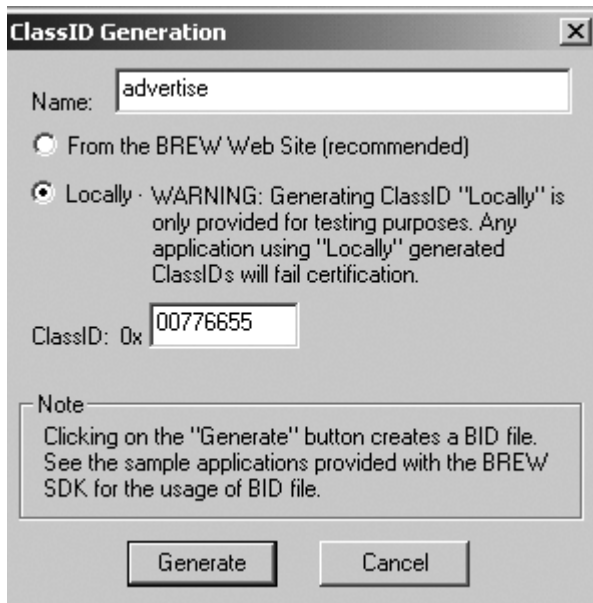


Figure 12-24: The new applet ClassID Generation window

Enter the name and ClassID of the applet and ensure that the ClassID name, BID filename, and MIF filename are the same. Click the Generate button at the bottom of the window and save the BID file. The window in Figure 12-25 appears after you click the Generate button.

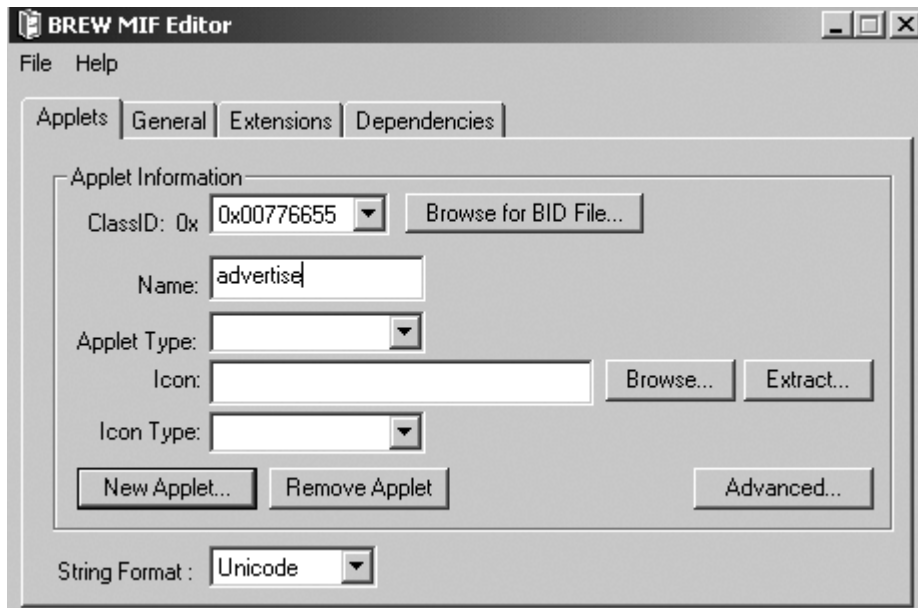


Figure 12-25: The MIF Editor window after entering the ClassID

Enter the Name, Applet Type, Icon, and Icon Type in the appropriate fields. Click the File Save button and save the file in the Applet directory or in a separate MIF folder.

Resource Editor

The application uses string and dialog resources. The Resource Editor has to be used to build the resources. To open the Resource Editor go to Start⇒Programs⇒BREW⇒Resource Editor. By clicking the menu the window in Figure 12-26 appears.

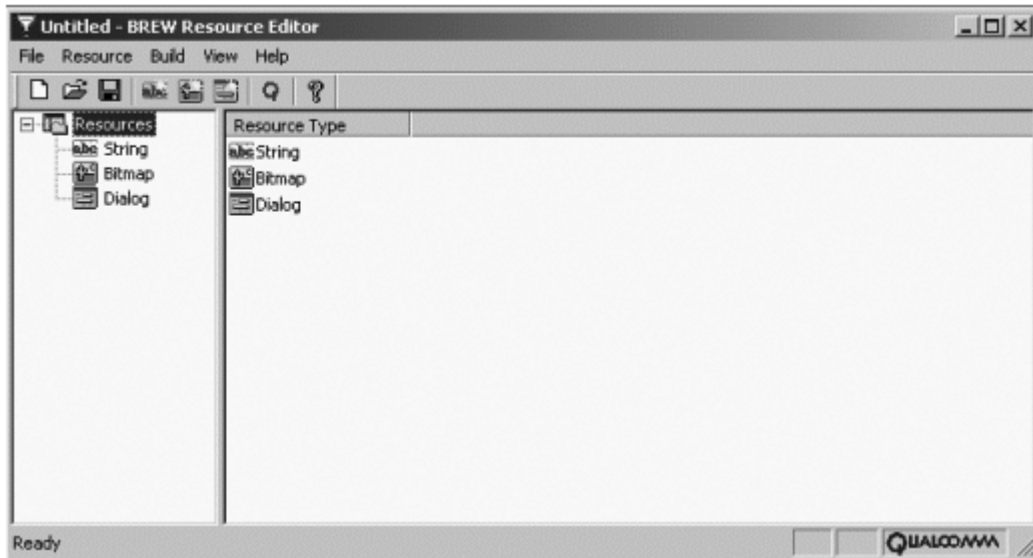


Figure 12-26: The BREW Resource Editor window

To create a string resource, go to Resource⇒New String or directly right-click on the string in the window. The window in Figure 12-27 appears.

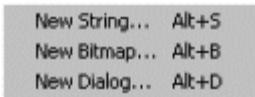


Figure 12-27: The Option window

Select New String. The window in Figure 12-28 appears.



Figure 12-28: The String Resource window

402 Chapter 12: 3G Programming Using BREW

Enter the Resource ID, Resource Name, String Type, and Value and click the OK button at the bottom of the window. A String resource is created. After you create the resources, the main Resource Editor window appears.

To create a Dialog Resource, go to Resource⇒New Dialog or directly right-click on Dialog in the window. The window in Figure 12-29 appears.

Enter the Resource ID, Resource Name, Dialog Title, Dialog Flags, Initial Focus ID, Initial Position X, Y, and Initial Size W, H and then select the type of control to use. Click the OK button at the bottom of the window after completing the previous procedure. A Dialog Resource is created.

After creating the resources, the main Resource Editor window appears. After all the Resources are created, click the Build button. The Build command creates the resource files, the Resource Header file (advertise_res.h), and a BAR file (advertise.bar). The header file has to be included in the application and the BAR file is defined as a resource file in the application.

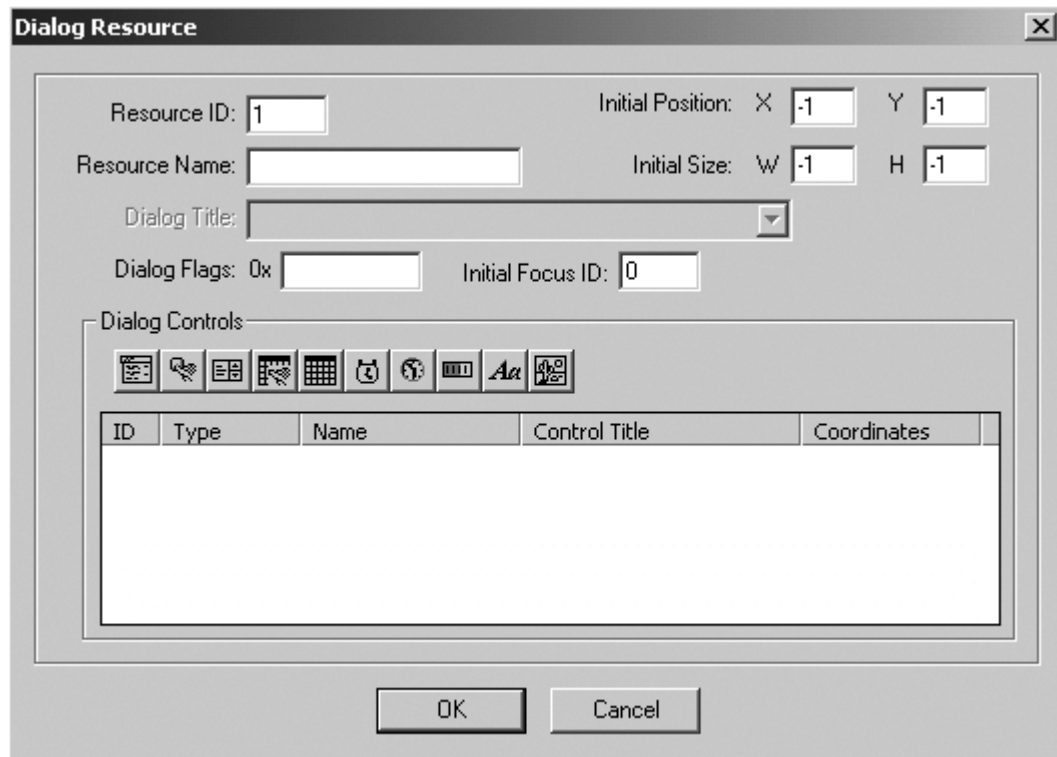


Figure 12-29: The Dialog Resource window

Copy the workspace of a sample application and write a new application using the code is given in Listing 12-4.

Listing 12-4: Advertise.C

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1.#include "AEEAppGen.h"
2.#include "AEEUsageAppIDs.h"
3.#include "AEE.h"
4.#include "AEEShell.h"
5.#include "AEEDisp.h"
```

```

6.#include "AEEStdLib.h"
7.#include "AEEFile.h"
8.#include "AEEMenu.h"
9.#include "AEEGraphics.h"
10.#include "AEEStdLib.h"
11.#include "advertise_res.h"
12.#include "advertise.bid"
13.typedef struct CIDialogApp {
14.AEEApplet          a;
15.IMenuCtl *   ime;
16.AEERect      rc;
17.} ida;
18.static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui);
19.static boolean initApp(IApplet* app);
20.static void freeApp(IApplet* app);
21.void mainMenu(ida *app);
22.boolean dailogUsage(ida *app, uint16 ui);
23.#define APP_RES_FILE      "advertise.bar"
24.#define SONY              102
25.#define KENWOOD           104
26.#define PANASONIC         105
27.#define MERCEDES           107
28.#define PHILIPS           109
29.#define BORDER_WIDTH 5
30.int AEEClsCreateInstance(AEECLSID ClsId, IShell* ish, IModule* po,
    void** obj)
31.{ *obj = NULL;
32.if(ClsId == AEECLSID_ADVERTISE){
33.if(AEEApplet_New(sizeof(ida), ClsId, ish,po,(IApplet**)obj,
34.(AEEHANDLER)eventHandle,(PFNFREEAPPDATA)freeApp) ==
    TRUE){ initApp((IApplet*)*obj);
35.return(AEE_SUCCESS);
36.} }
37.return (EFAILED);
38.}
39.static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui)
40.{ida * app = (ida*)pi;
41.switch (aee) {
42.case EVT_APP_START:
43.if(ISHELL_CreateInstance(app->a.m_pIShell, AEECLSID_MENUCTL,
44.(void **)&app->ime) != SUCCESS)
45.return TRUE;
46.mainMenu(app);
47.return(TRUE);
48.case EVT_APP_STOP:
49.if(app->ime != NULL){
50.IMENUCTL_Release(app->ime);
51.app->ime = NULL;}
52.return(TRUE);
53.case EVT_KEY:
54.switch (ui){
55.case AVK_UP:
56.case AVK_DOWN:

```

```

57.case AVK_SELECT:
58.if (ISHELL_GetActiveDialog(app->a.m_pIShell) != 0){
59.while (ISHELL_GetActiveDialog(app->a.m_pIShell) != 0)
60.ISHELL_EndDialog(app->a.m_pIShell);
61.IDISPLAY_EraseRect(app->a.m_pIDisplay, &app->rc);}
62.IMENUCTL_SetActive(app->ime,TRUE);
63.if(IMENUCTL_HandleEvent(app->ime, EVT_KEY, ui, 0))
64.return TRUE;
65.else return FALSE;
66.default:
67.if(!ISHELL_GetActiveDialog(app->a.m_pIShell)){
68.IMENUCTL_SetActive(app->ime,TRUE);
69.if (IMENUCTL_HandleEvent(app->ime, EVT_KEY, ui, 0))
70.return TRUE;
71.else return FALSE;}}
72.case EVT_COMMAND:
73.switch(ui){
74.case SONY:
75.case KENWOOD:
76.case PANASONIC:
77.case MERCEDES:
78.case PHILIPS:
79.IMENUCTL_SetActive(app->ime, FALSE);
80.dialogUsage (app, ui);
81.return TRUE;
82.default:
83.return FALSE;}
84.default:
85.break;}
86.return TRUE;}
87.static boolean initApp(IApplet* pi){
88.ida * app = (ida*)pi;
89.AEEDeviceInfo di;
90.if (app == NULL || app->a.m_pIShell == NULL)
91.return FALSE;
92.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
93.app->rc.x = BORDER_WIDTH;
94.app->rc.y = BORDER_WIDTH;
95.app->rc.dx = di.cxScreen - (2 * BORDER_WIDTH);
96.app->rc.dy = di.cyScreen - (2 * BORDER_WIDTH);
97.app->ime = NULL;
98.return TRUE;}
99.static void freeApp(IApplet* pi){
100.ida * app = (ida*)pi;
101.if(app->ime != NULL){
102.IMENUCTL_Release(app->ime);
103.app->ime = NULL;}}
104.boolean dialogUsage (ida * app, uint16 ui){
105.AEERect qrc;
106.AEEDeviceInfo di;
107.char szResFile[] = APP_RES_FILE;
108
109
110.AECHAR mbuf[] = {'T','A','T','A','B','E','N','Z','\0'};
111.AECHAR mbuf1[] = {'E','C','L','A','S','S','B','E','N','Z','\0'};
112.AECHAR probuf[] = {'P','R','O','D','U','C','T','S','\0'};

```

```

113.AECHAR sbuf[] = {'T','E','L','I','V','I','S','I','O','N','\0'};
114.AECHAR sbuf1[] = {'A','U','D','I','O','S','Y','S','T','E','M','\0'};
115.AECHAR sbuf2[] = {'H','A','N','D','I','C','A','M','\0'};
116.AECHAR sbuf3[] = {'H','I','F','I','S','Y','S','T','E','M','\0'};
117.if (app == NULL || app->a.m_pIShell == NULL || app->a.m_pIDisplay == NULL)
118.return FALSE;
119.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
120.qrc.x = 0;
121.qrc.y = 0;
122.qrc.dx = di.cxScreen;
123.qrc.dy = di.cyScreen;
124.IDISPLAY_EraseRect(app->a.m_pIDisplay,&qrc);
125.switch (ui){
126.case SONY:{
127.AECHAR title[20], text[100];
128.STR_TO_WSTR("SONY", title, sizeof(title));
129.STR_TO_WSTR("WELCOME TO THE WORLD OF SONY.",
130.text, sizeof(text));
131.ISHELL_MessageBoxText(app->a.m_pIShell, title, text);
132.ISHELL_EndDialog(app->a.m_pIShell);
133.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, probuf, -1,50,50,0,
IDF_TEXT_TRANSPARENT);
134.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, sbuf, -1,50,70,0,
IDF_TEXT_TRANSPARENT);
135.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, sbuf1, -1,50,80,0,
IDF_TEXT_TRANSPARENT);
136.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, sbuf2, -1,50,90,0,
IDF_TEXT_TRANSPARENT);
137.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, sbuf3, -1,50,100,0,
IDF_TEXT_TRANSPARENT);
138.IDISPLAY_Update (app->a.m_pIDisplay);
139.ISHELL_EndDialog(app->a.m_pIShell);}
140.break;
141.case KENWOOD:{
142.AECHAR title[20], text[100];
143.STR_TO_WSTR("KENWOOD", title, sizeof(title));
144.STR_TO_WSTR("WELCOME TO KENWOOD. THE MUSIC OF THE WORLD",
145.text, sizeof(text));
146.ISHELL_MessageBoxText(app->a.m_pIShell, title, text);
147.ISHELL_EndDialog(app->a.m_pIShell); }
148.break;
149.case PANASONIC:{
150.AECHAR title[20], text[100];
151.STR_TO_WSTR("PANASONIC", title, sizeof(title));
152.STR_TO_WSTR("WELCOME TO PANASONIC. THE LIFE IN ELECTRONICS",
153.text, sizeof(text));
154.SHELL_MessageBoxText(app->a.m_pIShell, title, text);
155.ISHELL_EndDialog(app->a.m_pIShell); }
156.break;
157.case MERCEDES:{
158.AECHAR title[20], text[100];
159.STR_TO_WSTR("MERCEDES", title, sizeof(title));
160.STR_TO_WSTR("WELCOME TO MERCEDES. THE DEFINITION OF CAR",
161.text, sizeof(text));
162.ISHELL_MessageBoxText(app->a.m_pIShell, title, text);
163.ISHELL_EndDialog(app->a.m_pIShell);

```

```

164.161.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, mbuf,
    -1,50,50,0, IDF_TEXT_TRANSPARENT);
165.161.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_BOLD, mbuf1,
    -1,50,60,0, IDF_TEXT_TRANSPARENT);
166.IDISPLAY_Update (app->a.m_pIDisplay);

167.ISHELL_EndDialog(app->a.m_pIShell); }
168.break;
169.case PHILIPS:{
170.AECHAR title[20], text[100];
171.STR_TO_WSTR("PHILIPS", title, sizeof(title));
172.STR_TO_WSTR("WELCOME TO PHILIPS . LETS MAKES THINGS BETTER",
173. text, sizeof(text));
174.ISHELL_MessageBoxText(app->a.m_pIShell, title, text);
175.ISHELL_EndDialog(app->a.m_pIShell); }
176.break;
177.default:
178.return FALSE;}
179.return TRUE;}
180.void mainMenu(ida *app){
181.AEERect qrc;
182.AEEDeviceInfo di;
183.AECHAR buf[100];
184.if (app == NULL || app->a.m_pIShell == NULL || app->ime == NULL)
185.return;
186.STR_TO_WSTR("Advertisements : press and Select the one you want", buf,
    sizeof(buf));
187.IMENUCTL_SetTitle(app->ime, NULL, 0, buf);
188.ISHELL_GetDeviceInfo(app->a.m_pIShell,&di);
189.qrc.x = 0;
190.qrc.y = 0;
191.qrc.dx = di.cxScreen;
192.qrc.dy = di.cyScreen;
193.IMENUCTL_SetRect(app->ime, &qrc);
194.STR_TO_WSTR("SONY", buf, sizeof(buf));
195.IMENUCTL_AddItem(app->ime, 0, 0, SONY, buf, 0);
196.STR_TO_WSTR("KENWOOD", buf, sizeof(buf));
197.IMENUCTL_AddItem(app->ime, 0, 0, KENWOOD, buf, 0);
198.STR_TO_WSTR("PANASONIC", buf, sizeof(buf));
199.IMENUCTL_AddItem(app->ime, 0, 0, PANASONIC, buf, 0);
200.STR_TO_WSTR("MERCEDES", buf, sizeof(buf));
201.IMENUCTL_AddItem(app->ime, 0, 0, MERCEDES, buf, 0);
202.STR_TO_WSTR("PHILIPS", buf, sizeof(buf));
203.IMENUCTL_AddItem(app->ime, 0, 0, PHILIPS, buf, 0);
204.IMENUCTL_SetActive(app->ime,TRUE);}

```

Code Description

- ◆ Line 1: Header file. The AEEAppGen.h file consists of AEEApplet declaration.
- ◆ Line 2: Header file. The AEEUsageAppIDs.h file contains ClassIDs of applet.
- ◆ Line 3: Header file. The AEE.h file contains the Standard AEE Declarations.
- ◆ Lines 4–10: Header files. The AEEShell.h contains AEE Shell Services. AEEDisp.h contains AEE Display Services. AEESTdLib.h contains AEE StdLib Services. AEEFile.h contains AEEFile Services. AEEMenu.h contains Menu Services. AEEGraphics.h contains Graphics Routines. AEESTdLib.h contains AEE stdlib services.

- ◆ Line 11: `animation_res.h` file is created by using the Resource Editor. This file contains the resources used by the applet.
- ◆ Line 12: `Animation.bid` file. This file contains the ClassID of the applet.
- ◆ Lines 13–17: Code for the data structure `IDialogApp`. This structure holds the data members of the applet throughout the life of the applet.
- ◆ Line 18: `Handle_event` function declaration.
- ◆ Lines 19–22: Code for applet-specific functions.
- ◆ Line 23: Defines the resource file
- ◆ Lines 24–29: Defines the application-specific constants.
- ◆ Lines 30–38: Create instance method. This function is invoked while the applet is being loaded. The module must verify the ClassID and then invoke the `AEEApplet_New()` function that has been provided in `AEEAppGen.c`. After invoking `AEEApplet_New()`, this function can do applet-specific initialization.
- ◆ Lines 39–86: The `handle_event` method. This method handles all the events of the applet. The parameter `pi` is pointer to the `AEEApplet` structure. The parameter `code` specifies the Event sent to this applet. In this case, all the events are handled using the switch statement.
- ◆ Lines 87–98: Code for the `InitAppData` method. This function initializes applet-specific data and allocates memory for the data.
- ◆ Line 99–103: Code for the `FreeAppData` method. This method frees data contained in the applet and memory for the data.
- ◆ Line 104–179: Code for the `dialogUsage` method. Lines 108–116 are for declaring the character buffers to store some data. Lines 120–123 are for declaring the coordinates to draw a rectangle. The switch case starts from line 125. Line 128 displays text as a title. Line 129 displays text as body, followed by the lines 133–137 for displaying text on the screen.
- ◆ Lines 180–204: Code for the build main menu method. This method is the first to display some text options on the screen in this application. Line 183 is to declare a character array of size 100. Line 186 puts some text on to the screen. Lines 189–192 are for defining coordinates. Lines 194, 196, 198, 200, and 202 put text on the screen.

Code Output

Figure 12-30 shows the initial display with the menu items when the applet is loaded on to the Emulator. Figure 12-31 shows the display when SONY menu item is selected.



Figure 12-30: Initial display in the screen when the applet is loaded

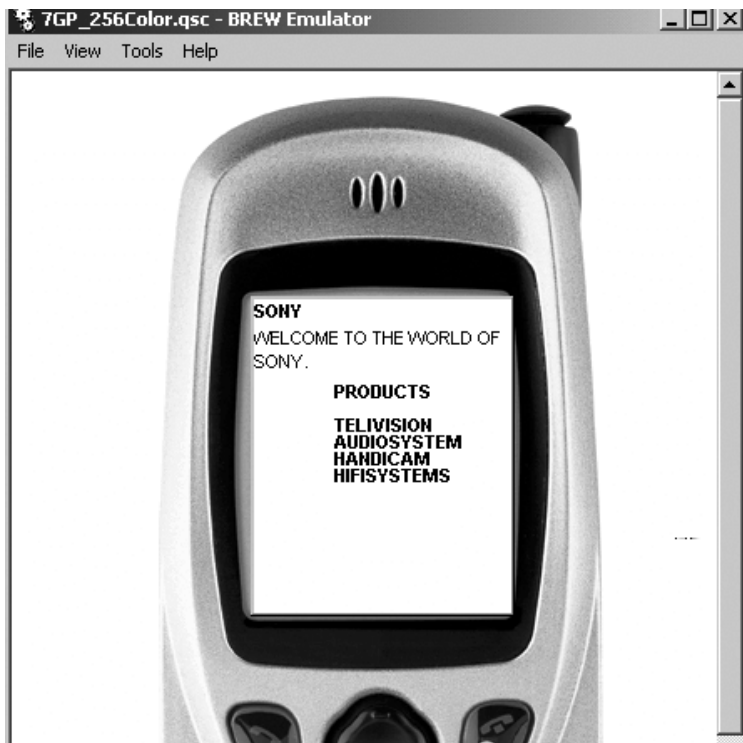


Figure 12-31: The display in the Emulator when the Sony option is selected

Application: Database

In this application, we create a database and a user interface on the mobile device to retrieve the information from the database. The code in Listing 12-5 creates an internal database and also provides the dialogues required for accessing the database. This internal database provides faster access to the data as compared to having an external database and accessing the data through the standards SQL commands. The basic code given here can be enhanced to create applications such as mobile commerce or accessing corporate databases.

Listing 12-5: Code for database application

© 2001 Dreamtech Software India Inc.

All Rights Reserved

```

1. #include "AEEModGen.h"
2. #include "AEEAppGen.h"
3. #include "AEEDB.h"
4. #include "AEEMenu.h"
5. #include "AEEStdLib.h"
6. #include "AEEUsageAppIDs.h"
7. #include "idbusage_res.h"
8. #define APP_RES_FILE "idbusage.bar"
9. #define OPENDATABASE 101
10. #define ADDRECORD 104
11. #define RETRIEVERECORD 107
12. #define UPDATE 110
13. #define REMOVE 111

14. #define SOFTKEY_MENU_HEIGHT 20
15. typedef struct CIDBApp
16. {
17.     AEEApplet a;
18.     IMenuCtl * ime;
19.     int lh;
20.     AEEDeviceInfo dinfo;
21. } dbap;
22. static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui);
23. static boolean initApp(IApplet* app);
24. static void freeApp(IApplet* app);
25. static void mainMenu(dbap *app);
26. static void dbUsage (dbap * app, uint16 ui);
27. static void display(dbap * app, int nline, char *pszStr);
28.
29. int AEEClsCreateInstance(AEECLSID ClsId, IShell * ish, IModule * po, void
    ** obj)
30. {
31.     *ibj = NULL;
32.     if (ClsId == AEECLSID_DATABASE_APP)
33.     {
34.         if (AEEApplet_New(sizeof(dbap), ClsId, ish, po, (IApplet**)ppObj,
35.             (AEEHANDLER)eventHandle, (PFNFREEAPPDATA)freeApp)
36.             == TRUE)
37.         {
38.             if (initApp((IApplet*)*obj) == TRUE)
39.             {
40.                 return(AEE_SUCCESS);

```

```

41. }
42.}
43.}
44.return (EFAILED);
45.}
46.static boolean eventHandle(IApplet * pi, AEEEvent aee, uint16
    ui, uint32 dui)
47. {
48. dbap * app = (dbap*)pi;
49. switch (aee)
50. {
51.     case EVT_APP_START:
52.         if(ISHELL_CreateInstance(app->a.m_pIShell, AEECLSID_MENUCTL, (void **)
            &app->ime)
53.             != SUCCESS)
54.         {
55.             return FALSE;
56.         }
57.         mainMenu(app);
58.         return(TRUE);
59.     case EVT_APP_STOP:
60.         return(TRUE);
61.     case EVT_KEY:
62.         if(app->ime != NULL && IMENUCTL_HandleEvent(app->ime, EVT_
            KEY, ui, 0))
63.             return TRUE;
64.         else
65.             return FALSE;
66.     case EVT_COMMAND:
67.         switch(ui)
68.         {
69.
70.         case OPENDATABASE:
71.         case ADDRECORD:
72.         case RETRIEVERECORD:
73.         case UPDATE:
74.         case REMOVE:
75.             dbUsage(app, ui);
76.             return TRUE;
77.         default:
78.             return FALSE;
79.         }
80.         default:
81.             break;
82.     }
83.     return FALSE;
84.}
85.static boolean initApp(IApplet* pi)
86.{
87.    int pnAscent;
88.    int pnDescent;
89.    dbap * app = (dbap*)pi;
90.    app->ime = NULL;
91.    app->lh = IDISPLAY_GetFontMetrics (app->a.m_pIDisplay,
        AEE_FONT_NORMAL,

```

```

92. &pnAscent, &pnDescent);
93. ISHELL_GetDeviceInfo(app->a.m_pIShell,&app->dinfo);
94. return TRUE;
95.}
96.static void freeApp(IApplet* pi)
97.{
98.dbap * app = (dbap*)pi;
99.if (app->ime != NULL)
100.{
101.IMENUCTL_Release (app->ime);
102.app->ime = NULL;
103.}
104.}
105.static void dbUsage (dbap * app, uint16 ui)
106.{
107.IShell *ish = app->a.m_pIShell;
108.IDISPLAY_ClearScreen (app->a.m_pIDisplay);
109.switch (ui)
110.{
111. case OPENDATABASE:
112.{
113.IDBMgr *idb = NULL;
114.IDatabase * ida = NULL;
115.boolean flag = FALSE;
116.ISHELL_CreateInstance(ish, AEECLSID_DBMGR, (void **)&idb);
117.if (idb == NULL)
118. return;
119.if ((ida = IDBMgr_OpenDatabase (idb, "db1", flag))
120. == NULL)
121. {
122. flag = TRUE;
123.if ((ida = IDBMgr_OpenDatabase (idb, "db1", flag))
124. != NULL)
125.{
126.display (app, -1, "Database open successful");
127.IDATABASE_Release (ida);
128.}
129.else
130.{
131. display (app, -1, "Database open failed");
132.}
133. }
134.else
135.{
136.display (app, -1, "Opened an already existing Database.");
137.IDATABASE_Release (ida);
138.}
139.IDBMgr_Release (idb);
140.}
141.break;
142.case ADDRECORD:
143.{
144. IDBMgr *idb = NULL;
145.IDatabase * ida = NULL;
146.IDBRecord *idbr = NULL, *ifbr1= NULL;

```

```

147.int nfields = 3;
148.AEEDBField field[3], field1[3];
149.uint32 phno = 1234567;
150.const char name [] = "Saidev";
151.const char address[] = "123 First Street, USA";
152.
153.uint32 phno1 = 1234568;
154.const char name1 [] = "hanuma";
155.const char address1[] = "124 Second Street, INDIA";
156.field[0].fName = AEEDBFIELD_FULLNAME;
157.field[0].fType = AEEDB_FT_STRING;
158.field [0].pBuffer = (void *)name;
159. field [0].wDataLen = STRLEN (name);
160. field [1].fName = AEEDBFIELD_ADDRESS;
161. field [1].fType = AEEDB_FT_STRING;
162. field [1].pBuffer = (void *)address;
163. field [1].wDataLen = STRLEN (address);
164. field [2].fName = AEEDBFIELD_HOME_PHONE;
165. field [2].fType = AEEDB_FT_DWORD;
166. field [2].pBuffer = (void *)&phno;
167. field [2].wDataLen = sizeof (uint32);
168. field1[0].fName = AEEDBFIELD_FULLNAME;
169. field1[0].fType = AEEDB_FT_STRING;
170. field1[0].pBuffer = (void *)name1;
171. field1[0].wDataLen = STRLEN (name);
172. field1[1].fName = AEEDBFIELD_ADDRESS;
173. field1[1].fType = AEEDB_FT_STRING;
174. field1[1].pBuffer = (void *)address1;
175. field1[1].wDataLen = STRLEN (address);
176. field1[2].fName = AEEDBFIELD_HOME_PHONE;
177. field1[2].fType = AEEDB_FT_DWORD;
178. field1[2].pBuffer = (void *)&phno1;
179. field1[2].wDataLen = sizeof (uint32);
180.ISHELL_CreateInstance(ish, AEECLSID_DBMGR, (void **)&idb);
181.if (idb == NULL)
182.return;
183.if ((ida = IDBMGR_OpenDatabase (idb, "db1", FALSE)) != NULL)
184.{
185.if ((idbr = IDATABASE_CreateRecord (ida, field,
    nfields))
186.!= NULL)
187.{
188.display (app, -1, "Create DB: successful");
189.
190.IDBRECORD_Release (idbr);
191.}
192.else
193.{
194.display (app, -1, "Create DB: Failed");
195.}
196.if ((idbr1 = IDATABASE_CreateRecord (ida, field1,
    nfields))
197.!= NULL)
198.{
199.display (app, -1, "Create DB: successful");

```

```

200.IDBRECORD_Release (idbr1);
201.}
202.else
203.{
204. display (app, -1, "Create DB: Failed");
205.}
206.IDATABASE_Release (ida);
207.}
208. IDBMgr_Release (idb);
209.}
210.break;
211.case RETRIEVERECORD:
212.{
213.IDBMgr      * idb = NULL;
214.IDatabase * ida = NULL;
215.IDBRecord * idbr1 = NULL;
216.IDBRecord * idbr2 = NULL;
217.char szBuf[50] = {0};
218.AEEDBFieldType ftype;
219.AEEDBFieldName fname;
220.uint16 flen;
221.byte * data = NULL;
222.int i,j=0;
223.uint32 rcount=0;
224.ISHELL_CreateInstance(ish, AEECLSID_DBMGR, (void **)&idb);
225.if (idb == NULL)
226. return;
227.if ((ida = IDBMgr_OpenDatabase (idb, "db1", FALSE)) == NULL)
228.{
229. IDBMgr_Release (idb);
230.return;
231.}
232. rcount =IDATABASE_GetRecordCount(ida);
233.for(i=0;i< rcount;i++){
234. idbr1 = IDATABASE_GetNextRecord (ida);
235.if (idbr1 != NULL)
236.{
237. ftype = IDBRECORD_NextField (idbr1, &fname, &flen);
238.data = IDBRECORD_GetField (idbr1, &fname, &ftype, &flen);
239.if (data != NULL)
240.{
241.SPRINTF (szBuf, "field 1: %s", (char *)data);
242.display (app, ++j, szBuf);
243.}
244. ftype = IDBRECORD_NextField (idbr1, &fname,
      &flen);
245.data = IDBRECORD_GetField (idbr1, &fname, &ftype, &flen);
246.if (data != NULL)
247.{
248.SPRINTF (szBuf, "field 2: %s", (char *)data);
249.display (app, ++j, szBuf);
250.}
251. ftype = IDBRECORD_NextField (idbr1, &fname, &flen);
252.data = IDBRECORD_GetField (idbr1, &fname, &ftype, &flen);
253.if (data != NULL)

```

```

254.{
255.SPRINTF (szBuf, "field 3: %d", (char *)data);
256.display (app, ++j, szBuf);
257.}
358.}
259.IDBRECORD_Release(idbr1);
260.}
261.IDATABASE_Release (ida);
262.IDBMGR_Release (idb);
263.}
264.break;
265.case UPDATE:
266.{
267.IDBMgr      * idb = NULL;
268.IDatabase * ida = NULL;
269.IDBRecord * idbr = NULL;
270.AEEDBField field [1];
271.int nRetVal = AEE_DB_EBADREC;
272.int nfields = 1;
273.const char lastName [] = "SaiDev";
274.char szBuf[30] = {0};
275.AEEDBFieldType ftype;
276.AEEDBFieldName fname;
277.byte * data = NULL;
278.uint16 flen;
279.ISHELL_CreateInstance(ish, AEECLSID_DBMGR, (void **)&idb);
280.if (idb == NULL)
281.return;
282.if ((ida = IDBMGR_OpenDatabase (idb, "db1", FALSE)) == NULL)
283.{
284.IDBMGR_Release (idb);
285.return;
286.}
287.idbr = IDATABASE_GetRecordByID (ida, 1);
288.ftype = IDBRECORD_NextField(idbr, &fname, &flen);
289.data = IDBRECORD_GetField (idbr, &fname, &ftype,
    &flen);
290. if (data != NULL)
291.{
292.SPRINTF (szBuf, "Field Val: %s", (char *)data);
293.display (app, 2, szBuf);
294.}
295. field [0].fName = AEEDBFIELD_LASTNAME;
296. field [0].fType = AEEDB_FT_STRING;
297. field [0].pBuffer = (void *)lastName;
298. field [0].wDataLen = STRLEN (lastName);
299.if ((nRetVal = IDBRECORD_Update (idbr, field, 1)) == SUCCESS)
300.{
301. ftype = IDBRECORD_NextField(idbr, &fname, &flen);
302.data = IDBRECORD_GetField (idbr, &fname, &ftype, &flen);
303.if (data != NULL)
304.{
305.SPRINTF (szBuf, "Updated Val: %s", (char *)data);
306.display (app, 4, szBuf);
307.}
308.}

```

```

309.else
310.{
311.    display (app, 4, "Update Failed.");
312. }
313. IDBRECORD_Release (idbr);
314.IDATABASE_Release (ida);
315.IDBMGR_Release (idb);
316. }
317. break;
318.case REMOVE:
319. {
320.IDBMgr      * idb = NULL;
321.IDatabase * ida = NULL;
322.IDBRecord * idbr1 = NULL;
323.uint32 rcount = 0;
324.char szBuf[30] = {0};
325. int nRetVal = AEE_DB_EBADREC;
326. ISHELL_CreateInstance(ish, AEECLSID_DBMGR, (void **)&idb);
327.     if (idb == NULL)
328.         return;
329.
330. if ((ida = IDBMGR_OpenDatabase (idb, "db1", TRUE)) == NULL)
331.     {
332.         IDBMGR_Release (idb);
333.         return;
334. }
335.rcount = IDATABASE_GetRecordCount (ida);
336.SPRINTF (szBuf, "No. of Records: %u", rcount);
337.display (app, 1, szBuf);
338. idbr1 = IDATABASE_GetRecordByID (ida, 1);
339.if ((idbr1 != NULL) && ((nRetVal = IDBRECORD_Remove (idbr1))
    == SUCCESS))
340.    {
341.        display (app, 2, "Removed Record 1");
342.    }
343. else
344.{
345. display (app, 2, "Failed removing record 1");
346. }
347. rcounr = IDATABASE_GetRecordCount (ida);
348. SPRINTF (szBuf, "No. of Records: %u", rcount);
349.     display (app, 5, szBuf);
350.     IDATABASE_Release (ida);
351.     IDBMGR_Release (idb);
352. }
353. break;
354.default:
355. return;
356. }
357.return;
358.}
359.static void mainMenu(dbap *app)
360.{
361.AEERect rc;
362.AECHAR szBuf[40];
363. if (app->ime == NULL || app->a.m_pIShell == NULL)

```



```

364. return;
365.STR_TO_WSTR("IDB Functions:", szBuf, sizeof(szBuf));
366.IMENUCTL_SetTitle(app->ime, NULL, 0, szBuf);
367. SETAEERECT (&rc, 0, 0, app->dinfo.cxScreen, app->dinfo.cyScreen);
368.IMENUCTL_SetRect(app->ime, &rc);

369.STR_TO_WSTR("2.  OpenDatabase", szBuf, sizeof(szBuf));
370.IMENUCTL_AddItem(app->ime, 0, 0, OPENDATABASE, szBuf, 0);
371.STR_TO_WSTR("5.  AddRecord", szBuf, sizeof(szBuf));
372.IMENUCTL_AddItem(app->ime, 0, 0, ADDRECORD, szBuf, 0);
373.STR_TO_WSTR("8.  RetrieveRecords", szBuf, sizeof(szBuf));
374.IMENUCTL_AddItem(app->ime, 0, 0, RETRIEVERECORD, szBuf, 0);
375.STR_TO_WSTR("11. UpdateRecord", szBuf, sizeof(szBuf));
376.IMENUCTL_AddItem(app->ime, 0, 0, UPDATE, szBuf, 0);
377.STR_TO_WSTR("12. RemoveRecord", szBuf, sizeof(szBuf));
378.IMENUCTL_AddItem(app->ime, 0, 0, REMOVE, szBuf, 0);
379.IMENUCTL_SetActive(app->ime, TRUE);
380.}

381.static void display(dbap * app, int nline, char *pszStr)
382.{
383. AECHAR * buff = NULL;
384.AECHAR * psz = NULL;
385.int pwidth;
386. AEFont font = AEE_FONT_NORMAL;
387. int pnFits = 0, dy;
388.int totalCh = 0;
389.if (app == NULL)
390.    return;
391.if ((buff = (AECHAR *)MALLOC(200)) == NULL)
392.return;
393.STR_TO_WSTR ((char *)pszStr, buff, 200);
394.    if (nline < 0) {
395.        dy = app->dinfo.cyScreen*2/5;
396.    }
397.else{
398.    dy = nline * app->lh;
399.    }
400.psz = buff;
401.totalCh = STRLEN ((char *)pszStr);
402.while ((totalCh > 0) && (*psz != NULL))
403.{
404.pwidth = IDISPLAY_MeasureTextEx(app->a.m_pIDisplay,
405.font,
406.(AECHAR *) psz,
407.-1,
408.app->dinfo.cxScreen - 5,
409.&pnFits);
410.if (pnFits == 0)
411.{
412.FREE (buff);
413.return;
414.}
415.IDISPLAY_DrawText(app->a.m_pIDisplay, AEE_FONT_NORMAL, psz, pnFits, 5 ,
416.dy, 0 , 0);
417.
418.psz += pnFits;

```

```

419.totalCh -= pnFits;
420.dy += app->lh;
421.
422.IDISPLAY_Update(app->a.m_pIDisplay);
423.if (totalCh < pnFits)
424.pnFits = totalCh;
425.}
426.FREE (buff);
427.return;
428.}

```

Code Description

- ◆ Lines 1–4: Header files for interface definition, applet interface definition, database interface definition, and menu interface definition.
- ◆ Lines 5–6: Header files for variable definitions.
- ◆ Line 7: Resource header file.
- ◆ Lines 8–14: Code for defining macros and constants.
- ◆ Lines 15–21: Defines Database applet structure. This is the main structure for this applet. This will hold all the data members that needs to be remembered throughout the life cycle of the applet. The first data member of this structure must be an AEEApplet OBJECT.
- ◆ Lines 22–27: Declarations of the function prototypes.
- ◆ Lines 29–45: CreateInstance() method. This function is invoked while the applet is being loaded. This section returns AEE_SUCCESS status upon loading the applet. The EFAILED is returned when loading is not successful. Line 32 verifies the ClassID. If it is true, line 34 invokes the AEEApplet_New() function and then InitAppData() is called to initialize AppletData .
- ◆ Lines 46–84: The HandleEvent() method. All events to this applet are handled in this function. It returns TRUE if the applet has processed the event, otherwise it returns FALSE. If it receives EVT_APP_START, it creates Imenu interface object in lines 52–56. If it successful, in line 57 mainMenu() function is invoked. When the user presses the End key, the applet receives EVT_APP_STOP event. When the user presses a key, the applet receives EVT_KEY. When the user selects a menu item, the applet receives EVT_COMMAND. If the user selects any one of the menu items (Opendatabase, Addrecord, retrieverecords, Remove, Update) the dbUsage() method is invoked in line 75.
- ◆ Lines 85–95: InitAppData() method. This function initializes appletspecific data, allocates memory for app data (AppletData) and sets it to pAppData of AEEApplet. This method returns TRUE if the allocation and initialization is successful, otherwise it returns FALSE. Line 89 initializes the MenuCtl pointer to NULL. Line 90–92 gets the font metrics information. Line 93 gets the device information.
- ◆ Lines 96–104: freeApp() method, which frees data and memory.
- ◆ Lines 105–359: dbUsage() method. This method is called when the user click any of the menu items and the menu ID is passed to this function as a parameter. IDBmgr object is created in line 113, IDatabase object is created in line 114. In lines 116–118 ISHELL_CreateInstance() method is invoked to create IDBmgr object. In lines 119–121 IDBMGR_Open Database() method is invoked to open existing database db1; if database does not exist, in lines 123–124 new database db1 is created and opened. Line 137 IDBMGR_Release() is invoked to release IDBmgr object. In line 139, IDBMGR_Release () is invoked to release IDatabase object.
- ◆ Lines 142–209: Explaining the functionality of adding records to the database.
- ◆ Lines 211–264: Eexplaining the functionality to retrieve the records.
- ◆ Lines 265–317: Explaining the functionality of updating an existing record.

- ◆ Lines 318–358: Explaining the functionality of removing an existing record.
- ◆ Lines 359–380: `mainMenu()` method. This method builds the main menu when this applet is started. In line 367, `IMENUCTL_SetTitle()` is invoked to set the title to the menu. In line 368, `IMENUCTL_SetRect()` method is invoked to set size for menu. In lines 370–378, `IMENUCTL_AddItem()` methods are invoked to set the menu items to the menu.
- ◆ Lines 381–428: Explaining `display()` method. This function displays an output string at a given line number on the screen. Line 391 is to allocate buffer to hold the string, and line 393 is to convert the string into Unicode. Lines 394–399 determines the starting location of the string on the screen. Line 400 `psz` keeps track of the point from where the next line should start.
- ◆ Line 401: For calculating the total string length, to decide whether wrapping is required.
- ◆ Lines 402–425: Keep displaying text string in multiple lines until the condition `((totalCh > 0) && (*psz != NULL))` is true. If the string cannot be accommodated in one line, the string will be split into two or more lines, with lines being 15 pixels apart from each other.

Code Output

When the previous application is run on the Emulator, the display is a menu list with options to Open Database, Add Record, Retrieve Records, Update Record, and Remote Record. If you select Open Database, the display shows “Opened an already existing database.” If the Retrieve Records option is selected, the display is as shown in Figure 12-32.

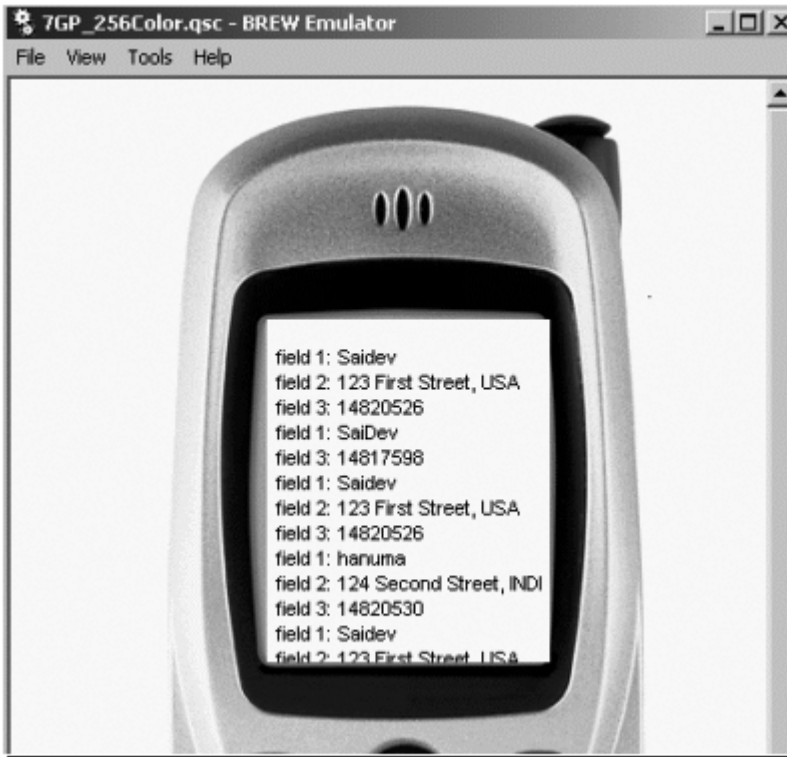


Figure 12-32: Display on the Emulator when 'Retrieve Records' menu item is selected

Summary

In this chapter, we discussed the implementation of wireless applications using BREW tool kit of Qualcomm. The applications, called applets, are created using C/C++ in Visual Studio environment. DLLs are created and loaded onto the Emulator to test the application before actual deployment in the field. We discussed the step-by-step procedure for creating a small application followed by applications for animation, music downloading, mobile advertising, and a database application.

BREW tool kit provides a good development environment for creating and testing wireless applications. Irrespective of the underlying interface, BREW can be used to develop applications on existing as well as future wireless networks.

Chapter 13

Voice and Video Communication over IP and Mobile IP Networks

In its early years, the Internet was extensively used for accessing information, which was mostly in text format. With the advent of the Web, multimedia content access has become the norm. However, the Web basically provides a one-way communication — the content stored in the Web server is transferred to the client. Web content can be a combination of text, graphics, audio, and video. Now, we are witnessing a revolution — to be able to transmit voice and video in real-time over the Internet and corporate Intranets. This is paving way for the *convergence*, meaning that we no longer need to use separate audio and video broadcasting networks — the Internet can be used for both audio and video broadcasting. Two-way and multi-party audio/video conferencing is also possible. However, special protocols are required for real-time transmission of audio and video over IP networks. The clients can run the normal TCP/IP as in the wired Internet, or they can run Mobile IP in the wireless Internet. In this chapter, we will briefly discuss the protocols and focus on implementation of real-time voice and video transmission over the IP networks. For the implementation, we use the Java Media Framework (JMF), the set of Application Programming Interfaces (APIs) released by Sun Microsystems for developing these applications.

Application of Voice and Video over IP

When we make telephone calls using a telephone network, the charges we incur vary according to the distance between the calling party and the called party because telephone networks use circuit switching. In circuit switching, a circuit is set up between the calling party and the called party, voice is transferred between the two terminals, and then the circuit is disconnected. On the other hand, when we use the Internet, we only pay for the connection charges for accessing the server of the Internet Service Provider (ISP), in addition to ISP subscription charges. The charges for using the Internet are not based on the distance between the client (the user terminal) and the server from which we are accessing the information. Thus, this “flat rate” charge is one of the major selling points for voice/video communication over the Internet. Making a long-distance call at the rate of a local call is economical by anyone’s standards. Major corporations that have Intranets spread over the country or across the continents can also drastically reduce their telecommunication costs by using voice/video communication over IP networks.

The various applications of voice/video over IP networks (Internet and Intranets) are

- ◆ Audio/voice mail
- ◆ Video mail for transmitting video clippings, for example, for offline lectures
- ◆ Audio broadcasting for applications, such as virtual class rooms, corporate presentations over the network, and broadcasting audio programs for the general public
- ◆ Video broadcasting for applications, such as virtual class rooms for online lectures, corporate presentations, and live video program broadcasting
- ◆ Two-way and multi-party audio and video conferencing

All these services can be provided at a fraction of the cost of a normal telephone network.

Protocols Overview

TCP/IP networks are not suitable for real-time transmission of audio or video because of the following:

- ◆ The IP does not provide a reliable transmission of the packets, meaning that the packets may not be received in the same order in which they were transmitted. Some packets may be lost and some packets may be duplicated (received more than once), and packets may arrive at the destination with different delays.
- ◆ To overcome unreliable transmission, the TCP layer has to take care of the previously mentioned problems. The TCP layer may ask for retransmission if the packets are lost, rearrange the packets if they are not received in sequence, and discard the packets which are duplicated. So, the TCP layer has to do lot of processing and processing requirements are very high. The TCP layer cannot do anything if all the packets do not have the same delay.
- ◆ For these reasons, it is not possible to ensure a required Quality of Service (QoS), which means that we cannot specify that the packets have to be delivered within a specified delay or the number of packets lost should be below a threshold, and so on. For voice/video transmission, this is an important requirement — the delay should be constant and packet loss should be minimal. Although QoS parameters can be specified in the TCP/IP network, you have no guarantee that these requests will be honored.

For real-time transmission of voice/video, the important requirement is that when packets are transmitted, they should be received at the destination with a constant delay. If there is variable delay, there will be breaks in speech. To reduce the processing requirements, for voice/video transmission over the IP networks, UDP (User Datagram Protocol) is used instead of TCP. Because UDP has less protocol overhead, it is better suited for real-time communication. Above UDP, two special protocols, RTP (Real Time Transport Protocol) and RTCP (Real Time Control Protocol) are used. The protocol suite for real-time transmission of voice/video over the IP networks is shown in Figure 13-1. In this figure, the IP can be IP version 4 or IP version 6, both of which run on fixed terminals connected to the Internet or a Mobile IP that runs on the mobile devices. Above the IP, UDP is used to take care of transport layer functionality. Above UDP, RTP provides the features for real-time transmission of voice/video along with RTCP. Above this layer, the audio/video application programs run.

Low Bit Rate Coding of Voice and Video

In the normal telephone network, speech is coded at 64 Kbps data rate using a technique called Pulse Code Modulation (PCM), which involves transmitting one second of speech. To store one second of speech 64 Kilobits are required. This is a very high data rate for present IP networks. So, to transmit voice over IP networks, in addition to PCM, low bit rate coding techniques are used, which reduce the data rates to 5.3 Kbps/6.3 Kbps. Similarly, for video applications, a 64 Kbps low data rate video-coding technique is used. As a result, the audio and video quality is not very good compared to broadcast quality audio or video, however, for normal business applications, the quality is good enough. For instance, using low bit rate video transmission for transmitting a dance program is not appealing to users as the user will feel many “jerks”; for business meetings where the image activity is less, low bit rate coding suffices.

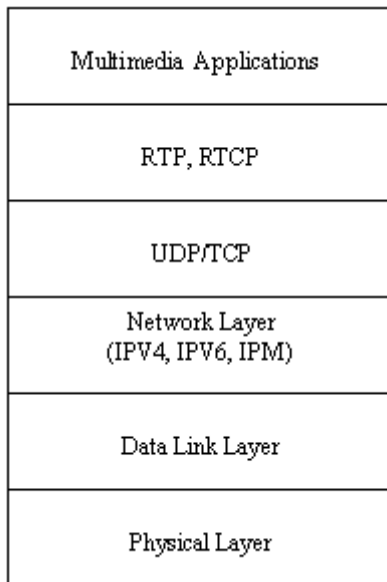


Figure 13-1: Protocol suite for voice and video over IP

H.323 Standards

International Telecommunications Union (ITU) has released the H.323 recommendations, which specify the standard protocols used for multimedia communication over IP networks. These standards do not guarantee the desired QoS. This protocol stack is shown in Figure 13-2. This protocol stack runs on the end terminals, called H.323 terminals, which can be PCs, laptops, and so forth. In addition to RTP and RTCP (mentioned earlier), H.323 recommendations also specify the standards for audio coding, namely, G.711 and G.723.1, and for video coding, namely, H.261 and H.263. To set up and disconnect calls over the IP networks, signaling is done through Q.931, which is the standard signaling used in ISDN (Integrated Services Digital Network).

The H.323 standards specify the communication protocols and the low bit rate coding techniques to be used for voice and video communication over the IP networks. The IP can be either the fixed IP (IP version 4 or version 6) or the Mobile IP that runs on the mobile devices. A number of vendors are incorporating a Mobile IP layer as a part of the network protocol suite that is bundled with the mobile Operating Systems. Presently, products are available that implement the H.323 protocol stack on fixed terminals such as PCs and H.323 telephones. Mobile H.323 devices are on the anvil, which provides an embedded solution of the H.323 protocol stack.

Mobile devices with H.323 support will provide killer applications to mobile subscribers: Subscribers will have access to voice and video services at a very low cost as compared to the fixed network services and existing mobile network services. The examples given in this chapter illustrate how to provide the H.323 capability to mobile devices. We assume here that the Mobile IP software will be running on these devices, and the devices have the Java Virtual Machine (JVM) running on them. With the support for Java Media Framework (JMF), the applications can be run directly on the mobile devices. However, it should be noted that in Mobile IP, the mobile device will have two addresses, a “home” address and a “care of” address. The network will send the packets to the home address; that address will, in turn, be forwarded to the mobile device to its point of attachment (the Mobile IP is discussed in Chapter 14). So, in the code given in this chapter, the IP address will be the home IP address. The other requirement of the mobile device is that it should have the capability for voice and video support. Mobile devices with small, integrated video cameras are already available, and video coding is done through software. Voice coding

is the other issue. JMF supports a number of coding techniques including the 13 Kbps coding technique used in GSM. So we need to choose the appropriate coding mechanism in the JMF based on the coding technique used in the mobile device.

The implementation examples discussed in this chapter can be tested on a LAN with the desktops as nodes or with laptops that provide wireless connectivity to the LAN. Because the implementation is in Java, the JVM has to be run on the nodes along with JMF. However, the best application of the code given here is to develop mobile H.323 terminals and to obtain the audio and video services over wireless 3G networks.

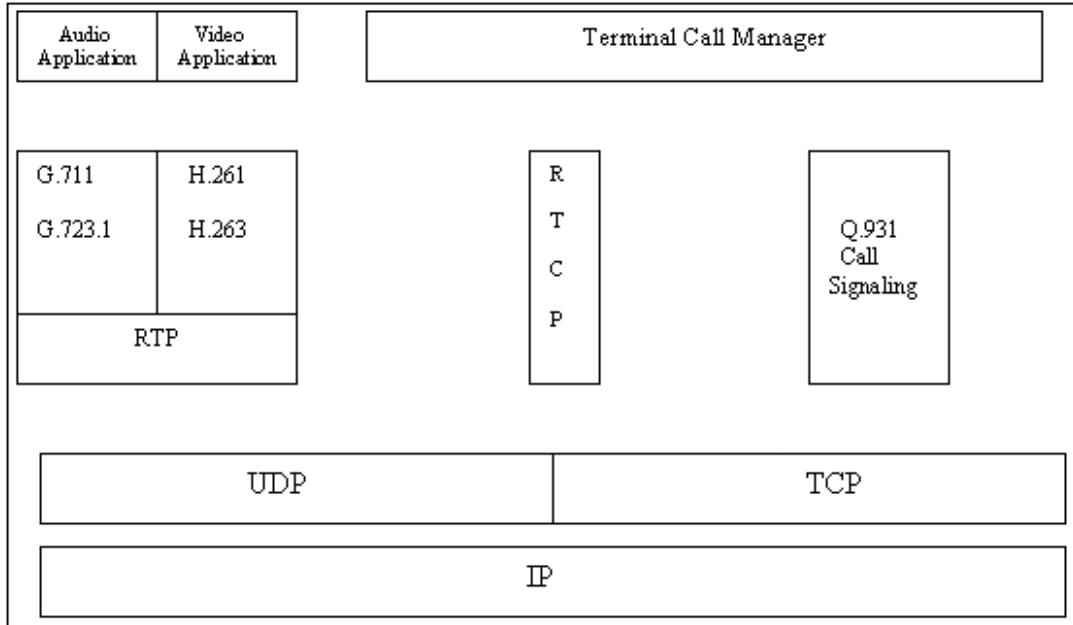


Figure 13-2: H.323 Terminal-side protocol stack

Java Media Framework

Sun Microsystems provides a set of Application Programming Interface (API) calls through Java Media Framework (JMF). The architecture of JMF is shown in Figure 13-3. JMF provides the necessary classes and methods to develop applications using the H.323 series of standards.

- ◆ The codes provide low bit rate coding of voice and video signals.
- ◆ The multiplexers facilitate combining the voice and video signals for transmission over the channel.
- ◆ The de-multiplexers do the reverse of multiplexers, dividing the audio and video to send them to the different devices for playback — audio to the sound card and video to the monitor.
- ◆ Renderers and effects allow for playback and manipulation of signals.
- ◆ Users can write their own APIs or use third party APIs through the JMF plug-in APIs.
- ◆ Java applications or applets can be created through the JMF presentation and processing APIs.

That is JMF in a nutshell. In the following sections, we demonstrate how interesting applications can be created using JMF. The complete code listings are given, and while explaining the code, we study the various classes and methods used for creating multimedia applications.

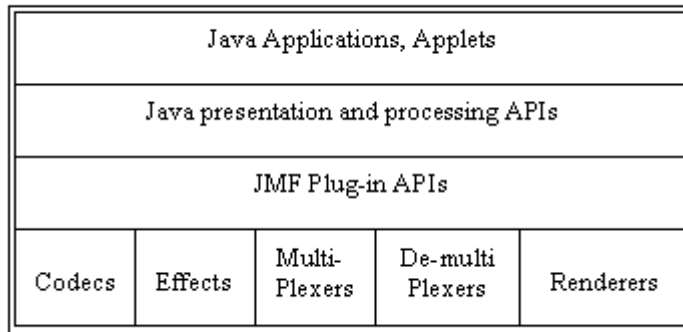


Figure 13-3: Architecture of JMF

Application Setup

To try the code given in the following examples, the following set-up is required:

- ◆ A Local Area Network with a Windows NT server and 95/98 clients.
- ◆ A few of the nodes on the LAN with multimedia capabilities:
 - A sound card with microphone and speakers
 - A desk-top video camera

Ensure that the corresponding drivers are installed and check the audio and video capabilities of the nodes before testing the code.

In the following sections, we discuss the implementation of three applications:

1. Development of the voice messaging application
2. Development of the audio broadcasting application
3. Development of the audio and video broadcasting application

These applications can be used for a variety of scenarios: for developing e-learning modules, for corporate presentations (live as well as offline), for sending e-mails with voice file attachments, and much more.

Application: Voice Messaging

This application is used to develop voice mail. Through implementation of this application, a user can create a voice message and transmit it over the network to another user just like a normal text mail. We need to develop two Java programs for this application: one is `AudioCapture.java`, which is for capturing the audio, and the other is `UserInter.java`, which is to create the necessary user interface. The hardware and software requirements are as follows:

Client side	
Development platform	JDK1.1 or Later

JMF API	2.1.1 or Later
Server side	
Web Server	Personal Web Server (PWS)

The code for the two programs is given in Listings 13-1 and 13-2. To run the application, you need to enable “capture from applets.” This can be done by using the JMFRegistry application and needs the swing library to run. If you are using JDK 1.2 or greater, you don’t have to do anything; but if you are running JDK 1.1.x, you need the swing jar file in the CLASSPATH. Invoke JMFRegistry and execute the followings steps:

1. Type the command **java JMFRegistry** at the command prompt.
2. Click User Settings
3. Click the Allow File Writing for Applets toggle.
4. Click the Allow Capture for Applets toggle.
5. Click the Commit button.
6. Quit JMFRegistry.

To give the security permissions for your current working directory files, you have to generate a policy file. To generate a policy file automatically, perform the following procedure:

1. Type the command **policytool** at your command prompt.
2. Click the Add Policy Entry button, which opens another window, Policy Entry.
3. Click the Add Permissions button, which opens another window, Permissions.
4. From the Permissions (first list box), choose File Permissions.
5. From the Target Name (second list box), choose <<ALL FILES>>.
6. From the Actions (third list box), choose ird, wdexecute.
7. Click OK.
8. Enter the values for CodeBase and SignedBy fields as follows:

```
CodeBase: file:///c:/mypro/vmail/      ("vmail" is my current working directory).
SignedBy: XXXXXX      (Your Name)
```

9. Choose File⇒Save as, browse your folder, and give a name for this policy file (for example c:\mypro\vmail\mypolicy).
10. Exit from the Policy tool.

The policy tool automatically generates the following code by doing the previous procedure:

```
/* AUTOMATICALLY GENERATED ON Sat Jun 02 15:13:47 GMT+05:30 2001*/
/* DO NOT EDIT */
grant {
    permission java.util.PropertyPermission "user.dir", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "java.class.path", "read";
    permission java.util.PropertyPermission "user.name", "read";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.misc";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.audio";
    permission java.lang.RuntimePermission "modifyThread";
    permission java.lang.RuntimePermission "modifyThreadGroup";
    permission java.lang.RuntimePermission "loadLibrary.*";
    permission java.io.FilePermission "<<ALL FILES>>", "read";
```

```

permission java.io.FilePermission "${user.dir}${/}jmf.log", "write";
permission java.io.FilePermission "${user.home}${/}.JMStudioCfg", "write";
permission java.net.SocketPermission "*", "connect,accept";
permission java.io.FilePermission "C:WINDOWSTEMP*", "write";
permission java.io.FilePermission "C:WINDOWSTEMP*", "delete";
permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
permission javax.sound.sampled.AudioPermission "record";
};
grant codeBase "file://c:/mypro" {
    permission java.io.FilePermission "<<ALL FILES>>", "read, write, delete,
execute";
};
grant codeBase "file://c:/mypro/vmail/" {
    permission java.io.FilePermission "<<ALL FILES>>", "read, write, delete,
execute";
};

```

Now perform the following steps.

1. You have to change the line numbers 104 and 109 in AudioCapture.java program (Listing 13-1) from "john1" to your server name or IP address. Here "john1" is the server name to store the voice mail in the following code.
2. Run the Personal Web Server (PWS) on your server and create a folder "mailboxes" in the personal Web server folder "Inetpub". The Webserver IIS (comes by default with Win2000) and other similar servers also support this program.
3. Create six folders in "inetpub\mailboxes\", with the names that are mentioned in the choice box of UserInter.java. (All these are user's mailboxes). For example, "Donald" folder is in \inetpub\mailboxes\, which is a mailbox for the user Donald.
4. Enter the source code of previous two files after making the changes in Step 3 and save them.
5. Compile two files, AudioCapture.java and UserInter.java (using javac)
6. Run the UserInter file using appletviewer with the following command: **appletviewer**. This command displays the screen in Figure 13-4.

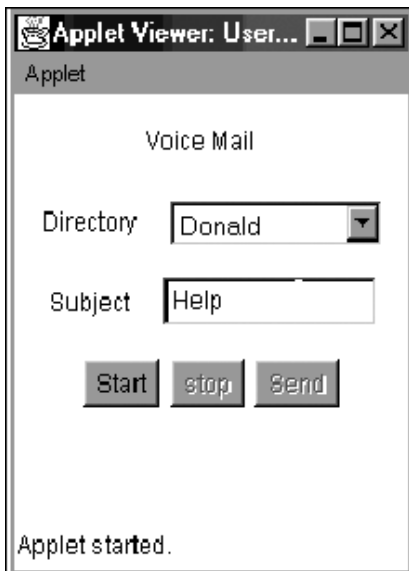


Figure 13-4: Applet Viewer window

The procedure for sending a voice mail is as follows: Select a name from the Directory list box, enter “Subject” in the Subject field, and click the Start button. Now you can speak into the microphone. If you click the Stop button, a temp file will be created, as `samp.wav` at the URL provided in line 65 (`file:///mypro/vmail/samp.wav`). If you click the Send button, the `samp.wav` will be copied to the server user’s mailbox that you have selected from the Directory list box.

For example, assume that you have selected the user name Donald from the Directory, entered Help in Subject field, and composed the voice message. The voice message is stored in a file at the server in Donald’s mailbox with the filename `<sender’s name>_<subject>_<filecount+1>.wav`. Here the sender’s name is taken as the client’s system name, for example, Susan. The subject is Help and the count is the number of files in the receiver’s folder. Therefore, if Donald’s mail box already contains 6 messages, the filename becomes `Susan_Help_7.wav`.

Listing 13-1 presents the code for `AudioCapture.java`.

Listing 13-1: AudioCapture.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved.

```

1.  import java.io.*;
2.  import java.net.*;
3.  import java.util.*;
4.  import java.awt.*;
5.  import java.awt.event.*;
6.  import javax.media.*;
7.  import javax.media.format.*;
8.  import javax.media.control.*;
9.  import javax.media.protocol.*;
10. import javax.media.rtp.*;
11. import javax.media.util.*;
12. import javax.media.rtp.event.*;
13.
14. public class AudioCapture implements Runnable, ControllerListener
15. {
16.     Processor p=null;
17.     Object waitObject;
18.     AudioFormat format;
19.     Vector devices;
20.     CaptureDeviceInfo di=null;
21.     DataSource source;
22.     DataSink filewriter;
23.     Thread t1;
24.     boolean realized,configured=true,prefetched,failed;
25.     boolean closed,eom,stoped;
26.     public AudioCapture()
27.     {
28.         t1=new Thread(this);
29.         t1.start();
30.     } // end of constructor
31.     public void run()
32.     {
33.         waitObject=new Object();
34.         format=new AudioFormat(AudioFormat.LINEAR,44100,16,1);
35.         devices=CaptureDeviceManager.getDeviceList(format);
36.         di=null;
37.         if(devices.size()>0)

```

```

38.         {
39.             di=(CaptureDeviceInfo)devices.elementAt(0);
40.         }
41.         else
42.         {
43.             System.out.println("Device not found");
44.             System.exit(0);
45.         }
46.         try{
47.             p=Manager.createProcessor(di.getLocator());
48.         }catch(IOException ie){System.out.println("p ioexception");}
49.         }catch(NoProcessorException ie){System.out.println("p
noprocessoexception");}
50.         p.configure();
51.         if(!waitForState(p.Configured))
52.         {
53.             System.out.println("no configured= "+configured);
54.         }
55.         else
56.             System.out.println("configured= "+configured);
57.         p.setContentDescriptor(new
58.         FileDescriptor(FileDescriptor.WAVE));
59.         p.realize();
60.         if(!waitForState(p.Realized))
61.         {
62.             System.out.println("Realized= "+configured);
63.         }
64.         else
65.             System.out.println("Realized= "+configured);
66.         source=p.getDataOutput();
67.         String url = "file://mypro/vmail/samp.wav";
68.         MediaLocator dest = new MediaLocator(url);
69.         DataSink filewriter = null;
70.         try {
71.             filewriter = Manager.createDataSink(source, dest);
72.         } catch (NoDataSinkException e) { e.printStackTrace();}
73.         catch (SecurityException e) { e.printStackTrace();}
74.         try {
75.             filewriter.open();
76.             filewriter.start();
77.         } catch (IOException e) { e.printStackTrace(); }
78.         p.start();
79.         int j=0;
80.         while(!stopped)
81.         {
82.             try{
83.                 waitObject.wait(1);
84.             } catch (Exception e) {}
85.             j++;
86.         }
87.         p.close();
88.         filewriter.close();
89.         System.out.println("started");
90.     } // end of run

public void stop()

```

```

91.  {
92.      stoped=true;
93.  } // end of stop
94.
95.  public void sendMail()
96.  {
97.  try{
98.      InetAddress localhost=InetAddress.getLocalHost();
99.      String s=localhost.toString();
100.     s=s.substring(0,s.indexOf("/"));
101.     s=s.trim();
102.     System.out.println("Sender: "+s);
103.
104.     File filecount=new File("//john1/mailboxes/"+UserInter.username);
105.     String files[]=filecount.list();
106.     int count=files.length;
107.     System.out.println("Total number of mails in
"+UserInter.username+"'s mailbox "+count);
108.
109.     File remotefile=new
File("//john1/mailboxes/"+UserInter.username+"/"+s+"_"+UserInter.subject+"_"+cou
nt+".wav");
110.     FileOutputStream fout = new FileOutputStream(remotefile);
111.     FileInputStream ii=new FileInputStream ("c://mypro//vmail//samp.wav");
112.     byte b[]=new byte[ii.available()];
113.     int i=ii.read(b);
114.     fout.write (b);
115.     System.out.println("Your mail has been sent");
116.     }catch(Exception e){ e.printStackTrace(); }
117.     } // end of sendMail
118.
119.  public boolean waitForState(int state)
120.  {
121.      synchronized (waitObject)
122.      {
123.          try
124.          {
125.              while (p.getState() < state && configured)
126.              {
127.                  waitObject.wait(1);
128.              }
129.              waitObject.notifyAll();
130.          } catch (Exception e) {}
131.      }
132.      return configured;
133.  } // end of waitForState
134.
135.  public synchronized void controllerUpdate(ControllerEvent ce)
136.  {
137.      if (ce instanceof RealizeCompleteEvent)
138.      {
139.          configured = true;
140.          synchronized (waitObject)
141.          {
142.              try{

```

```

143.         waitObject.notifyAll();
144.     }catch (Exception e) {}
145.     }
146. }
147. else if (ce instanceof ConfigureCompleteEvent)
148. {
149.     configured = true;
150.     synchronized (waitObject)
151.     {
152.         try{
153.             waitObject.notifyAll();
154.         } catch (Exception e) {}
155.     }
156. }
157. else if (ce instanceof PrefetchCompleteEvent)
158. {
159.     prefetched = true;
160. }
161. else if (ce instanceof EndOfMediaEvent)
162. {
163.     eom = true;
164. }
165. else if (ce instanceof ControllerErrorEvent)
166. {
167.     failed = true;
168. }
169. else if (ce instanceof ControllerClosedEvent)
170. {
171.     closed = true;
172. }
173. else if (ce instanceof ResourceUnavailableEvent)
174. {
175.     configured=false;
176. }
177. else
178. {
179.     return;
180. }
181. waitObject.notifyAll();
190. } // end of controllerUpdate
191.
192. public static void main(String args[])
193. {
194.     new AudioCapture();
195. } // end of main method
196. } // end of AudioCapture

```

Code Description

- ◆ Line 1: `java.io` package, which provides support for I/O operations.
- ◆ Line 2: `java.net` package, which provides support for networking.
- ◆ Line 3: `java.util` package contains enhancements added by Java 2 collections. Note: A *collection* is a group of objects.

- ◆ Line 4: `java.awt` package contains classes and methods that allow you to create and manage windows, manage fonts, output text, and utilize graphics.
- ◆ Line 5: `java.awt.event` package contains the classes to handle events generated by the mouse, the keyboard, and various controls, such as a push button.
- ◆ Line 6: `javax.media` package contains interfaces, such as `Controller`, `ControllerListener`, `Processor`, `Player` and classes, such as `CaptureDeviceInfo`, `CaptureDeviceManager`, `Format`, `Manager`, and `MediaLocator`.
- ◆ Line 7: `javax.media.format` package contains classes for JMF-supported media formats.
- ◆ Line 8: `javax.media.control` package contains classes for controlling the bit rate, frame rate, buffers, tracks, and quality of media.
- ◆ Line 9: `javax.media.protocol` package contains interfaces and classes for file type descriptor, content descriptor, and creating the data source from which you have to capture the data.
- ◆ Line 10: `javax.media.rtp` package contains interfaces and classes for data transmission through real-time transport protocol.
- ◆ Line 11: `javax.media.util` package contains classes to convert a video Buffer object to an AWT Image object that you can render using AWT methods. It can also convert an AWT Image object to a JMF Buffer object.
- ◆ Line 12: `javax.media.rtp.event` package, contains the classes to handle events generated by Receive Streams, Send Streams, Sessions, and Time out. These are all related to the RTP protocol.
- ◆ Line 13: Empty line.
- ◆ Line 14: User-defined Class `AudioCapture` starts at line 14 and ends at line 196, which implements the `Runnable` interface for using threads and the `ControllerListener` interface for handling asynchronous events generated by controllers,. These are in the `javax.media` package.
- ◆ Lines 16–25: Variables declaration.
- ◆ Lines 26–30: Constructor of `AudioCapture`; line 28 creates a thread and invokes `run` method by calling the `start` method in line 29.
- ◆ Lines 31–88: `run()` method.
- ◆ Line 34: For creating an audio format instance with the `AudioFormat` class.
- ◆ Line 35: Gets a list of `CaptureDeviceInfo` objects that correspond to devices that can capture data in the specified format.
- ◆ Line 39: `CaptureDeviceInfo` object contains information about a particular capture device and returns this information to variable `di`, which is of type `CaptureDeviceInfo`.
- ◆ Line 46: To create a processor for the specified media using `createProcessor` method, which is in `Manager` class, by passing the `MediaLocator`.
- ◆ Lines 49–63: To configure and realize the processor.
- ◆ Line 64: To get the output `DataSource` from the processor
- ◆ Line 65: Create a URL for storing the temp file in the local system.
- ◆ Line 66: Create a destination media locator with the URL specified in line 65.
- ◆ Line 69: Create a data sink, using the method `createDataSink`, which is in `Manager` class, with the source and destinations.
- ◆ Lines 72–76: Open data sink to write the data from the data source.
- ◆ Line 85: To close the processor.
- ◆ Line 86: To close the data sink.

- ◆ Lines 90–93: `stop()` method to stop the process, capturing, and writing to the data sink.
- ◆ Lines 95–117: `sendMail()` method to send the temporary file from the local system to receiver's mail box.
- ◆ Lines 98–102: To get the sender's system name from the `InetAddress`.
- ◆ Lines 104–107: To count the total number of files (e-mails) in the receiver's mailbox.
- ◆ Lines 109–114: Copy the `samp.wav` file from sender's system to server system's receiver's mailbox.
- ◆ Lines 119–133: `waitForState()` method to configure the processor.
- ◆ Lines 135–190: `controllerUpdate()` method, which is method in `ControllerListener` interface. This method is called when an event is generated by a controller that this listener is registered with.
- ◆ Lines 192–195: `main()` method to create an instance of class `AudioCapture`.

Listing 13-2 presents the code for `UserInter.java`.

Listing 13-2: UserInter.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1.  import java.io.*;
2.  import java.awt.*;
3.  import java.applet.*;

4.  import java.awt.event.*;
5.
6.  /*****
7.   To run this file use the following command:
8.   appletviewer UserInter-J-           please correct this statement
Djava.security.policy = c:\mypro\vamil\mypolicy
9.   *****/
10.
11. /* <applet code="UserInter" width=200 height=180> </applet> */
12.
13. public class UserInter extends Applet implements ActionListener
14. {
15.     Button start, stop, send;
16.     AudioCapture as;
17.     static String username, subject;
18.     Choice chdirectory;
19.     TextField tfsubject;
20.     Label labeldirectory, labelsubject, header;
21.     public void init()
22.     {
23.         header= new Label("Voice Mail");
24.         labeldirectory =new Label("Directory");
25.         chdirectory=new Choice();
26.         labelsubject =new Label("Subject");
27.         tfsubject=new TextField(12);
28.         start=new Button("Start");
29.         stop=new Button("stop");
30.         send=new Button("Send");
31.         chdirectory.addItem("Donald");
32.         chdirectory.addItem("John");

```

```

33.         chdirectory.addItem("Mary");
34.     chdirectory.addItem("Susan");
35.         chdirectory.addItem("Henry");
36.         chdirectory.addItem("Bill");
37.         stop.setEnabled(false);
38.         send.setEnabled(false);
39.         start.addActionListener(this);
40.         stop.addActionListener(this);
41.         send.addActionListener(this);
42.         Panel p0 = new Panel();
43.         Panel p1 = new Panel();
44.         Panel p2 = new Panel();
45.         Panel p3 = new Panel();
46.         p0.add(header);
47.         p1.add(labeldirectory);
48.         p1.add(chdirectory);
49.         p2.add(labelsubject);
50.         p2.add(tfsubject);
51.         p3.add(start);
52.         p3.add(stop);
53.         p3.add(send);
54.         add(p0);
55.         add(p1);
56.         add(p2);
57.         add(p3);
58.         setSize(200,200);
59.         setVisible(true);
60.     }
61.
62.     public void actionPerformed(ActionEvent ae)
63.     {
64.         if (ae.getSource()==start)
65.         {
66.             as=new AudioCapture();
67.             stop.setEnabled(true);
68.             start.setEnabled(false);
69.         } // end of if for start button
70.         if (ae.getSource()==send)
71.         {
72.             username = chdirectory.getSelectedItem().trim();
73.             subject = tfsubject.getText().trim();
74.             as.sendMail();
75.         } // end of if for send button
76.         if(ae.getSource()==stop)
77.         {
78.             stop.setEnabled(false);
79.             start.setEnabled(true);
80.             send.setEnabled(true);
81.             as.stop();
82.         } // end of if for stop button
83.     } // end of actionPerformed()
84. } // end of UserInter

```

Code Description

- ◆ Line 1: `java.io` package that provides support of I/O operations.

- ◆ Line 2: `java.awt` package contains classes and methods to create and manage windows, manage fonts, output text, and utilize graphics.
- ◆ Line 3: `java.applet` package; contains the `Applet` class and three interfaces `AppletContext`, `AudioClip`, and `AppletStub`. These applet methods provide detailed control over the execution of the applet.
- ◆ Line 4: `java.awt.event` package contains the classes to handle events generated by the mouse, the keyboard, and various controls, such as a push button.
- ◆ Line 11: `<applet>` tag, for executing this class file using `appletviewer`.
- ◆ Lines 13–84: User-defined class `UserInter` that extends `java.applet.Applet` class and implements the interface `ActionListener`.
- ◆ Lines 15–20: Variables declaration.
- ◆ Lines 21–60: Applet life cycle invokes the `init()` method when applet starts. In this `init()` method, we add the components choice box (directory), buttons (Start, Stop, Send), and a text box (Subject) to the applet; and add action listener to the buttons.
- ◆ Lines 62–83: `actionPerformed()` method that is invoked when an action occurs.

Application: Audio Broadcasting

In this application, we develop programs to do audio broadcasting over a network. The application consists of one module that runs on the server where audio broadcasting is done and one module that runs on the clients that receive the broadcast. The server module consists of two programs:

`AudioSendStreams.java` for transmitting audio and `Professor.java` that creates the user interface. The client module also consists of two programs: `AudioReceiveStreams.java` to receive the audio and `Student.java` to create the user interface.

The source code of these programs is given in Listings 13-3, 13-4, 13-5, and 13-6, respectively. The procedure for executing this application is as follows:

1. Enter each of these programs and compile them; the `Professor.class` file is created.
2. Run the Professor program at the server using the command **java Professor**.
3. A screen will appear that has Start, Stop, and Exit buttons. When you click the Start button, the audio is captured by the microphone and is transmitted to whomever is logged on to the server.
4. At the client-side, when you compile the code, `Student.class` file is created.
5. Execute the Student program using the command: **java Student**. A screen will appear with Start and Exit buttons.
6. After you click the Start button, you will hear the audio broadcast.

Listing 13-3: AudioSendStreams.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. import java.io.*;
2. import java.net.*;
3. import java.awt.*;
4. import java.util.*;
5. import javax.media.*;
6. import java.awt.event.*;
7. import javax.media.rtp.*;
8. import com.sun.media.ui.*;
9. import com.sun.media.rtp.*;
```

```

10. import javax.media.format.*;
11. import javax.media.control.*;
12. import javax.media.protocol.*;
13. import javax.media.rtp.rtcp.*;
14. import javax.media.rtp.event.*;
15.
16. public class AudioSendStreams implements Runnable, ControllerListener
17. {
18.     Processor p=null;
19.     int port=49150;
20.     Object waitObject;
21.     AudioFormat format;
22.     Vector devices,playerlist;
23.     String HostName;
24.     CaptureDeviceInfo di=null;
25.     Thread t1;
26.     boolean realized,configured=true,prefetched,failed,closed,eom,stoped;
27.     boolean encodingOk;
28.
29.     public AudioSendStreams(String Name)
30.     {
31.         HostName=Name;
32.         System.out.println("AudioSendStream-Name="+Name);
33.         playerlist=new Vector();
34.         t1=new Thread(this);
35.         t1.start();
36.     }
37.     public void run()
38.     {waitObject=new Object();
39.         format=new AudioFormat(AudioFormat.LINEAR,44100,16,1);
40.         devices=CaptureDeviceManager.getDeviceList(format);
41.         di=null;
42.         if(devices.size()>0)
43.         {   di=(CaptureDeviceInfo)devices.elementAt(0);
44.             }
45.         else
46.         {System.out.println("AudioSendStream- Device not found");
47.             System.exit(-1);
48.         }
49.         try
50.         {   p=Manager.createProcessor(di.getLocator());
51.             } catch(IOException ie)
52.             {   System.out.println("p ioexception");
53.             } catch(NoProcessorException ie)
54.             {   System.out.println("p noprocessorexception");
55.             }
56.             p.configure();
57.         if(!waitForState(p.Configured))
58.         {System.out.println("AudioSendStream-no configured= "+configured);
59.         }
60.         p.setContentDescriptor(new ContentDescriptor(ContentDescriptor.RAW_RTP));
61.         TrackControl track[] = p.getTrackControls();
62.         boolean encodingOK=false;
63.         for(int i=0;i<track.length;i++)
64.         {   if(!encodingOk && track[i] instanceof FormatControl)

```

```

65.         { if(((FormatControl)track[i]).setFormat(new
                                   AudioFormat(AudioFormat.GSM_RTP,8000,8,1))==null)
66.             { track[i].setEnabled(false);
67.             }
68.         else
69.             { encodingOk=true;
70.             }
71.         }else
72.             track[i].setEnabled(false);
73.     }
74.
75.     if(encodingOk)
76.     {p.realize();
77.     if(!waitForState(p.Realized))
78.     {System.out.println("AudioSendStream-Realized= "+configured);
79.     }
80.     SessionManager mgr=new com.sun.media.rtp.RTPSessionMgr();
81.
82.     if (mgr == null) System.exit(-1);
83.     mgr.addFormat(new AudioFormat(AudioFormat.GSM_RTP,8000,8,1),18);
84.     String cname = mgr.generateCNAME();
85.     String username = null;
86.     try
87.     { username = System.getProperty("user.name");
88.     } catch (SecurityException e)
89.     { username = "user";
90.     }
91.
92.     SessionAddress localaddr = new SessionAddress();
93.     try
94.     { InetAddress destaddr = InetAddress.getByName(HostName);
95.     SessionAddress sessaddr = new
96.         SessionAddress(destaddr,port,destaddr,port + 1);
97.     SourceDescription[] userdesclist= new SourceDescription[]
98.     { new SourceDescription(SourceDescription.SOURCE_DESC_EMAIL,username+
99.         "@company.com",1,false),
100.     new SourceDescription(SourceDescription.SOURCE_DESC_CNAME,cname,1,false),
101.     new SourceDescription(SourceDescription.SOURCE_DESC_TOOL,"JMF RTP Player
102.         v2.0",1,false)
103.     };
104.     mgr.initSession(localaddr,userdesclist,0.5,0.25);
105.     mgr.startSession(sessaddr,1,null);
106.     } catch (Exception e)
107.     { System.err.println(e.getMessage());
108.     }
109.     DataSource source=p.getDataOutput();
110.     p.start();
111.     try
112.     { SendStream s= mgr.createSendStream(source,0);
113.     s.start();
114.     } catch (IOException ie)
115.     { ie.printStackTrace(); }
116.     catch (UnsupportedFormatException upe)

```

```

124.         {         upe.printStackTrace();         }
125.     int j=0;
126. } /* End of IF Condition */
127. } /* End of Run Method */
128.
129. public void stop()
130. {stoped=true;         }
131.
132. public boolean waitForState(int state)
133. {synchronized (waitObject)
134.     {   try
135.         {   while (p.getState() < state && configured)
136.             { waitObject.wait(1);
137.             }
138.         waitObject.notifyAll();
139.         }   catch (Exception e) {}
140.     }
141.     return configured;
142. }
143.
144. public synchronized void controllerUpdate(ControllerEvent ce)
145. {if (ce instanceof RealizeCompleteEvent)
146.     {   configured = true;
147.         synchronized (waitObject)
148.         {   try
149.             {   waitObject.notifyAll();
150.             }   catch (Exception e) {}
151.         }
152.     }
153.     else if (ce instanceof ConfigureCompleteEvent)
154.     {   configured = true;
155.         synchronized (waitObject)
156.         {   try
157.             {   waitObject.notifyAll();
158.             }   catch (Exception e) {}
159.         }
160.     }
161. }
162.     else if (ce instanceof PrefetchCompleteEvent)
163.     {   prefetched = true;   }
164.     else if (ce instanceof EndOfMediaEvent)
165.     {   eom = true;   }
166.     else if (ce instanceof ControllerErrorEvent)
167.     {   failed = true;   }
168.     else if (ce instanceof ControllerClosedEvent)
169.     {   closed = true;   }
170.     else if (ce instanceof ResourceUnavailableEvent)
171.     {   configured=false;   }
172.     else
173.     {   return;   }
174. } /* End of controllerUpdate */
175.
176. class RTPPlayerWindow extends PlayerWindow
177. {public RTPPlayerWindow(Player player, String title)
178.     {   super(player);
179.         setTitle(title);

```

```

180.     }
181.     public void Name(String title)
182.     {         setTitle(title);             }
183.     }
184.     public static void main(String args[])
185.     {new AudioSendStreams("255.255.255.255"); }
186. }

```

Code Description

- ◆ Line 1: Package for the input, output stream classes.
- ◆ Line 2: Package for networking-related classes.
- ◆ Line 3: Package for the frames, panels, and related classes.
- ◆ Line 4: This is an automatic-imported package; at compile time, this package extracts all related classes belonging to this package.
- ◆ Line 5: This package is for time-based media.
- ◆ Line 6: Package for event handlers.
- ◆ Line 7: Package that provides APIs for playback and transmission of RTP streams.
- ◆ Line 8: This package constructs all instances of `com.sun.media.ui.*` at compilation time.
- ◆ Line 9: This package constructs an instance of `com.sun.media.rtp.RTPSessionMgr`. `RTPSessionMgr` is an implementation of `SessionManager` provided with the JMF reference implementation.
- ◆ Line 10: This package provides all audio and video format classes available in JMF.
- ◆ Line 11: This package provides the all Controller interfaces in JMF.
- ◆ Line 12: This package is a Type-Import-on-Demand declaration (it extracts the classes from this application at compile time).
- ◆ Line 13: `javax.media.rtp.rtcp` provides support for RTP.
- ◆ Line 14: This package is the reorganized RTP package. The reorganization consists of the following changes: The RTP event classes that were in `javax.media.rtp.session` are now in `javax.media.rtp.event`.
- ◆ Line 16: Beginning of `AudioSendStreams` class.
- ◆ Line 17: The `Processor` interface defines a module for processing and controlling time-based media data. `Processor` extends the `Player` interface.
- ◆ Line 19: Indicates port number as 49150 for transmitting audio streams
- ◆ Lines 20–28: The necessary variables are declared in this code.
- ◆ Lines 29–36: Creation of the constructor for `AudioSendStreams`.
- ◆ Lines 37–127: This code is the `Run` method to send audio streams to the clients.
- ◆ Lines 42–48: Checking for Capturing Device. If the device is not found, all the streams go to `CaptureDeviceInfo`. Otherwise, it returns "AudioSendStream-Device not found" and exits from the loop.
- ◆ Lines 49–55: Try block for creating a processor for the specified media. This `createProcessor` method returns a string value.
- ◆ Line 60: To set the output content descriptor to `RAW_RTP`, this line will limit the supported formats reported from `Track.getSupportedFormats` to only valid RTP formats.

- ◆ Line 61: To get the tracks from the Processor, by using the `getTrackControls` method and a `TrackControl` interface for each track in the media stream. This method can only be called once after `Processor` interface has been configured.
- ◆ Lines 63–82: Program for setting the track lengths
- ◆ Line 102: Creation of instance of the local session address (assigned to variable name called "localaddr").
- ◆ Lines 103–115: Try block for the `InetAddress` class represents an IP Address. In this application, we used the method `getByName` to create a new `InetAddress` instance.
- ◆ Lines 106–110: Here the `SourceDescriptor` constructs the type of source description. This is description of the actual source and frequency.
- ◆ Lines 129–130: For stopping the transmission.
- ◆ Lines 132–142: Synchronization method for `waitForState` with one parameter, State of the processor. This `waitForState` method returns a Boolean value.
- ◆ Lines 144–174: Synchronization method, the `controllerUpdate` for different types of events, such as completion of configuration of the processor, reaching of end of media, and so on.
- ◆ Lines 176–183: Creates a class for `RTPPlayerWindow`, which extends the `PlayerWindow` class.
- ◆ Lines 184–186: Creating an instance of `AudioSendStreams`.

Listing 13-4: Professor.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. import java.awt.*;
2. import java.awt.event.*;
3.
4. public class Professor extends Frame implements ActionListener
5. { Button start, stop, exit;
6.   AudioSendStreams as;
7.   public Professor()
8.   {setLayout(new FlowLayout());
9.     start=new Button("Start");
10.    stop=new Button("stop");
11.    exit=new Button("Exit");
12.    stop.setEnabled(false);
13.    start.addActionListener(this);
14.    stop.addActionListener(this);
15.    exit.addActionListener(this);
16.    Panel p1= new Panel();
17.    Panel p2= new Panel();
18.    Panel p3= new Panel();
19.    p1.add(start);
20.    p2.add(stop);
21.    p3.add(exit);
22.    add(p1);
23.    add(p2);
24.    add(p3);
25.    setSize(100,100);
27.    setVisible(true);
28. }
29. public void actionPerformed(ActionEvent ae)

```



```

30. {if (ae.getSource()==start)
31.   {as=new AudioSendStreams("255.255.255.255");
32.   stop.setEnabled(true);
33.   start.setEnabled(false);
34. }else if (ae.getSource()==exit)
35. {System.exit(0);
36. }else
37.   {   stop.setEnabled(false);
38.       start.setEnabled(true);
39.       as.stop();
40.   }
41. }
42. public static void main(String[] args)
43. {new Professor();      }
44. }

```

Code Description

- ◆ Line 1: Package for frames, panels, and related classes.
- ◆ Line 2: Package for event handlers.
- ◆ Line 4: Beginning of the main `Professor` class.
- ◆ Lines 7–28: In this constructor, buttons are initialized. To write event handlers on the buttons, we use the `addActionListener` method provided in `ActionListener` interface. Through `addActionListener` method, we can directly register the components into `ActionListener` interface.
- ◆ Lines 9–11: Initializing the `Start`, `Stop`, and `Exit` variables through a button class.
- ◆ Lines 13–15: Registering the buttons to the `Listener` interface through `addActionListener` method.
- ◆ Lines 16–18: Initialization of the panels.
- ◆ Lines 19–21: Adding the buttons to the panels.
- ◆ Lines 22–24: Adding panels to a frame.
- ◆ Line 25: Setting size of the frame.
- ◆ Lines 29–41: This is an `actionPerformed` method to write the event handling for `Start`, `Stop`, and `Exit` buttons.
- ◆ Lines 42–43: Creation of an instance of the `Professor` class.
- ◆ Line 44: End of the `Professor` class.

Listing 13-5: AudioReceiveStreams.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4. import java.awt.*;
5. import java.awt.event.*;
6. import javax.media.*;
7. import javax.media.format.*;
8. import javax.media.control.*;
9. import javax.media.protocol.*;

```

```

10. import javax.media.rtp.*;
11. import javax.media.rtp.event.*;
12. import javax.media.rtp.rtcp.*;
13. import com.sun.media.ui.*;
14. import com.sun.media.*;
15. public class AudioReceiveStreams implements Runnable,
16. ControllerListener, ReceiveStreamListener
17. { Processor p=null;
18.
19. int port=49150;
20. Object waitObject;
21. AudioFormat format;
22. Vector devices, playerlist;
23. String HostName;
24. CaptureDeviceInfo di=null;
25. DataSource source;
26. DataSink filewriter;
27. Thread t1;
28. boolean realized, configured=true, prefetched, failed, closed, eom, stoped;
29. boolean encodingOk;
30. public AudioReceiveStreams(String Name)
31. { HostName=Name;
32. System.out.println("AudioReceiveStream-Name="+Name);
33. playerlist=new Vector();
34. t1=new Thread(this);
35. t1.start();
36. }
37. public void run()
38. { waitObject=new Object();
39. format=new AudioFormat(AudioFormat.LINEAR, 44100, 16, 1);
40. devices=CaptureDeviceManager.getDeviceList(format);
41. di=null;
42. if(devices.size()>0)
43. { di=(CaptureDeviceInfo)devices.elementAt(0);
44. }
45. else { System.out.println("Device not found");
46. System.exit(-1);
47. }
48. try
49. { p=Manager.createProcessor(di.getLocator());
50. } catch(IOException ie)
51. { System.out.println("p ioexception");
52. } catch(NoProcessorException ie)
53. { System.out.println("p noprocessorexception");
54. }
55. p.configure();
56. if(!waitForState(p.Configured))
57. { System.out.println("AudioTransmitStream-no configured= "+configured);
58. }
59. p.setContentDescriptor(new ContentDescriptor(ContentDescriptor.RAW_RTP));
60. TrackControl track[] = p.getTrackControls();
61. boolean encodingOK=false;
62. for(int i=0; i<track.length; i++)
63. { if(!encodingOk && track[i] instanceof FormatControl)
64. { if(((FormatControl)track[i]).setFormat(new

```

```

        AudioFormat(AudioFormat.GSM_RTP,8000,8,1))==null)
65.     {    track[i].setEnabled(false);
66.     }else

67.         encodingOk=true;
68.     }else
69.         track[i].setEnabled(false);

70.     }
71.     if(encodingOk)
72.     {    p.realize();
73. if(!waitForState(p.Realized))
74. {System.out.println("AudioTransmitStream- Realized= "+configured);
75. }

76.         source=p.getDataOutput();
77.     SessionManager rtpsm=new com.sun.media.rtp.RTPSessionMgr();
78.     SessionManager mgr= rtpsm;
79.     if (mgr == null) System.exit(-1);
80. mgr.addFormat(new AudioFormat(AudioFormat.GSM_RTP,8000,8,1),18);
81. mgr.addReceiveStreamListener(this);
82.     String cname = mgr.generateCNAME();
83.     String username = null;
84.     try
85.     {    username = System.getProperty("user.name");
86.         }    catch (SecurityException e)
87.         {    username = "user";
88.         }
89.     SessionAddress localaddr = new SessionAddress();
90.     try
91.     {InetAddress destaddr = InetAddress.getByName(HostName);
92.     SessionAddress sessaddr = new SessionAddress(destaddr,port,destaddr,port +
93. 1);
94.     SourceDescription[] userdesclist= new SourceDescription[]
95. {new SourceDescription(SourceDescription.SOURCE_DESC_EMAIL,username+
96.     "@company.com",1,false),
97. new SourceDescription(SourceDescription.SOURCE_DESC_CNAME,cname,1,false),
98. new SourceDescription(SourceDescription.SOURCE_DESC_TOOL,"JMF RTP Player
99.     v2.0",1,false)
100. };
101. mgr.initSession(localaddr,userdesclist,0.5,0.25);
102. mgr.startSession(sessaddr,1,null);
103.     }    catch (Exception e)
104.     {    System.err.println(e.getMessage());
105.     }
106.     p.start();
107.     int j=0;
108.     System.out.println("AudioTransmitStream-started");
109.     }else
110.     System.out.println("AudioTransmitStream-zdZd");
111. } /* End of Run Method */
112. public void stop()
113. {stoped=true;    }
114. public boolean waitForState(int state)
115. {synchronized (waitObject)
116. {    try
117.     {    while (p.getState() < state && configured)
118.         {    waitObject.wait(1);    }

```

```

116.         waitObject.notifyAll();
117.     } catch (Exception e) {}
118. } return configured;
119. }
120. public synchronized void controllerUpdate(ControllerEvent ce)
121. {if (ce instanceof RealizeCompleteEvent)
122. {    configured = true;
123.     synchronized (waitObject)
124.     {    try
125.     {        waitObject.notifyAll();
126.     }        catch (Exception e) {}
127.     }

128. }else if (ce instanceof ConfigureCompleteEvent)
129. {    configured = true;
    synchronized (waitObject)
130.     {    try
131.     {        waitObject.notifyAll();
132.     }        catch (Exception e) {}
133.     }
134. }else if (ce instanceof PrefetchCompleteEvent)
135. {    prefetched = true;    }
136. else if (ce instanceof EndOfMediaEvent)
137. {    eom = true;    }
138. else if (ce instanceof ControllerErrorEvent)
139. {    failed = true;    }
140. else if (ce instanceof ControllerClosedEvent)
141. {    closed = true;    }
142. else if (ce instanceof ResourceUnavailableEvent)
143. {    configured=false;    }
144. else
145. {    return;
146. }
147. } /* End of synchronized void controllerUpdate(ControllerEvent ce) */
148. public void update(ReceiveStreamEvent event)
149. {Player player = null;
150.  ReceiverPlayerWindow playerWindow = null;
151.  System.out.println("in ReceiveStreamEvent1");
152.  SessionManager source = (SessionManager)event.getSource();
153.  if (event instanceof NewReceiveStreamEvent)
154.  {    String cname = "Java Media Player";
155.      ReceiveStream stream = null;
156.      try
157.      {    stream = ((NewReceiveStreamEvent)event).getReceiveStream();
158.          Participant participant = stream.getParticipant();
159.          if (participant != null) cname = participant.getCNAME();
160.          DataSource dsource = stream.getDataSource();
161.
162. /* create a player by passing datasource to the Media Manager */
163.
164.          player = Manager.createPlayer(dsource);
165.          System.out.println("created player " + player);
166.          try
167.          {    Thread.currentThread().sleep(500);
168.          }        catch(Exception e){}

```

```

169.         }        catch (Exception e)
170. {System.err.println("NewReceiveStreamEvent exception "+ e.getMessage());
171.         }
172.         if (player == null) return;
173.         playerlist.addElement(player);
174.         player.addControllerListener(this);
175./* send this player to player GUI */
176.     playerWindow = new ReceiverPlayerWindow( player, cname);
177.     }
178. } /* update(ReceiveStreamEvent event) */
179.
180. class ReceiverPlayerWindow extends PlayerWindow
181. {public ReceiverPlayerWindow( Player player, String title)
182.     {    super(player);
183.
184.         setTitle(title);
185.     }    public void Name(String title)
186.     {        setTitle(title);
187.     }
188.
188.     public static void main(String args[])
189.     {new AudioReceiveStreams("131.200.2.25"); }
190. }

```

Code Description

- ◆ Lines 1–14: Importing of packages as in the previous code.
- ◆ Line 16: Beginning of `AudioReceiveStreams` class.
- ◆ Line 17: The `Processor` interface defines a module for processing and controlling time-based media data.
- ◆ Line 18: Blank line.
- ◆ Line 19: Port number is defined as 49150 for Receiving Audio Streams.
- ◆ Lines 20–28: Necessary variables are declared.
- ◆ Lines 30–36: Creation of the constructor for `AudioReceiveStreams`.
- ◆ Lines 37–108: This is the `Run` method to Receive Audio Streams to the clients in the format (`AudioFormat.LINEAR, 44100, 16, 1`) as mentioned at line 39.
- ◆ Lines 42–47: Checking for Capturing Device. If the device is not found, all the streams will go to `CaptureDeviceInfo`; otherwise it returns "Device not found" and exits from the loop.
- ◆ Lines 48–54: `Try` block for creating a processor for the specified media. This `createProcessor` method returns a `String` value.
- ◆ Line 59: Set the output content descriptor to `RAW_RTP`.
- ◆ Line 60: To get the tracks from the processor by using the `getTrackControls` method and a `TrackControl` interface for each track in the media stream. This method can only be called after the `Processor` interface has been configured.
- ◆ Lines 62–70: Program for setting the track lengths.
- ◆ Line 77: Here an instance of `com.sun.media.rtp.RTPSessionMgr` is created.
- ◆ Lines 90–102: `Try` block for the `InetAddress` class that represents an Internet Protocol (IP) address. In this application, we used the method `getByName` to create a new `InetAddress` instance.

- ◆ Lines 93–97: Here the `SourceDescriptor` is for specifying the description of the source.
- ◆ Lines 109–110: For stopping the received streams.
- ◆ Lines 111–119: Synchronization method for `waitForState` with one parameter: the state of the processor.
- ◆ Lines 120–147: Synchronization method, `controllerupdate` for different types of events.
- ◆ Lines 148–178: `ReceiveStreamEvent` notifies a listener of all events that are received on a particular `ReceiveStream`. This allows the user to get details on all the `ReceiveStreams` as they transition through various states.
- ◆ Lines 180–187: Creates a class for `RTPPlayerWindow`. This `RTPPlayerWindow` extends the `PlayerWindow` class.
- ◆ Lines 188–190: Creating an instance of `AudioReceiveStreams`.

Listing 13-6: Student.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. import java.awt.*;
2. import java.awt.event.*;
3.
4. public class Student extends Frame implements ActionListener
5. {
6.     Button start, exit;
7.     AudioReceiveStreams as;
8.     public Student()
9.     {
10.         setLayout(new FlowLayout());
11.         start=new Button("Start");
12.         exit=new Button("Exit");
13.         start.addActionListener(this);
14.         exit.addActionListener(this);
15.         Panel p1= new Panel();
16.         Panel p2= new Panel();
17.         p1.add(start);
18.         p2.add(exit);
19.         add(p1);
20.         add(p2);
21.         setSize(100,75);
22.         setVisible(true);
23.         setTitle("Student");
24.     }
25.     public void actionPerformed(ActionEvent ae)
26.     {
27.         if (ae.getSource()==start)
28.         {
29.             as=new AudioReceiveStreams("131.200.2.25");
30.             start.setEnabled(false);
31.         }
32.         else if (ae.getSource()==exit)
33.         {
34.             start.setEnabled(true);
35.             as.stop();
36.             System.exit(0);
37.         }
38.     }
39.     public static void main(String[] args)
40.     {
41.         new Student();
42.     }
43. }

```

Code Description

- ◆ Line 1–2: Code for importing of packages.
- ◆ Line 4: The beginning of the main `Student` class.
- ◆ Lines 7–22: In this constructor, we have initialized buttons. To write event handlers on the buttons, we had the `addActionListener` method provided in the `ActionListener` interface. Through this `addActionListener` method, we can directly register the components into `ActionListener` Interface.
- ◆ Lines 9 and 10: Initializing the `Start`, `Stop`, and `Exit` variables through a `Button` class.
- ◆ Lines 11 and 12: Registering the buttons to the `Listener` interface through `addActionListener` method.
- ◆ Lines 13 and 14: Initializing the panels.
- ◆ Lines 15 and 16: Adding the buttons to the panels.
- ◆ Lines 17 and 18: Adding the panels to a frame.
- ◆ Line 19: Setting the size of the frame.
- ◆ Lines 23–32: This is an `actionPerformed` method to write the event handling for `Start`, `Stop`, and `Exit` Buttons.
- ◆ Lines 33 and 34: Creation of an instance of the `Student` class.
- ◆ Line 35: End of the `Student` class.

Application: Audio–Video Broadcasting

In this application, we develop the code for broadcasting of audio and video over a network using JMF. One node is on the network that is designated as server; the broadcast originates here. There can be a number of nodes, or clients, which can receive the broadcast. On the server, there will be two programs: `AudioVideoTransmit.java`, which is for the transmitting, and `Professor.java`, which provides the user interface. On each client are two programs: `AudioVideoReceive.java`, which is for reception of audio and video and `Student.bat`, which is a batch file. On all the machines, we need JDK 1.1 or later version and JMF 2.1.1 or later to be able to run. On the server, a video camera is required to transmit the video in addition to the sound card. Each client should have the sound card to receive the audio.

The code for `AudioVideoTransmit.java`, `Professor.java`, and `AudioVideoReceive.java` are provided in Listings 13-7, 13-8, and 13-9, respectively.

Listing 13-7: AudioVideoTransmit.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```
1. import java.awt.*;
2. import java.io.*;
3. import java.net.InetAddress;
4. import javax.media.*;
5. import javax.media.protocol.*;
6. import javax.media.format.*;
7. import javax.media.control.*;
8. import javax.media.rtp.*;
9. import javax.media.rtp.rtcp.*;
10. import com.sun.media.rtp.*;
11.
```

```

12. public class AudioVideoTransmit implements ControllerListener
13. {
14.     MediaLocator medialocator;
15.     String broadcastAddress = "255.255.255.255";
16.     int portNumber;
17.     boolean failed = false;
18.     Processor p = null;

19.     RTPManager rm[];
20.     DataSource datasource = null;
21.     public AudioVideoTransmit(int port)
22.     {
23.         medialocator= new MediaLocator("vfw://0&jawasound://44100");
24.         portNumber = port;
25.     }

26.     public synchronized String startTransmitter()
27.     {
28.         String status;
29.         status = startProcessor();
30.         if (status != null)
31.         {
32.             return status;
33.         }
34.         status = createRTPSession();
35.         if (status != null)
36.         {
37.             p.close();
38.             p = null;
39.             return status;
40.         }
41.         p.start();
42.         return null;
43.     }

44.     public void stopTransmitter()
45.     {
46.         synchronized (this)
47.         {
48.             if (p != null)
49.             {
50.                 p.stop();
51.                 p.close();
52.                 p = null;
53.                 for (int i = 0; i < rm.length; i++)
54.                 {
55.                     rm[i].removeTargets("Session ended.");
56.                     rm[i].dispose();
57.                 }
58.             }
59.         }

60.     }

61.     String startProcessor()
62.     {
63.         DataSource datasourceProcess;

```



```

67.         ContentDescriptor cdesc;
68.         TrackControl [] trackcontrol;
69.         boolean status, trackForSupport = false;
70.         Format supportedFormats[];
71.         Format selectedFormat;
72.     try {
73.         datasourceProcess = Manager.createDataSource(medialocator);
74.         } catch (Exception e)
75.         {return "Error in creating DataSource";}
76.     try {
77.         p = Manager.createProcessor(datasourceProcess);
78.     } catch (NoProcessorException pe)
79.     {return "Error in creating processor";}

80.     catch (IOException ie) {return "Error in creating processor";}
81.     status = wait(p, Processor.Configured);
82.     if (status == false)
83.         return "Error in configuring processor";
84.     trackcontrol = p.getTrackControls();
85.     if (trackcontrol == null || trackcontrol.length < 1)
86.         return "No tracks in processor";
87.     cdesc = new ContentDescriptor(ContentDescriptor.RAW_RTP);
88.     p.setContentDescriptor(cdesc);
89.     for (int i = 0; i < trackcontrol.length; i++)
90.     {
91.         Format format = trackcontrol[i].getFormat();
92.         if (trackcontrol[i].isEnabled())
93.         {
94.             supportedFormats = trackcontrol[i].getSupportedFormats();
95.             if (supportedFormats.length > 0)
96.             {
97.                 selectedFormat = supportedFormats[0];
98.                 trackcontrol[i].setFormat(selectedFormat);
99.                 System.out.println("Track " + i + " is transmitting ");
100.                 trackForSupport = true;
101.             } else
102.                 trackcontrol[i].setEnabled(false);
103.             } else
104.                 trackcontrol[i].setEnabled(false);
105.         }
106.     if (!trackForSupport)
107.         return "No Supported tracks for valid RTP format";
108.     status = wait(p, Controller.Realized);
109.     if (status == false)
110.         return "Not realized the processor";
111.     datasource = p.getDataOutput();
112.     return null;
113.     }
114. String createRTPSession()
115.     {
116.         PushBufferDataSource pbd = (PushBufferDataSource)datasource;
117.         PushBufferStream pbs[] = pbd.getStreams();
118.         rm = new RTPManager[pbs.length];
119.         SessionAddress localAddress, destAddress;
120.         InetAddress ipAddress;
121.         SendStream sendStream;

```

```

122. int port;
123. SourceDescription srcDesList[];
124. for (int i = 0; i < pbs.length; i++)
125. {
126.     try {
127.         rm[i] = RTPManager.newInstance();
128.         port = portNumber + 2*i;
129.         ipAddress = InetAddress.getByName(broadcastAddress);
130.         localAddress = new SessionAddress( InetAddress.getLocalHost(), port);
131.         destAddress = new SessionAddress( ipAddress, port);
132.         rm[i].initialize( localAddress);
133.         rm[i].addTarget( destAddress);

134. System.out.println("RTP session: "+broadcastAddress + " " + port);
135.         sendStream = rm[i].createSendStream(datasource, i);
136.         sendStream.start();
137.         } catch (Exception e) { return e.getMessage();}
138.     }
139.     return null;
140. }

141.
142. private synchronized boolean wait(Processor p, int state)
143.     {
144.         p.addControllerListener(this);
145.         failed = false;
146.         if (state == Processor.Configured)
147.         {
148.             p.configure();
149.         }
150.         else if (state == Processor.Realized)
151.             {
152.                 p.realize();
153.             }
154.         while (p.getState() < state && !failed)
155.             {
156.                 synchronized (this)
157.                 {
158.                     try {
159.                         this.wait();
160.                     } catch (InterruptedException ie) { return false;}
161.                 }
162.             }
163.         if (failed)
164.             {
165.                 return false;
166.             }
167.         else
168.             {
169.                 return true;
170.             }
171.     }
172. public synchronized void controllerUpdate(ControllerEvent ce)
173.     {
174.         if (ce instanceof ControllerClosedEvent)
175.             {
176.                 failed = true;
177.             }
178.     }

```

```

177.         if (ce instanceof ControllerEvent)
178.         {
179.             this.notifyAll();
180.         }
181.     }
182. }

```

Code Description

- ◆ Lines 1–10: Importing of the packages.
- ◆ Line 12: Class for `AudioVideoTransmit` starts here.
- ◆ Line 14: The `MediaLocator` class provides the way to identify the location of a media stream. It can be file or capture device source.
- ◆ Line 15: Variable declaration for broadcast address with name "broadcastAddress".
- ◆ Line 16: Variable declaration for port to transmit data.
- ◆ Line 17: Variable declaration for checking processor failure.
- ◆ Line 18: The `Processor` interface defines a module for processing and controlling time-based media data. `Processor` extends the `Player` interface.
- ◆ Line 19: Variable declaration for `RTPManager`. `RTPManager` API creates sessions for each media track of the processor.
- ◆ Line 20: The `DataSource` is an abstraction for media protocol handlers. `DataSource` manages the life cycle of the media source by providing a simple connection protocol. This `DataSource` is available in the `javax.media.protocol` package.
- ◆ Lines 21–25: This code is for the `AudioVideoTransmit` constructor with parameter port, meaning, the port number on which the track will transmit. If you want to transmit a prerecorded file, replace line 23 with the following line: `medialocator= new MediaLocator("file://c:/media/samp.mov");`
- ◆ Lines 27–44: Method `startTransmitter()` for starting the transmission.
- ◆ Line 30: Start a processor for the specified media locator by calling method `startProcessor()`. If the processor is created, it returns `null`. Lines 31–34 are for checking whether the processor is created.
- ◆ Line 35: Creating an RTP session by calling the method `reateRTPSession()`. If the RTP session is created successfully, it returns `null`.
- ◆ Line 42: Starts the processor if both the above methods are successful and returns `null`.
- ◆ Lines 46–62: Method `stopTransmitter()` to stop the transmission, that is, closing the processor and disposing of the RTP manager.
- ◆ Lines 64–113: Method `startProcessor()` for creating a processor for the specified media.
- ◆ Lines 66–71: Variable declarations for `DataSource`, `TrackControl`, `ContentDescriptor`, `status`, `trackForSupport`, `SupportedFormats`, and `selectedFormat`.
- ◆ Lines 72–75: Try block for creating the `DataSource` using `createDataSource` method for the specified `medialocator`.
- ◆ Lines 76–80: Try block to create a processor to handle the input `medialocator`.
- ◆ Line 81: Calls the method `wait()`, waiting to configure, returns `true` if successful
- ◆ Lines 82 and 83: To check whether the processor is configured. If the processor is not configured, it returns "Error in configuring processor".

- ◆ Lines 84–86: To get the tracks from the processor by using the `getTrackControls` method and a `TrackControl` interface for each track in the media stream. This method can only be called after the `Processor` Interface has been configured and after checking for at least one track.
- ◆ Line 87: Create the content descriptor to `RAW_RTP`.
- ◆ Line 88: Sets the content descriptor to `Processor`.
- ◆ Lines 89–105: Checking for the tracks to supported formats.
- ◆ Line 107: If no track is detected, it returns "No Supported tracks for valid RTP format".
- ◆ Line 108: Calls the `wait()` method to realize the processor
- ◆ Lines 109 and 110: Checks whether the status of the process is `false`.
- ◆ Line 111: Gets the output data source of the processor.
- ◆ Lines 114–140: Method `createRTPSession()`, which creates sessions for each media track of the processor. Note that at line 116 the `PushBufferDataSource` class abstracts a data source that manages data in the form of push streams. The streams from this data source contain `Buffer` objects, meaning `PushBufferStreams`.
- ◆ Lines 142–170: Synchronization method for wait with two parameters, `Processor` and `State` of the processor. This method is used for configuring and realizing the `Processor`.
- ◆ Lines 171–182: Method `controllerUpdate()` for `ControllerListener` to catch the controller events.

Listing 13-8: Professor.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved.

```

1. import java.awt.*;
2. import java.awt.event.*;
3. public class Professor extends Frame implements ActionListener
4. {
5.     Button start,exit;
6.     AudioVideoTransmit avt;
7.     Panel panel;
8.     public Professor()
9.     {
10.         setLayout(null);
11.         panel = new Panel();
12.         panel.setLayout(null);
13.         panel.setBounds(10,20,170,50);
14.         start=new Button("Start");
15.         start.setBounds(10,10,60,20);
16.         exit=new Button("Exit");
17.         exit.setBounds(80,10,60,20);
18.         exit.setEnabled(false);
19.         start.addActionListener(this);
20.         exit.addActionListener(this);
21.         panel.add(start);
22.         panel.add(exit);
23.         add(panel);
24.         setBounds(350,200,170,70);
25.         setTitle("Professor");
26.         setVisible(true);
27.     }

```

```

28.         public void actionPerformed(ActionEvent ae)
29.         {
30.             if (ae.getSource()==start)
31.             {
32.                 avt=new AudioVideoTransmit(49250);
33.                 String result=avt.startTransmitter();
34.                 if (result != null)
35.                 {
36.                     System.out.println("Error : " + result);
37.                     System.exit(0);
38.                 }
39.                 exit.setEnabled(true);
40.                 start.setEnabled(false);
41.             }
42.             else if (ae.getSource()==exit)
43.             {
44.                 avt.stopTransmitter();
45.                 System.exit(0);
46.             }
47.         }
48.         public static void main(String[] args)
49.         {
50.             new Professor();
51.         }
52. }

```

Code Description

- ◆ Lines 1 and 2: Importing of packages.
- ◆ Lines 3–52: The main Professor class starts.
- ◆ Lines 5–7: Variable declaration of Buttons, Panel, and AudioVideoTransmit classes.
- ◆ Lines 8–27: In this constructor Buttons and Panel are initialized and added to the Frame, and the boundaries are set; event handlers are written on the Buttons, using addActionListener method provided in ActionListener Interface.
- ◆ Lines 28–47: This is an actionPerformed method to write the event handling for Start and Exit buttons.
- ◆ Line 48–52: Main method to call the Professor class.

Listing 13-9: AudioVideoReceive.java

© 2001 Dreamtech Software India Inc.
All Rights Reserved

```

1. import java.io.*;
2. import java.awt.*;
3. import java.net.*;
4. import java.awt.event.*;
5. import java.util.Vector;
6. import javax.media.*;
7. import javax.media.rtp.*;
8. import javax.media.rtp.event.*;
9. import javax.media.rtp.rtcp.*;
10. import javax.media.protocol.*;
11. import javax.media.format.*;

```

```

12. import javax.media.control.BufferControl;
13. public class AudioVideoReceive implements ReceiveStreamListener,
    SessionListener, ControllerListener
14. {
15.     String sessions[] = null;
16.     RTPManager managers[] = null;
17.     Vector receiverWindows = null;
18.     boolean dataObtained = false;
19.     Object waitObject = new Object();
20.     public AudioVideoReceive(String sessionsAddress[])
21.     {
22.         this.sessions = sessionsAddress;
23.     }
24.     protected boolean initialize()
25.     {
26.         long currenttime, waitingtime;
27.         try {
28.             InetAddress ipAddress;
29.             SessionAddress localAddress = new SessionAddress();
30.             SessionAddress destAddress;
31.             int lengthOfSessions = 2;
32.             int port[] = {49250, 49252};
33.             int ttl = 1;
34.             managers = new RTPManager[lengthOfSessions];
35.             receiverWindows = new Vector();
36.             for (int i = 0; i < lengthOfSessions; i++)
37.             {
38.                 System.out.println("RTP session for : " + sessions[0] + " port: " + port[i]
39.                 + " ttl: " + ttl);
40.                 managers[i] = (RTPManager) RTPManager.newInstance();
41.                 managers[i].addSessionListener(this);
42.                 managers[i].addReceiveStreamListener(this);
43.                 ipAddress = InetAddress.getByName(sessions[0]);
44.                 if( ipAddress.isMulticastAddress())
45.                 {
46.                     localAddress= new SessionAddress( ipAddress, port[i], ttl);
47.                     destAddress = new SessionAddress( ipAddress, port[i], ttl);
48.                 }
49.                 else
50.                 {
51.                     localAddress= new SessionAddress( InetAddress.getLocalHost(), port[i]);
52.                     destAddress = new SessionAddress( ipAddress, port[i]);
53.                 }
54.                 managers[i].initialize( localAddress);
55.                 BufferControl bc = (BufferControl)managers[i].getControl("BufferControl");
56.                 if (bc != null)
57.                 {
58.                     bc.setBufferLength(350);
59.                     managers[i].addTarget(destAddress);
60.                 }
61.                 } catch (Exception e){
62.                     System.out.println("Not creating the RTP Session: " +
63.                     e.getMessage());
64.                     return false;
65.                 }
66.             }
67.             currenttime = System.currentTimeMillis();

```

```

64. waitingtime = 30000;
65. try{
66.     synchronized (waitObject)
67.     {
68.         while (!dataObtained && System.currentTimeMillis() - currenttime <
waitingtime)
69.         {
70.             if (!dataObtained)
71.                 System.out.println("Waiting for the RTP data");
72.             waitObject.wait(1000);
73.         }
74.     }
75. } catch (Exception e) {}
76. if (!dataObtained)
77. {
78.     System.out.println("No RTP data was received.");
79.     close();
80.     return false;
81. }
82.     return true;
83. }
84. public boolean isExecute()
85. {
86. return receiverWindows.size() == 0;
87. }
88. protected void close()
89. {
90. for (int i = 0; i < receiverWindows.size(); i++)
91. {
92.     try {
93.         ((Receiver)receiverWindows.elementAt(i)).close();
94.     } catch (Exception e) {}
95. }
96. receiverWindows.removeAllElements();
97. for (int i = 0; i < managers.length; i++)
98. {
99.     if (managers[i] != null)
100.    {
101.        managers[i].removeTargets("Closing this session.");
102.        managers[i].dispose();
103.        managers[i] = null;
104.    }
105. }
106. }
107. Receiver findPorS(Player p)
108. {
109. for (int i = 0; i < receiverWindows.size(); i++)
110. {
111.         Receiver rf = (Receiver)receiverWindows.elementAt(i);
112.         if (rf.player == p)
113.             return rf;
114.     }
115. return null;
116. }
117. Receiver findPorS(ReceiveStream rstrm)

```

```

118.     {
119.     for (int i = 0; i < receiverWindows.size(); i++)
120.     {
121.         Receiver rf = (Receiver)receiverWindows.elementAt(i);
122.         if (rf.stream == rstrm)
123.             return rf;
124.     }
125.     return null;
126.     }
127.     public synchronized void update(SessionEvent sevt)
128.     {
129.         if (sevt instanceof NewParticipantEvent)
130.         {
131.             Participant pse = ((NewParticipantEvent)sevt).getParticipant();
132.             System.out.println("New participant joined: " + pse.getCNAME());
133.         }
134.     }
135.     public synchronized void update( ReceiveStreamEvent revt)
136.     {
137.         Participant prse = revt.getParticipant();
138.         ReceiveStream stream = revt.getReceiveStream();
139.         if (revt instanceof RemotePayloadChangeEvent)
140.         {
141.             System.exit(0);
142.         }
143.         else if (revt instanceof NewReceiveStreamEvent)
144.         {
145.             try {
146.                 stream = ((NewReceiveStreamEvent)revt).getReceiveStream();
147.                 DataSource ds = stream.getDataSource();
148.                 RTPControl rtpctl = (RTPControl)ds.getControl("RTPControl");
149.                 Player p = Manager.createPlayer(ds);
150.                 if (p == null)
151.                     return;
152.                 p.addControllerListener(this);
153.                 p.realize();
154.                 Receiver rf = new Receiver(p, stream);
155.                 receiverWindows.addElement(rf);
156.                 synchronized (waitObject)
157.                 {
158.                     dataObtained = true;
159.                     waitObject.notifyAll();
160.                 }
161.             } catch (Exception e)
162.             {
163.                 System.err.println("Exception " + e.getMessage());
164.                 return;
165.             }
166.         }
167.     }
168.     else if (revt instanceof StreamMappedEvent)
169.     {
170.         if (stream != null && stream.getDataSource() != null)
171.         {
172.             DataSource ds = stream.getDataSource();

```



```

173.         RTPControl rtpctl = (RTPControl)ds.getControl("RTPControl");
174.         if (rtpctl != null)
175.         {
176.             System.out.println(" " + rtpctl.getFormat());
177.         }
178.     }
179. }
180. else if (revt instanceof ByeEvent)
181. {
182.     Receiver rf = findPorS(stream);
183.     if (rf != null)
184.     {
185.         rf.close();
186.         receiverWindows.removeElement(rf);
187.     }
188. }
189. }
190. public synchronized void controllerUpdate(ControllerEvent ce)
191. {
192.     Player p = (Player)ce.getSourceController();
193.     if (p == null)
194.         return;
195.     if (ce instanceof RealizeCompleteEvent)
196.     {
197.         Receiver rf = findPorS(p);
198.         if (rf == null)
199.         {
200.             System.exit(0);
201.         }
202.         rf.initialize();
203.         rf.setVisible(true);
204.         p.start();
205.     }
206.     if (ce instanceof ControllerErrorEvent)
207.     {
208.         p.removeControllerListener(this);
209.         Receiver rf = findPorS(p);
210.         if (rf != null)
211.         {
212.             rf.close();
213.             receiverWindows.removeElement(rf);
214.         }
215.     }
216. }

217.     class Receiver extends Frame
218.     {
219.         Player player;
220.         ReceiveStream stream;
221.         Panel viewpanel;
222.         Component visualcomponent, controlcomponent;
223.         Receiver(Player p, ReceiveStream strm)
224.         {
225.             player = p;
226.             stream = strm;
227.         }

```

```

228. public void initialize()
229. {
230.     viewpanel = new Panel(new BorderLayout());
231.     if ((visualcomponent = player.getVisualComponent()) != null)
232.         viewpanel.add("Center", visualcomponent);
233.     if ((controlcomponent = player.getControlPanelComponent()) != null)
234.         viewpanel.add("South", controlcomponent);
235.     add(viewpanel);
236. }
237. public void close()
238. {
239.     player.close();
240.     setVisible(false);
241.     dispose();
242. }
243. public void addNotify()
244. {
245.     super.addNotify();
246.     pack();
247. }
248. }
249. public static void main(String args[])
250. {
251.     if (args.length == 0)
252.     {
253.         System.out.println("Usage : java AudioVideoReceive ipAdress");
254.     }
255.     else
256.     {
257.         AudioVideoReceive avr = new AudioVideoReceive(args);
258.         if (!avr.initialize())
259.         {
260.             System.out.println("Failed to initialize.");
261.             System.exit(0);
262.         }
263.         try {
264.             while (!avr.isExecute())
265.                 Thread.sleep(1000);
266.         } catch (Exception e) {System.out.println(e);}
267.     }
268. }
269. }

```

Code Description

- ◆ Lines 1–12: Importing of necessary packages.
- ◆ Lines 13 and 14: Starting of the `AudioVideoReceive` class to receive RTP Transmission.
- ◆ Lines 15–19: Variables Declaration for `RTPManager`. This `RTPManager` creates sessions for each media track of the processor.
- ◆ Lines 20–23: The constructor of the `AudioVideoReceive`.
- ◆ Lines 24–83: Method `initialize()` is for receiving audio video streams.
- ◆ Lines 28–30: Variable declaration of `InetAddress` and `SessionAddress`.
- ◆ Line 34: Creating the `RTPManagers` for sessions.

- ◆ Lines 36–58: Creating the new instance of `RTPManagers` and adding the `SessionListener` and `ReceiveStreamListener` and initializing the `RTPManagers`.
- ◆ Lines 54–56: Trying with some other buffer size for better smoothness.
- ◆ Line 57: Pass `destAddress` to `RTPManager` using the `addTarget` method.
- ◆ Lines 63–81: Getting the current time and setting the wait time for 30 seconds. This section waits for 30 seconds to get the RTP data from transmitter. If data is received, it returns `true`; otherwise `false`.
- ◆ Lines 84–87: Method `isExecute()` is used for checking whether the data is received or not, after initialization of audio-video receiver.
- ◆ Lines 88–106: This is a method to close the players and the session managers.
- ◆ Lines 107–126: Methods `findPorS()` for finding `Player` or `ReceiveStream`.
- ◆ Lines 127–134: Method `update(SessionEvent sevt)` for `SessionListener`.
- ◆ Lines 135–189: Method `update(ReceiveStreamEvent revt)` for `ReceiveStreamListener`.
- ◆ Lines 190–216: Method `controllerUpdate(ControllerEvent ce)` for `ControllerListener`.
- ◆ Lines 217–248: Creating a GUI for receiver for a player.
- ◆ Lines 249–269: Main method for creating an instance of `AudioVideoReceive` class and initializing it.

To run the `AudioVideoTransmit.java` and `Professor.java` programs at the server:

1. Enter the source code and compile the files.
2. `Professor.class` file is created. Run the `Professor.class` file on the server system by giving the command `java Professor`. A screen display contains the Start and Exit buttons. After you click the Start Button, the `AudioVideoTransmit` class captures the video of the Professor from the video camera and sound from the microphone. It then transmits audio and video to whomever is logged on to the server.

To run the `AudioVideoReceive.java` and `Student.java` programs on the client:

1. Enter the code and compile the JAVA file.
2. Run the `Student.bat` file (see the following section) on the client's system. You can see the image of the Professor (meaning, the Transmitter image) and hear audio through the headphone.

You can create a file `Student.bat` file with the following line:

```
java AudioVideoReceive 255.255.255.255
```

If you have any problem running the `Student.bat` file, change the IP addresses in `Student.bat` file to the Professor's system IP address. For example:

```
java AudioVideoReceive 131.200.2.25
```

Here `131.200.2.25` is the IP address of Professor's system.

Summary

This chapter presented the implementation of audio and video applications based on H.323 standards using JMF. The H.323 standards specify the protocols and coding techniques used for voice and video communication over IP networks. However, these standards do not guarantee a desired quality of service. So, special protocols — RTP and RTCP — are defined by which we can achieve real-time transmission of voice and video over IP networks. In addition, low bit rate coding standards are specified for both

voice and video. JMF provides the necessary class libraries for implementation of H.323 standards. Voice and video communication over IP networks will be widely available in the future because of the savings in cost it provides. Implementation of H.323 over mobile devices is much more attractive because we can have very low cost voice and video communication using mobile devices. The examples discussed in this chapter can be effectively used for developing applications, such as m-learning (mobile learning) — you can listen to a professor while relaxing in a garden rather than sitting at your desktop in a brick-and-mortar classroom. For 3G services to take off on a grand scale in the near future, the key lies with the content development through 3G programming for devices that run a full-fledged Java Virtual Machine and H.323 protocol stack.

Chapter 14

The Future of Wireless Networks

The 2G wireless networks of yesteryear are slowly being upgraded to 2.5G networks, which support higher data rates. In the next few years, these systems will be upgraded to 3G networks, which support much higher data rates to provide full-fledged audio- and video-streaming applications over mobile devices. Still, the end user's demand for much higher bandwidths and value-added services are forcing the network operators and infrastructure providers to develop new technologies to support broadband services. In this chapter, we will outline the new developments taking place in the wireless arena. We will discuss the emergence of *convergence* — convergence of networks and convergence of services, which will facilitate new services, such as instant messaging and unified messaging, which are also discussed. We will highlight the developments taking place in the field of mobile devices, content development, and protocols.

Convergence Technologies

Today, we use different networks for accessing different services. We use the Public Switched Telephone Network (PSTN) for making calls from a land-line telephone and for sending fax messages; we use the Public Land Mobile Network (PLMN) to make calls from a mobile telephone; we use the desktop to access the Internet through an Internet Service Provider (ISP); we use the paging network to page a person on the move; we use the cable TV network to receive TV programs; and we use the radio to receive audio from audio broadcasting stations.

As shown in Figure 14-1, a number of networks provide different services, and to access these networks, users have a number of terminals. We also keep a number of mailboxes to receive messages — the e-mail boxes at our ISP, or mailboxes at Web-based e-mail service providers (such as Hotmail or Yahoo!), voice mailboxes located at servers of fixed telephone service providers, and mobile telephone service providers. Certainly, all these services together provide us the power to communicate and to be in touch with our office, our home, and our friends. The main drawbacks of this telecommunication architecture are

- ◆ The user has to maintain a number of terminals to access different services through different networks, has to keep track of a number of mailboxes, and also has to keep track of the multiple bills from different service providers.
- ◆ The operators and service providers have to upgrade their networks continually to provide higher bandwidths as the subscriber capacity and the demand for high bandwidth services grow.
- ◆ To provide value-added services to the users (for instance, a single mailbox for all types of mail — e-mails or voice mails), each operator has to come to an understanding with other operators, which calls for the resolution of many administrative issues.

To make life simpler for the end user, we are now witnessing a revolution in telecommunications in the form of convergence. The main objective of convergence is to provide the end user with a simple and efficient means of accessing the services, so that the user is not concerned with the underlying network technologies and protocols but can obtain the desired service using a terminal of his choice.

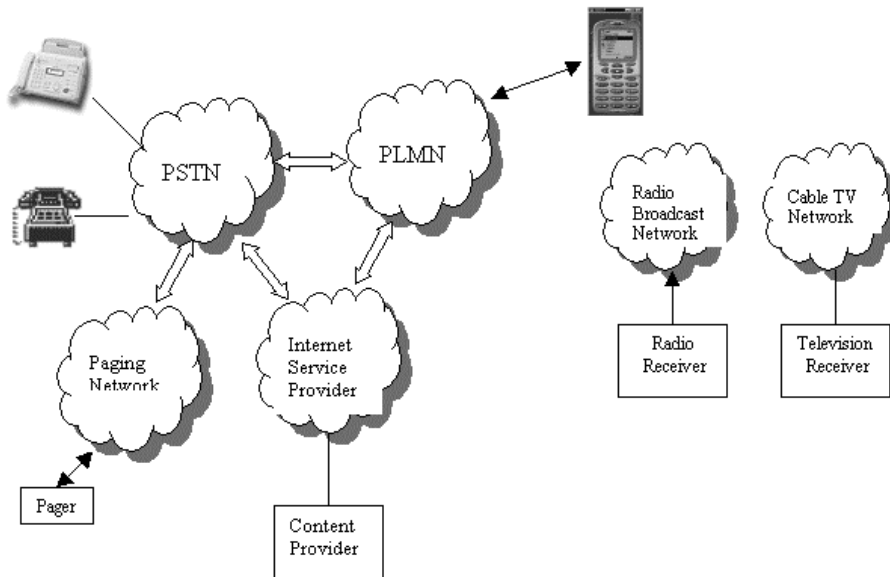


Figure 14-1: Architecture of telecommunications networks

Convergence of Networks

The first step in convergence is to redefine the telecommunications architecture, as shown in Figure 14-2. Here we have a backbone network, which is a very high-speed optical-fiber network. The backbone network can support very high data rates for data, voice, and video applications. The backbone network is connected to various *access networks* — these networks provide access to the end users. The access network can be a fixed telephone network, cable TV network, mobile network, and so forth. The end user gets connected to the backbone network through the access network to obtain various data, voice, and video services using a terminal of his choice — it can be a desktop PC, a laptop, a mobile device, a WebTV, and so on. The end user can also have a Personal Area Network (PAN), which is the ad-hoc network of the devices of the user. The content providers and the applications providers connect the servers to the backbone network.

The architecture shown in Figure 14-2 clearly demonstrates how Bluetooth and 3G technologies can work together to provide services to the end users. For instance, one can access the Internet through the desktop PC, download files, transfer them to the laptop, simultaneously download MP3 music from a Web site onto the mobile device, and listen to the music through the headset — all that without the need for wires.

Wireless last mile

The traditional wired telecommunication systems provided a great challenge only in the “last mile.” The link from the telephone exchange (or switch or end office) to the home/office is the costliest element in the whole network. This link, known as the subscriber loop or local loop, takes away nearly 50 percent of the total cost of the providing telephone. The installation and maintenance costs are very high for wired local loops. In remote/rural areas, this cost is much higher, because the population is geographically dispersed. Laying cables in mountain/hilly regions is a Herculean task. The wireless local loop is now the best solution because we can avoid wires. This results in a drastic cost reduction. However, in the earlier days, the wireless local loop supported only low data rates. With the advent of 2.5G and 3G systems, very high data rates are supported. The last mile problem is solved through these wireless networks.

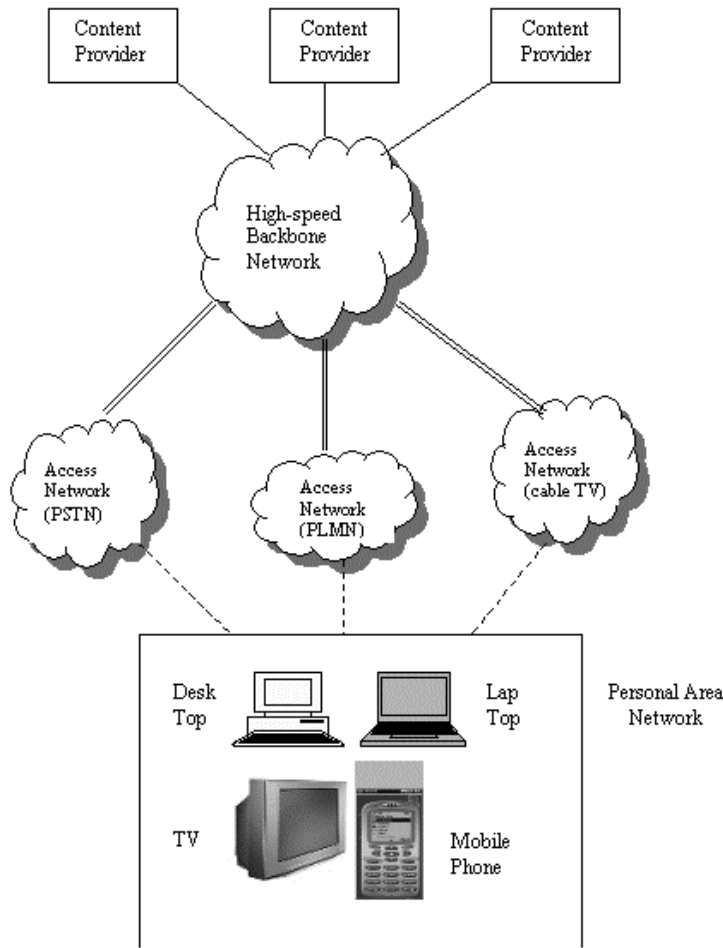


Figure 14-2: Convergence of networks

Wireless last inch

Technologies such as Bluetooth network connect the devices in a small area, such as home or office cabin. This network in turn can get connected to the access network. This “wireless last inch” attains significance mainly because it provides mobility to the users — the user need not be glued to a desktop or a land-line telephone. In the future, the tariffs on different networks will be different. Bluetooth-enabled devices will have the intelligence to decide which access network to connect for the desired services. The Convergence paradigm beautifully arranges the marriage between 3G technology and Bluetooth technology.

Convergence of services

The architecture shown in Figure 14-3 provides high data rate services to the end users. The service spectrum shown in Figure 14-3 depicts the various service categories: data, one-way audio, interactive audio, one-way video, and two-way video services. This figure also represents the data rates requirements; however, note that the data rates are not to scale.

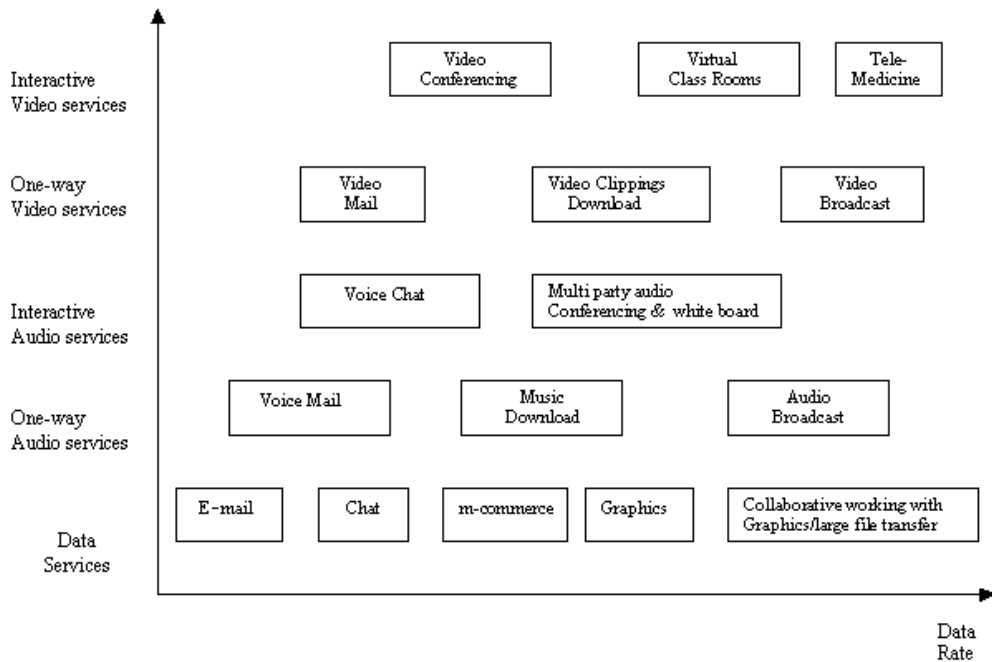


Figure 14-3: The service spectrum

Data services include e-mail, text chat, mobile commerce, graphics and collaborative working. In collaborative working, groups of persons working on different corners of the earth can form a virtual group to carry out, for example, software development. This calls for large file transfers, high-resolution graphics, and so forth.

The one-way audio applications include voice mail, downloading of music, and audio broadcast. Audio broadcasting over the Internet is an important paradigm shift — the broadcasting networks will “merge” with the Internet, and the Internet itself can be used for listening to audio programs. During the next few years, “wireless Web radios” will be a consumer item that can be used to tune the desktop or a mobile device to a Web site to download audio programs.

The interactive audio applications include voice chat between two persons and multi-party audio conferencing with white board facility to share graphics. At present, over the Internet, the interactive audio applications use low bit rate coding of voice. The quality of speech, therefore, is not very good. With the high-speed wireless networks of 2.5G and 3G, voice quality can be improved substantially, and high fidelity audio can be used for voice applications.

One-way video services include video mail, wherein video clippings can be attached to the e-mail messages, downloading of video clippings, and video broadcasting. Again, this is another important aspect of convergence — the convergence of cable TV and Internet. TV programs can be distributed over the Internet backbone to the cable TV access network, which can then be accessed by the end user from a desktop PC, an Internet-enabled TV set, or from a mobile device.

Interactive video services include two-way video conferencing or multi-party video conferencing. This service has potential applications in many areas, the two most important being education and medicine. In regard to education, lessons can be taught through virtual classrooms so that one can sit in the comfort of his/her home and participate in lectures and interact with the teacher. The teacher and the students can share the data, graphics, and so forth, which, of course, require high bandwidths. Another area where this service is of great benefit is in telemedicine, or “medicine from a distance.” One can transmit the patient data (such as ECG, EEG, and x-ray) over the Internet to specialists located in distant places to obtain

medical help. This is of great significance, particularly for providing emergency services from ambulances: paramedic staff can send diagnostic information to the hospital within minutes after, say, an accident. Providing telemedicine service through the wireless networks is also of great importance in rural and remote areas where there are no medical facilities. Mobile hospitals can carry the diagnostic equipment, and the paramedic staff can have a doctor in the nearest city analyze the information. Certainly, such services will reduce the “digital divide” between the different sections of the society. Note that the transmission of high-resolution diagnostic data such as x-rays requires high bandwidths that can be supported only by 3G networks.

From an end-user point of view, the main advantage of the convergence of networks is that the user can use a terminal of his choice to access the service. Invariably, the choice would be a mobile device, as the mobile device is considered “the most personal gadget” one has. Of course, one still has to rely on the desktop PC for obtaining large-screen, high-resolution graphics or to compose a large document and send it over the network.

You can visualize the advantages of accessing a converged network from any device. If you are at home, you can still use your mobile phone to make a call through the PSTN. The Bluetooth-enabled mobile phone will automatically detect the presence of the PSTN connection availability and establish the call over the PSTN because the call from the PSTN would be a cheaper choice. Or, you can use your mobile phone to access any of the mailboxes.

A new set of applications is now being made available based on three emerging technologies: speech recognition, text to speech conversion, and computer-telephony integration.

Emerging Technologies

Although we use text as the medium of communication, speech is the most convenient and effective means of communication among human beings. If we are able to communicate with computers in speech form, communication will be really effective. Speech recognition, text-to-speech conversion, and computer telephony integration technologies help in achieving this objective.

Speech Recognition

Automatic speech recognition technology has now matured to a stage where it is commercially viable. Using this technology, computers or mobile devices can be made to understand the words we speak. Nowadays, some mobile phones support a “voice dialing” feature. For example, when a user speaks the word “home” into the mobile phone, it will recognize the word and connect the user to his or her home number by looking up the corresponding phone number in the database within the mobile device. Of course, one must “train” the mobile phone to recognize his or her voice.

Another interesting possibility is to browse the Web through voice commands. For example, you can say “hungryminds.com” and the corresponding Web site will be displayed. Or you can say “Search” and the search engine will appear, or say “3G programming” and all the books on 3G programming will be displayed. Later in the chapter, we will see how these functions can be achieved through a new mark-up language called VoiceXML.

Text-to-Speech Conversion

Just the way we want computers and mobile devices to recognize our speech, computers can be made to convert text into speech. Conversion of text into speech is not straightforward because of the pronunciation idiosyncrasies of languages (for example, the letter “u” is pronounced differently in the words “put” and “but”). Special software for text-to-speech conversion is required. It takes the text as input and produces the output. Text-to-speech conversion enables us to get Web content through speech. WAP, for example, facilitates obtaining focused information through text messages (such as stock quotes). Using the text-to-speech conversion technology, we can obtain the stock quotes in speech form through a server. If we combine text-to-speech and speech recognition, the mobile phone can serve as a

very user-friendly mechanism to obtain information from the Web. For instance, a user can dial a stock trading portal through voice dialing and say, “What is the stock price of Microsoft?” The server will recognize the speech, retrieve the information from the database, convert it into speech format, and deliver the stock price in speech format: “The stock quote of ABC Inc. is 100 dollars as at eleven hours.”

Computer Telephony Integration (CTI)

A lot of information is presently available on computers in the form of text files, databases, and such. Because the penetration of PCs is low as compared to penetration of telephones (mobile and fixed), if we can provide the information through speech, it is ideal. CTI technology facilitates this by a combination of speech recognition, text-to-speech conversion, and Interactive Voice Response (IVR) systems. In IVR systems, the computer will prompt the user to give the input through the keypad on the telephone. For instance, if you want to find out the status of your bank balance in your bank account, you can dial the bank IVR system; the IVR system will ask you to dial the account number, which you can input through the keypad. The IVR system retrieves the information from the database, does text-to-speech conversion, and informs you “Your bank balance is five hundred and sixty dollars.”

During the next few years, these technologies will be integrated into the services being provided on mobile devices to make the interaction between humans and the Internet servers much more effective. The new services emerging can be categorized into instant messaging, unified messaging, and precise location-based services.

Instant Messaging

Today, we spend lot of time finding out whether we have received new e-mails by connecting to the Internet and logging into our mailboxes. With packet-based wireless networks, mobile devices can be “always connected” to the network, and instead of pulling the information from the servers, the server can push the information to the user. For example, whenever a new e-mail message arrives in your mailbox, instant messaging informs you immediately. Based on the urgency, you can retrieve the message. The instant message can be sent to a device of your choice — your obvious choice is your mobile phone. Instant messaging can also be used for informing you when your friends or relatives have logged on to a Web server when you are also logged on. This facilitates chat or sharing other information. With the convergence of networks, the instant message can be delivered to any of the devices — a mobile phone, a pager, or a desktop.

Another value addition of instant messaging is that you can combine it with location-based services. Whenever a person in your database or address book is near you (or your mobile phone), you will receive a notification to that effect.

Unified Messaging

Technology has made it possible to communicate with anyone, anywhere, anytime using different media: voice, data, fax, and video. However, the user has to use different devices to access different networks, call different numbers depending on the location of the called person, and receives multiple bills for the different services. Unified messaging aims at solving this problem by providing the ability to access different networks using a device of one’s choice. It also provides a single mailbox to access messages of different types, such as voice, data, or video. The driving factor for this unified messaging is users’ demands for simple and easy-to-use interfaces for meeting their communication needs.

With increased use of communication facilities, subscribers are demanding a number of services, such as

- ◆ One mailbox for all types of media, not different mailboxes for voice, e-mail, and so forth
- ◆ Access to different services from one device of their choice; the device typically can be a mobile device that one always carries.

- ◆ Simple, easy-to-use interface for accessing different services
- ◆ A single, consolidated bill for all the services, not different bills for different networks
- ◆ A single directory number to call a person irrespective of the location of the called person

In due course, all these will be possible. At present, however, a few of these applications are being made available. We discuss these in the following sections.

Applications of Unified Messaging

The following are some of the applications of unified messaging that use speech recognition, text-to-speech conversion, and interactive voice response systems.

Voice messaging

If a called party does not want to be reached or is not available, voice mail is left in the voice mailbox. However, presently, the voice mailboxes are many — at the PSTN service provider, at the mobile service provider, or at the subscriber premises. Instead of so many boxes, a single voice mailbox can be provided that can be used for voice mails from the PSTN or a mobile phone. The voice mailbox can be accessed from the telephone, fixed or mobile, or through a PC.

E-mail

To access text messages, there is a separate mailbox (or multiple mailboxes if one has multiple mail addresses). The voice mailbox can also be used for storing e-mails. In addition, e-mail (mail in text form) can be retrieved through a telephone (fixed or mobile). After one accesses the mailbox, the text is converted into speech through text-to-speech conversion software and then played to the user.

Fax mail

Fax messages can also be stored in the same mailbox as e-mails. Fax messages can be retrieved from the mailbox using a normal fax machine or they can be read through a telephone (of course, with the limitation that the pictures cannot be read). This calls for special software that converts the fax text into normal text and then converts the text into speech.

Short messaging service

Whenever mail (e-mail, voice mail, fax mail, or video mail) arrives in one's mailbox, the user can be alerted through a short message. The user can program to receive the short message on his mobile phone, pager, or on the PC.

Call forwarding

Nowadays, call forwarding is supported on many networks. A person can program his mobile device for forwarding all the calls to a fixed line or vice versa. This allows a person to be in touch with office/home all the time, and the calling party is saved the bother of trying different numbers.

Voice dialing

Voice dialing presents real advantages when it comes to easily accessing mailboxes or other telephones. However, with the present technology, the user has to train the device for his voice to obtain good recognition accuracy.

Interactive voice response systems

Users can access information available in databases using IVR from mobile phones in speech form. The user can interact with the database (without operator intervention) and obtain the information regarding, say, bank account or credit cards. Advanced IVR systems also facilitate receiving fax messages on demand. The fax message can be routed to a normal fax machine or a message box.

Video messaging

Presently, video messaging is not widespread because video occupies a large bandwidth, and if low bit rates are used, the resulting quality is poor. However, with good video-streaming technologies presently in development, desktop video conferencing is becoming popular. As soon as the Internet backbone can support higher data rates, video messaging will be extensively used, and it will be an integral part of unified messaging.

Exciting times are ahead due to the unified messaging. With it you can communicate with anyone, anywhere, anytime by using just one number and with any device of your choice.

Precise Location-Based Services

We discussed various location-based services earlier in the chapter. Based on the location of the mobile device, location-specific services can be provided such as information about hotels, restaurants, hospitals, and so forth.

Location-based services also help in navigation. Your location information can also be used by a network operator to route the calls. For instance, if you are at home, the MSC of your mobile network will know that you are at home (as the MSC keeps track of your mobile phone), and the incoming calls to your mobile device can be routed to your home telephone number.

Another possibility is to use Bluetooth technology for location identification. When you are at home, the location of your Bluetooth-enabled mobile device is known to the mobile operator's MSC; when a call is made to your mobile phone, it can be automatically forwarded by the MSC to the home telephone. (Note that the user must use the call-forwarding feature for this to work.)

The issues related to privacy will be of paramount importance in providing location-specific services. Location-based services have yet to emerge on a large-scale commercial level, so we need to watch how location-based services will be received by the public.

To provide all these exciting services, we next need to address the issue of the capabilities of the mobile devices.

Mobile Devices

During the last few years, a lot of research has gone into making more "intelligent" mobile devices. The 2G mobile devices were mostly voice-only devices that supported two-way voice communication. Now there is a shift from voice to voice and data, a shift from black-and-white monitors to color monitors, a shift from low processing power to high processing power. Also, there is a shift from Europe and North America to Asia where the mobile phone market growth rate is very high. The Asian market demands mobile devices that are capable of handling text content in different languages. Providing content in regional languages is a challenge particularly for content providers.

In the arena of mobile devices, there are two schools of thought and, accordingly, two types of devices. According to one school of thought, the mobile device needs just a browser (a micro-browser, if the capability of the device is small) and any application or content can be downloaded from the network servers. Those who believe that "the network is the computer" (a slogan popularized by Sun Microsystems) feel that it is enough if the mobile device runs a browser. Java phones are based on this concept. The Java programs can be downloaded from the server to the mobile device, and content can be presented to the user. According to the second school of thought, the mobile device needs to run an Operating System (OS). The OSs that have been developed for the mobile device market include Palm OS, Symbian's EPOC, and Microsoft's Stinger, which is the optimized version of Win CE for mobile devices. Stinger has Outlook companion, which is the mobile version of Outlook Express and Mobile Internet Explorer, which can interpret WML and HTML. Certainly, devices with and without operating

systems will be in use, because the cost considerations will ultimately contribute to the decision of the end users.

Tools for Content Development

As we have seen in the earlier chapters, the content for the wireless networks is available in different forms — WML, HTML, XHTML, XML/XSL, Java programs, and so on. Making content available to different mobile devices based on the characteristics of the device is a great challenge. At present, Internet content access through mobile devices has not been a big success, mainly because the content is not presented in a very user-friendly manner due to varying device characteristics. Ideally, the server at the content provider should obtain the capabilities of the mobile device. Based on these capabilities (display size, monochrome or color display, graphics capability in terms of pixels, whether the device is Java-enabled, the browser running on the device, the protocols supported by the device, and so forth), the content has to be presented to the user in an appealing way. This calls for a database at the server that stores the mobile subscribers' device capabilities. When the user tries to access the data, the server will consult the database, obtain the capabilities, and format content in a suitable format and send it to the device.

Many people still dislike accessing Internet content and seeing text messages on the mobile device display. For such people, presenting information through voice is a good option. With the use of text-to-speech conversion and speech recognition; such systems are already in use. To provide such services, content creation requires a new markup language, which is VoiceXML.

VoiceXML

The VoiceXML forum (www.voicexmlforum.org) was founded by AT&T, IBM, Lucent Technologies, and Motorola to promote Voice Extensible Markup Language (VoiceXML). VoiceXML has been designed to make Internet content available through voice from telephones (both mobile and fixed telephones). VoiceXML, in short, makes it possible to achieve a “voice-enabled Web.” VoiceXML version 1.0 was released in March 2000.

Access to the Web is normally achieved through a desktop PC. The information obtained is rich in content and graphics. But the PC penetration is very low in many areas of the world, particularly in developing countries; computer literacy is also a must in order to access Web services. Accessing the Web through the mobile phone using WAP protocols is another alternative, but WAP-enabled mobile phones are costly. Because of the limited display on the mobile phones, WAP services are not user-friendly.

If Web services were accessible through normal telephones or mobile phones, with the output in voice form, Web reach could be much more extensive, and the services would still be user-friendly because speech is a very natural way of communication among humans. VoiceXML provides this possibility.

Consider a simple example of obtaining weather information from an Internet Web server. The dialogue between the computer (C) and the human (H) can take one of the two forms: directed dialogue and mixed initiative dialogue.

- ◆ Directed dialogue: In this approach, the interaction between C and H can be as follows:
 - C: Please tell the state for which you want the weather information.
 - H: Illinois.
 - C: Please tell the city.
 - H: Chicago.
 - C: The maximum temperature in Chicago is 40 degrees.
- ◆ Mixed initiative dialogue: In this approach, the interaction between C and H can be as follows:

- C: Please tell the city and state for which you want the weather information.
- H: Chicago, Illinois.
- C: The maximum temperature in Chicago is forty degrees.

This kind of interaction is possible (completely through speech) for obtaining information available on the Web. This calls for interfacing a text-to-speech conversion system, speech recognition system, and also, if required, an IVR system to the Web servers. It is possible to provide voice-enabled Web service without VoiceXML as well, but because all these components are built around proprietary hardware and software, it is difficult to port the application for different platforms.

VoiceXML has been designed with the following goals:

- ◆ To integrates voice services and data services.
- ◆ To separates the service logic (CGI scripts) to access the databases, interfaces with legacy databases, and so on, from the user interaction code (VoiceXML).
- ◆ To promotes service portability across implementation platforms because VoiceXML is a common language for content providers, tool providers, and platform providers.
- ◆ To shield the application developers from low-level platform-dependent details, such as hardware and software for text-to-speech conversion, IVR digit recognition, and speech recognition.

The operation of voice-enabled Web is shown in Figure 14-4. The VoiceXML server contains the necessary hardware and software for telephone interface, speech recognition, text-to-speech conversion, and audio play/record. The Web server contains the information required for the specific application in the form of VoiceXML documents along with the service logic in the form of CGI scripts and necessary database interfaces. When a user calls an assigned telephone number to access, say, the weather information through PSTN or PLMN, the call reaches the VoiceXML server and this server converts the telephone number to a Uniform Resource Locator (URL). This server obtains the information corresponding to the URL from the Web server, which is in the format of VoiceXML. VoiceXML server converts the content into speech format and plays it to the user. When the user utters some words (for example, the city and the state for obtaining the weather information), the VoiceXML server recognizes these words and, based on the information available in the database, plays the information to the user.

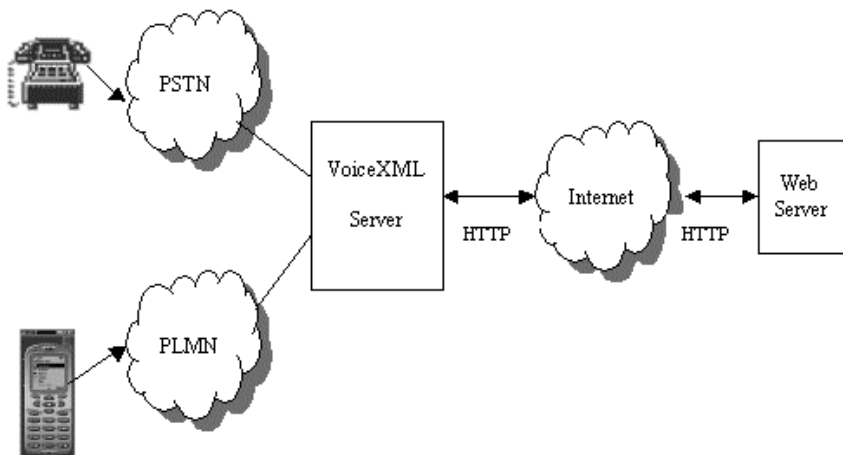


Figure 14-4: Architecture of voice-enabled Web

The VoiceXML server is capable of doing the following functions to provide information in speech form:

- ◆ Recognition of the digits dialed by the user from the fixed or mobile phone

- ◆ Recognition of the words spoken by the user
- ◆ Conversion of the text obtained from the Web server into speech form using text-to-speech conversion software and playing the speech to the user
- ◆ Recording of the speech input by the user (if the user wants to leave a message)

VoiceXML has been developed so that content can be written by a content developer and can be interfaced to any hardware/software used for text-to-speech conversion and speech recognition. VoiceXML provides the means of interaction with the user through forms and menus. Forms collect values for a set of variables (for instance, city and state for weather information) and menus provide the users with a set of choices (for instance, selection of cosmetics, fashion-wear, or jewelry in an m-commerce application).

VoiceXML provides a simple yet efficient method of providing content for developing voice-enabled Web applications. In the next decade, these services will catch up, allowing for very user-friendly Web browsing through telephones.

SyncML

A person using multiple devices (desktop, laptop, mobile phone, and so forth) encounters a very serious problem: The information in the various devices may not be the same. For instance, the appointments stored in the mobile handset and laptop may be different, but they need to be synchronized with each other. Similarly, the contact information (addresses) stored in the desktop and laptop need to be synchronized periodically so that both the devices contain the same data. The same goes for “to do” lists that are stored on different devices. Similarly, the files on different devices need to be in synchronization (contain the same data). Synchronizing information and updating applications between the information on the network and the devices themselves is generally done manually, or in some cases through proprietary solutions developed by different vendors. Synchronization Markup Language (SyncML) is an industry initiative to develop a data synchronization protocol. SyncML standardization activity has been initiated by IBM, Lotus, Motorola, Nokia, Palm Inc, Psion, and Starfish Software. SyncML defines the protocols to locate and update information on the fly. Exchanging information about the updates and resolving the conflicts between the data on the network and the device is known as *data synchronization*.

SyncML defines the data formats and the protocols to synchronize the data. The data can be personal data, such as contact information (called vCard) or calendar information (called vCalendar), or e-mails, network news, XML, HTML documents, and so forth.

The protocol stack for SyncML is shown in Figure 14-5. SyncML is designed to run on different protocol stacks, such as TCP/IP and HTTP, WAP (WSP), and Bluetooth on the client (the mobile device). When a mobile device has to synchronize the data with a server (say, a desktop), the user invokes the Sync client application, and the Sync client agent software communicates with the Sync server software through SyncML protocols to carry out the synchronization and, if necessary, update the information.

Go to www.SyncML.org for the latest information on SyncML.

Protocols

The present protocol stack that is running on the mobile devices for wireless Internet access is not very efficient and is designed for only low-speed networks. In the future, mobile devices, because of their higher processing capability, can run protocols with better functionality and also can be more heavy weight.

The WAP protocol stack has been developed mainly because the TCP/IP protocol stack requires lot of processing to be done on the mobile devices. However, because mobile devices are now capable of higher processing power with more memory, running the TCP/IP stack on a mobile device will not be difficult. Embedding networking protocols in mobile devices is now a distinct possibility. With this, IP

can run on mobile devices to provide IP-based services, such as Voice over IP, fax over IP, and video over IP, which allow low-cost voice, fax, and video communication over the Internet through mobile devices. However, with the unprecedented growth of the Internet and mobile devices capable of accessing the Internet, two important changes are required in the IP: the new version of IP (which has been introduced) and mobile IP.

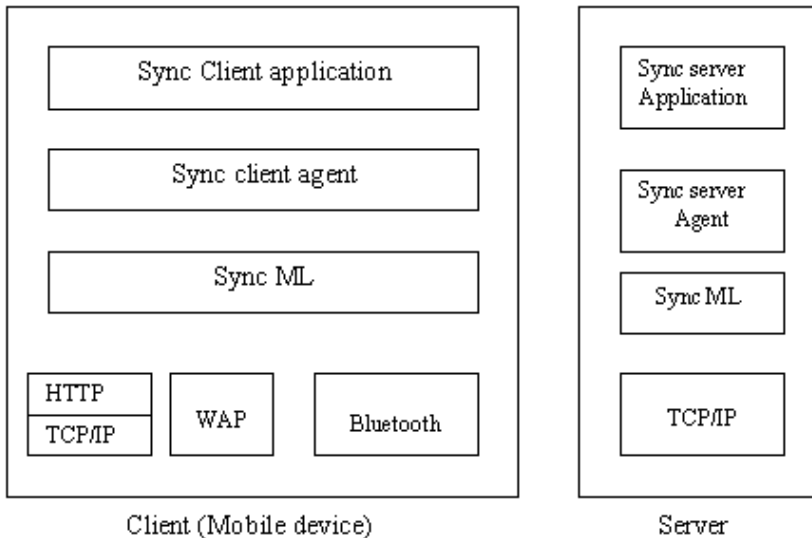


Figure 14-5: SyncML Protocol stack

IP version 6

The IP that is currently running on the Internet is IP version 4 (abbreviated IPv4). IPv4 supports 32-bit address, meaning, each device connected to the Internet is assigned a 32-bit unique address. With this addressing capability, at most 4 billion addresses can be given. Now we want every mobile device, every TV, every laptop, and so on to be connected to the Internet, and this addressing capability is not sufficient any more. IP version 4 has the following limitations:

- ◆ Limited addressing capability. Thus, if we want every mobile device to also have an IP address, the 32-bit address format is not sufficient, and we need to enhance the address length.
- ◆ The IP in its present form has a header field which is fixed, and the routers need to do lot of processing to route the packets to the correct destination. So, fast packet transmission is not guaranteed, and there will also be a delay. Hence, in the present form, IP is not well suited for real time audio and video transmission.
- ◆ Applications (such as e-commerce and mobile commerce) require high security, which is not provided in the present version.

To overcome these problems, IP version 6 has been released (IP version 5 is used only at a few Internet sites). In IPv6, each device is given a 128-bit address. In addition, the IPv6 provides a number of additional advantages:

- ◆ Increased security features through authentication and encryption
- ◆ Modified header format to reduce the processing at the routers so that delay can be minimized
- ◆ Support for resource allocation to facilitate real-time audio and video transmission
- ◆ Support of unicast, multicast, and anycast addressing formats. Unicast implies sending a packet to a specific address, muticast implies sending a packet to multiple addresses (which is required in

applications, such as audio/video conferencing) and anycast implies sending a packet to any address.

Presently, software on hosts and routers is being upgraded for supporting IPv6. However, this is a big job because millions of hosts and routers are to be upgraded. Compatibility with IPv4 is provided — an address with 96 zero bits followed by 32-bits of IPv4 address. However, a translator software is required for conversion of IPv4 packets (called datagrams in IP documents) into IP6 packets.

Mobile IP

In the wired Internet, when two systems have to exchange data, first a TCP connection is established between the two systems and packets are exchanged. The IP addresses of the source and destination, together with the TCP port numbers on the two systems, help in the routing of the packets from the source to the destination. The IP address contains the network address as a part of it. A router analyzes the incoming packet for the destination address, takes out the network address, and routes the packet to that network. The network to which the system is attached is called the home network.

In the wireless Internet, this scheme does not work because the mobile device keeps changing the location, and, hence, the point of attachment changes. Thus, the packets cannot be routed properly to the mobile device. Hence the need for a new protocol at IP level arises, which is called the Mobile IP. The Mobile IP is defined in the RFC 2002 of Internet Engineering Task Force (IETF).

A mobile node is given two addresses: a fixed (or static) IP address called the home address and a care of address that changes at each point of attachment. The static address is to identify the TCP connection and the care of address is to identify the point of attachment (the present network to which it is connected). So, mobile IP requires the existence of a network node called Home Agent (HA), which is the permanent address. When the mobile device is not attached to its Home Network, Home Agent gets all the packets addressed to the mobile node and then forwards them to the present point of attachment, which is known as the Foreign Agent (FA). Whenever the mobile device changes its point of attachment, it registers its care of address with the home agent. The packets are then forwarded to the care of address by the Home Agent. The Mobile IP mechanism is depicted in Figure 14-6. Initially, the Mobile Device (MD) is attached to the HA. When it is on the move, it reaches an FA locality. The FA keeps advertising its service. The MD requests the service to the FA. FA relays the request to the HA, and the HA can either reject or accept the request. If the HA accepts the request, the care of address is used to redirect all the packets received to the MD through the FA to the MD. The functions of the Mobile IP functions are

- ◆ Discovering the care of address
- ◆ Registering the care of address
- ◆ Redirection to the care of address

Mobile IP has been implemented by a number of vendors, but presently security is a concern. IPv6 and Mobile IP together will provide the required features to carry out secure transactions over mobile devices. Because of IP, the Internet is a very powerful tool; the same power will now be available on mobile devices through Mobile IP.

4G Systems

Now that the 3G technologies are standardized and the data rates are fixed, we will have access to multimedia services over mobile devices. Now, the equipment manufacturers and the operators are focusing on the fourth-generation (4G) wireless networks. 4G is still at a conceptual stage as far as network architecture and protocols definitions are concerned, but the present focus is on the kind of services that can be provided to the users. Certainly the data rates will be in the range of 2 Mbps to 8 Mbps. This allows very high-resolution graphics, high-fidelity audio, and broadcast quality video services to be provided to mobile users.

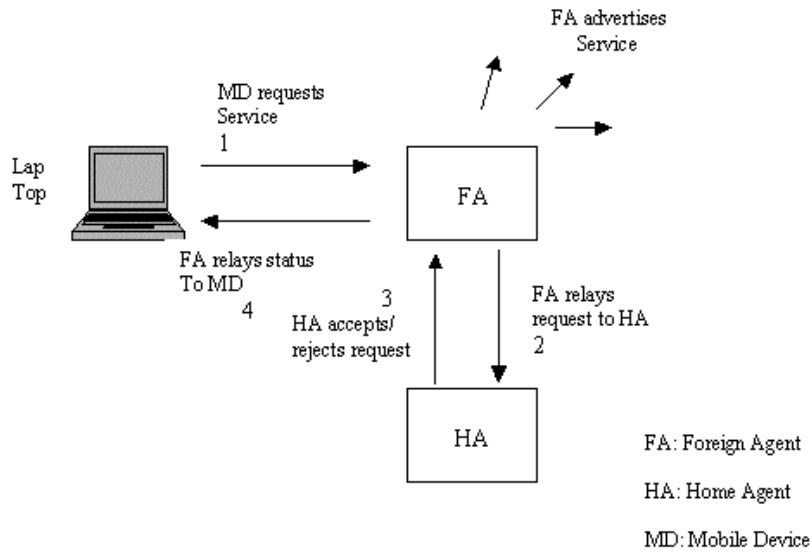


Figure 14-6: Mobile IP

Summary

In this chapter, we discussed the technology trends to provide value-added services for the end users through the mobile networks. The convergence of telecommunication networks and services is paving the way for many value-added services to be available to the users, such as instant messaging, unified messaging, precise location-based services, and accessing the Web through telephones in voice format. The IP version 6 and Mobile IP will provide the protocol infrastructure for IP services to be made available on mobile devices with the necessary security. Applications of the high bandwidth services will be in many areas, such as collaborative working, multi-party audio and video conferencing, virtual classrooms, telemedicine, and much more.

Appendix A

What's on the CD-ROM

This appendix provides information about this book's companion CD-ROM, found on the inside back cover of the book. For the latest information, please refer to the ReadMe file located at the root of the CD.

System Requirements

This book's CD-ROM runs on Microsoft Windows 95, 98, 2000. Your computer must be equipped with a CD-ROM drive that is double-speed (2x) or faster. If your computer doesn't match up to these requirements, you may have a problem using the contents of the CD.

CD Contents

The CD-ROM contains source code examples, applications, and an electronic version of the book. The following is the summary of the contents of the CD-ROM:

Source Code

The folder named "Source Code" is categorized into different folders named according to the chapter numbers. The source code of the case studies and the programs are contained in their respective folders. Following is the list of the folders in the source code folder:

- ◆ Chapter 2: This folder contains the two folders: "Information Master Application" and "Restaurant Application." These folders contain the source code of the application. These case studies are built using WML and WMLScript.
- ◆ Chapter 3: This folder contains the source code for Question Quiz Application. This project is built using Cold Fusion with WAP.
- ◆ Chapter 4: This folder contains the source code for the WTA program, which illustrates WTAI function call. This example is built using WML.
- ◆ Chapter 5: This folder contains the source code of the Weather Application. This application is built using the Servlet, JDBC and WML.
- ◆ Chapter 6: This folder contains the two folders: "Pushing the stock quotes" and "Shopping cart with advertisement push." These folders contain the source code of the application. These case studies are built using WML, HTML and Java.
- ◆ Chapter 8: This folder contains the two folders: "Airport Kiosk" and "Shopping Mall Kiosk." These folders contain the source code of the applications. These case studies are built using ASP, WML, and WML Script.
- ◆ Chapter 9: This folder contains the following folders, which contain the source code of their respective projects. All these projects are built using the Bluetooth Development Kit. This is using VC++ programming style.

- HCI Programming.
- SDP Programming
- File Transfer Application
- Chat Application
- ◆ Chapter 11: This folder contains the source code of all the programs explained in this chapter. All these programs are developed by using XML, XHTML, WML, ASP, XSL, and Java.
- ◆ Chapter 12: This folder contains the following folders, which contain the source code of their respective Projects. All these projects are built using the Brew toolkit provided by Qualcomm.
 - A New Application using Brew
 - Developing Animation Application
 - Application for music Downloading onto a Mobile
 - Mobile Advertisement Application
 - Database Application
- ◆ Chapter 13: This folder contains the following folders, which contain the source code of their respective projects. All these projects are built using JMF (Java Media Framework)
 - Voice Messaging Application
 - Audio Broadcasting Application
 - Audio-Video Broadcasting Application

Applications

The following applications are on the CD-ROM:

- ◆ **Nokia WAP Software** folder contains the following:
 - Nokia WAP Toolkit 2.1: This toolkit Provides developers, the PC environment required for developing and testing WAP applications. It offers the tools needed for developing WML and WMLScript content, adding graphics etc. there by equipping them fully to avail the push functionality.
 - Nokia Activ Server 2.0 Professional Edition: This is an open software platform that offers secure mobile connectivity to a company's current information systems, both intranet and internet. WAP-Enabled services may be connected to the Nokia Activ Server using Circuit Switched Data (CSD) and also Short Message Services (SMS).
- ◆ **Tomcat Server** folder contains the Tomcat Server:
 - Tomcat Server 3.0: With the tomcat environment, development of Javaserivlets and JSP are possible without having to install a full-fledged web server as entailed by ASP, CGI, Perl etc. In the tomcat environment all classes are available, as suited to the server side Java programming environment.
- ◆ **Macromedia** folder contains the following:
 - Macromedia Cold Fusion Studio 4.5 Enterprise Edition: Cold Fusion Studio provides an integrated development environment for Cold Fusion applications. The studio is optimized to suit development of Cold Fusion- based web sites and applications. Some of the prominent features of this studio are: Project management, Code Snippets, Expression Builder, Visual database tools, Validation tools, Code debugging, Design layout and page preview.
 - Macromedia Homesite 4.5: Homesite is an HTML editor with an award to its credit. It makes for creating websites better and at the expense of lesser time. By Virtue of this HTML editor you can integrate money lending web technologies such as JSP,CFML and WML.

- ◆ **Java Developers Kit** folder contains the following:
 - **Forte for Java release 2.0-** The Forte for Java release 2.0 software is an integrated development environment used for developing Java application. It is an IDE provided by Sun microsystem.
 - **Java 2 SDK-** This is a software development kit for Java standard edition required for developing Java applications.
- ◆ **Acrobat Reader** folder contains the Acrobat Reader 5.0
 - **Acrobat Reader 5.0-** This software enables you to view the Adobe PDF files over a liberal range of hardware and operating systems. You can avail the Acrobat reader 5.0 for adding digital signature to files even while remaining connected to your web browser or for converting your office documents to Adobe PDF file for the Acrobat Reader for Palm OS.

E-Book

Those readers, who desire an electronic copy of the contents in the book, can avail the CD-ROM, which accompanies this book. This CD-ROM contains the PDF files of all the chapters as well as the appendices in the book. These files can be viewed through the Acrobat Reader 5.0 software, which has been incorporated in the CD-ROM.

Troubleshooting

If you have trouble installing or using the CD-ROM programs, then try the following solutions:

- ◆ **Turn off any anti-virus software that you may have running.** Installers sometimes mimic virus activity and can make your computer incorrectly believe that it is attacked by a virus. (Be sure to turn the anti-virus software back on later.)
- ◆ **Close all running programs.** The more programs you're running, the less memory is available to the other programs. Installers also typically update files and programs; if you keep other programs running, installation may not work properly.

If you are still having trouble with the CD, please call Hungry Minds Customer Service at (800) 762-2974. If you are not in the United States, call (317) 572-3994. You can also contact Hungry Minds Customer Service by e-mail at techsupdum@hungryminds.com. Please note that Hungry Minds will provide only technical support for installation and other general quality control items. For technical support on the applications themselves, please consult the program's vendor or author.

Appendix B

Tomcat Installation and Configuration

Introduction to a Web Server

Web servers allow you to serve content over the Internet using the HyperText Markup Language (HTML). In this process, the Web server responds to a request made by the browser in the appropriate HTML document. The Web server can accept requests from various browsers, such as Netscape and Internet Explorer and then respond to the appropriate HTML documents. A number of server-side technologies can be used to increase the power of the server to deliver standard HTML pages; these include CGI Scripts, Server-Side Includes, Java Server Pages (JSP), Servlets, SSL, and Active Server Pages (ASP).

How a Web Server Works: An Overview

When a browser sends a request for a Web page, such as `http://www.yahoo.com/index.html`, to the Web server, it maps the `index.html` from the computer. After mapping it, the server sends the response to the user. The communication done between the user and the server, that is, the request and the response are handled by the HTTP (HyperText Transfer Protocol), which is also known as the request response protocol. This workflow is shown in the Figure B-1.

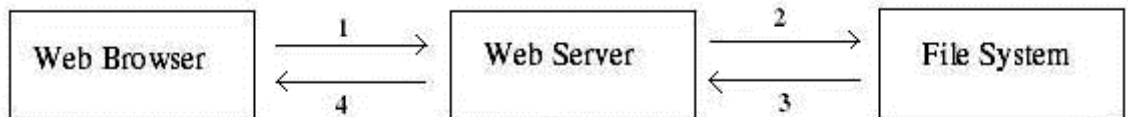


Figure B-1: Web server handling HTML request

This process in Figure B-1 works as follows:

1. User requests the document (`index.html`).
2. Web server looks for the document on the file system.
3. Web server maps the file from the file system.
4. Web server returns the data document to the Web browser.

This simple arrangement was the initial concept behind the World Wide Web, which allows the handling of static content such as HTML. The exchange of complex and vital information between browsers and Web servers at the global level became possible by virtue of the WWW. But the most important part of the Web is dynamic content. The Common Gateway Interface (CGI) is the oldest and the most popular standard for dynamic content. It basically works as an interface for the user's request for dynamic pages. The request is converted into an HTML response by the Web server, which sends it back to the user browser. Nowadays, various technologies are emerging that handle the user requests, which are to be processed by the Web server and sent back to the browser in the minimum possible time. We can

implement ASP, JSP, servlets, and so on, instead of CGI. Figure B-2 shows how the Web server works with these technologies.



Figure B-2: Web server handling JSP or servlet request

The process in Figure B-2 works as follows:

1. User requests a servlet or JSP Program (`hello.jsp`).
2. Web server receives the request.
3. Web server launches and processes `hello.jsp`.
4. Web server sends its parameters to the browser requested.
5. Web server retrieves the output from the `hello.jsp` program.
6. Web server passes the output from `hello.jsp` program to the browser.

Introduction to the Tomcat Web Server

Tomcat is used as the Web server for implementing Java Servlet 2.2 and Java Server Pages 1.1 technologies. It is a servlet container, which is a runtime shell that manages and invokes servlets on behalf of users.

TOMCAT is an extension of the Apache-server, but it runs independently of it. You can also use the Tomcat server as an extension of the IIS Server. Thus, when you use TOMCAT 3.1, it actually runs on your machine as a process separate from the APACHE. When configured correctly APACHE will continue serving HTML pages, while TOMCAT actually serves the JSP pages and runs the servlets that your site contains.

Tomcat includes a small HTTP server front end that can be used instead of APACHE. Additionally, the interface between the HTTP server and Tomcat is a published standard that can be implemented by components outside the Tomcat project. This means it is possible to make Tomcat work with any HTTP Web server.

Install the Tomcat Web Server

The Tomcat Web server installation is explained as follows.

Step 1: Download

To configure Tomcat as a stand-alone server you need to download the **Tomcat 3.2.1** and the **JDK 1.3 Standard Edition**. Tomcat server can be downloaded from:

<http://jakarta.apache.org/builds/jakarta-tomcat/release/>. You can download the different versions of Tomcat server from this site. You can also check for the latest version of Tomcat server on <http://jakarta.apache.org/site/binindex.html>. When you visit the Tomcat site you will find the files in different formats, such as TAR, HQX, and so forth. To install Tomcat on Windows, you must download the zip file. You can download JDK 1.3 Standard Edition from: <http://java.sun.com/products>.

Step 2: Installing on Windows NT/2000

Unzip the file onto the hard disk (say D:). This should create a new subdirectory named “tomcat”. After you have extracted Tomcat, you have to put your JDK into Tomcat's CLASSPATH and set the TOMCAT_HOME environment variable. To do this under NT/2000, you must open the Control Panel. (See Figure B-3).

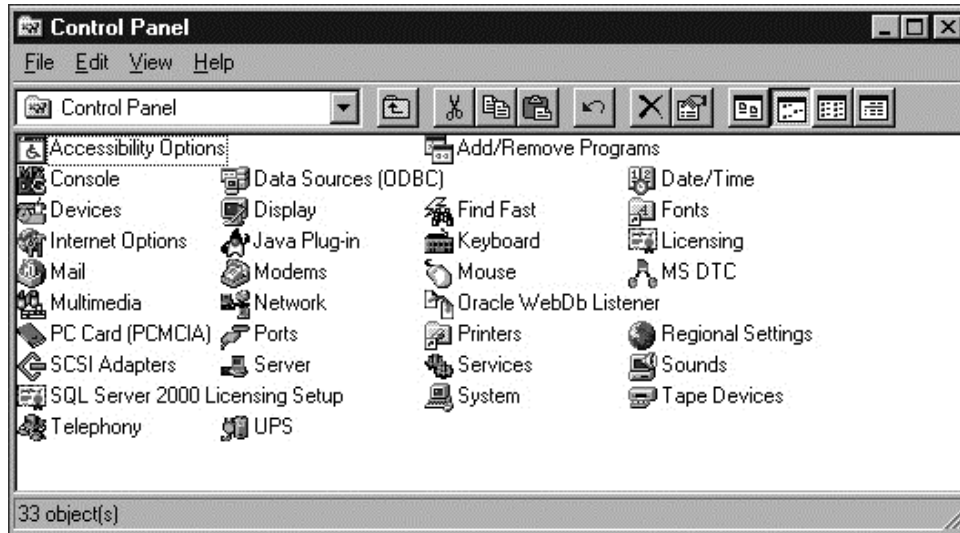


Figure B-3: NT/2000 Control Panel

Using Windows 2000

Go to Start⇒System Properties and select the Advanced tab (see Figure B-4).

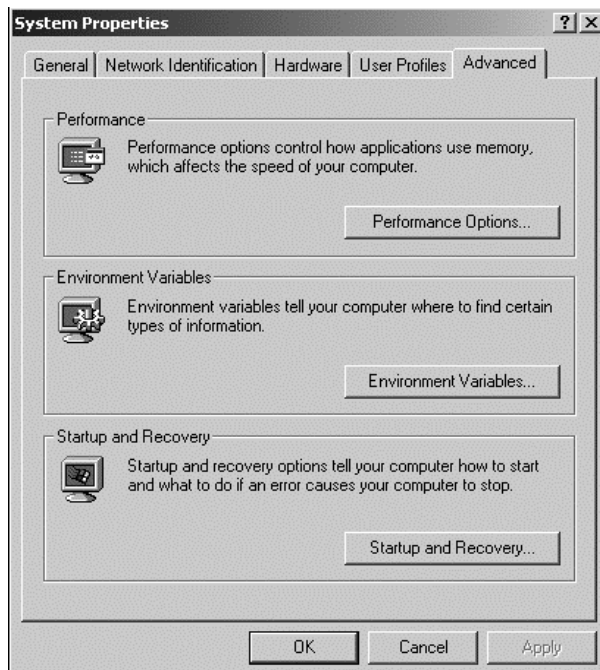


Figure B-4: Windows 2000 System Properties

Click the Environment Variables button on the System Properties dialog box. You will then see the screen in Figure B-5.

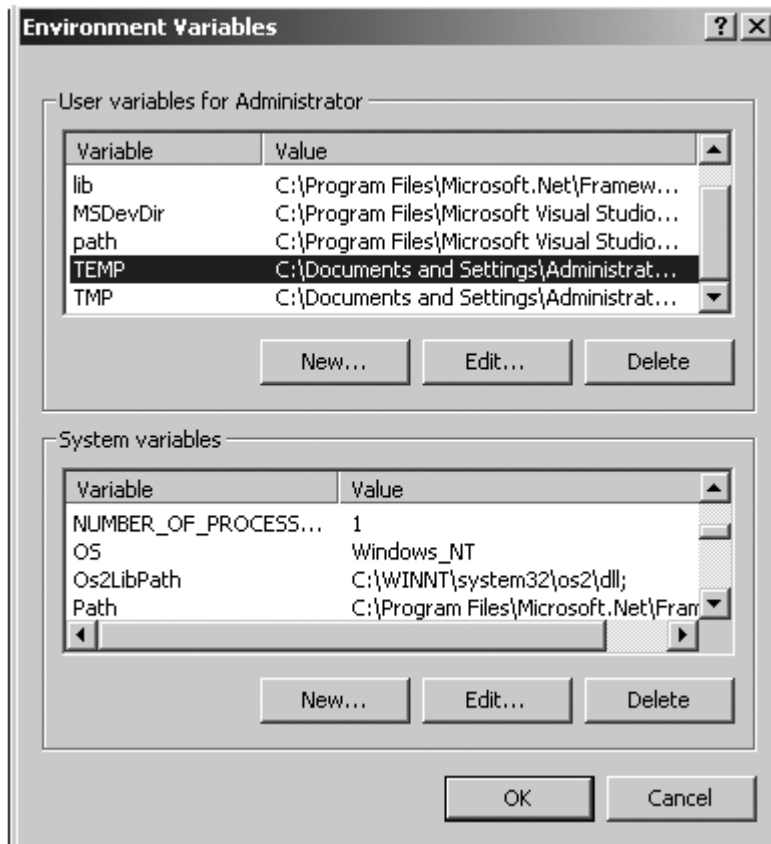


Figure B-5: The Environment Variables dialog box of Windows 2000

Now click the New button under System variables. Type **JAVA_HOME** as the Variable Name and set the path of the JDK in the Variable Value field (see Figure B-6).

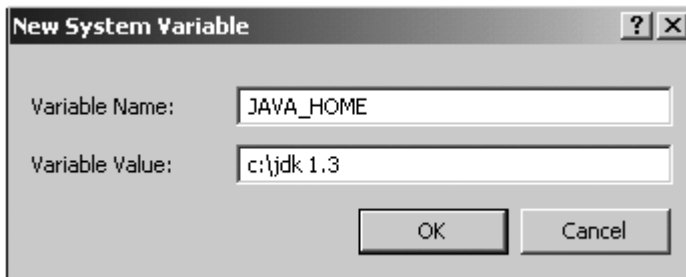


Figure B-6: JAVA_HOME Environment Settings for Windows 2000

Repeat the previous step, but this time add **TOMCAT_HOME** as the Variable Name and set the Variable Value as **D:\jakarta-tomcat-3.2.1**.

Using Windows NT

Go to Start⇒System Properties and select the Environment tab, as shown in Figure B-7.

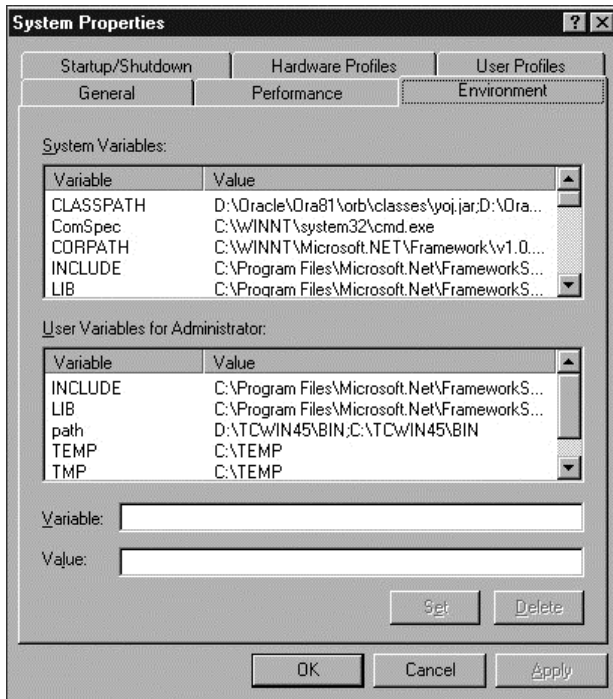


Figure B-7: Windows NT System Properties

Type **JAVA_HOME** in the Variable field and set the location of your JDK in the Value field (see Figure B-8).

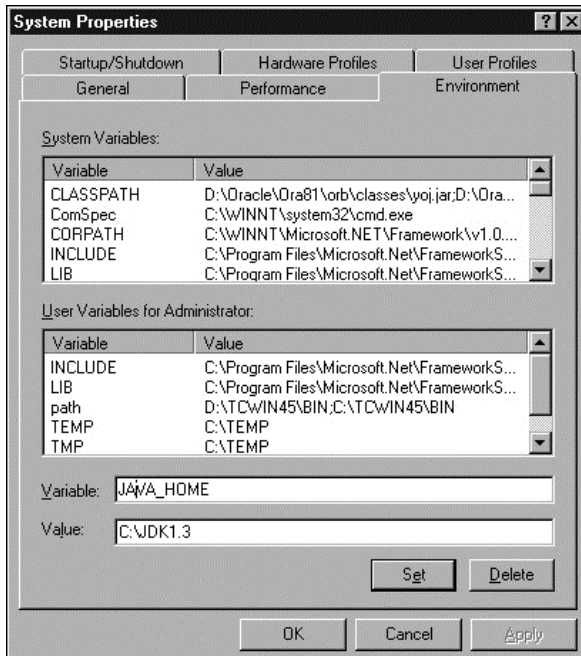


Figure B-8: JAVA_HOME Environment Settings for Windows NT

482 Appendix B: Tomcat Installation and Configuration

Repeat the previous step by typing **TOMCAT_HOME** in the variable field and **D:\jakarta-tomcat-3.2.1** in the Value field.

You can also set these environment variables in the file `tomcat.bat` located in the Jakarta-tomcat-tomcat-3.2.1 folder. To do this, open the file in Notepad and write the following below the line `rem ---`
-- Save Environment Variables That May Change -----:

```
Set JAVA_HOME=c:\jdk1.3
Set TOMCAT_HOME=d:\ jakarta-tomcat-3.2.1
```

Using Windows 98

The value for the DOS environment space is 1024 bytes. If you receive a message, such as “out of environment space” on your command prompt when you are running the `tomcat.bat` file, you have to make changes in the `config.sys` file. Go to your command prompt and open the `config.sys` file (`c:/>edit config.sys`) and make the following changes:

```
shell=c:\command.com /p /e:4096
```

This command changes the size of environment space from 1024 bytes to 4096 bytes.

Restart your computer after making this change.

Additional environment variables

Some changes need to be made to the `autoexec.bat`, as shown here:

```
SET PATH=C:\PERL\BIN;c:\jdk122\bin;
SET TOMCAT_HOME=c:\TOMCAT
SET JAVA_HOME=c:\jdk122
SET CLASSPATH=c:\jdk122\lib\tools.jar
```

- ◆ Include the path of JDK
- ◆ Set the path `TOMCAT_HOME` and `JAVA_HOME` environment variables to the directories where Tomcat and the JDK are installed. Note: Do not use “;” at the end of `TOMCAT_HOME` and `JAVA_HOME`
- ◆ Also include the classpath of the `tools.jar` file. You will get this file in the `lib` folder of the JDK

Table B-1 shows the Tomcat startup and shutdown commands.

Table B-1: Tomcat Startup/Shutdown Commands

Startup	Shutdown
<code>TOMCAT_HOME\bin\startup.bat</code>	<code>TOMCAT_HOME\bin\shutdown.bat</code>

You can start the Tomcat server by clicking the `startup.bat` file. After a few seconds, you will see the screen in Figure B-9.

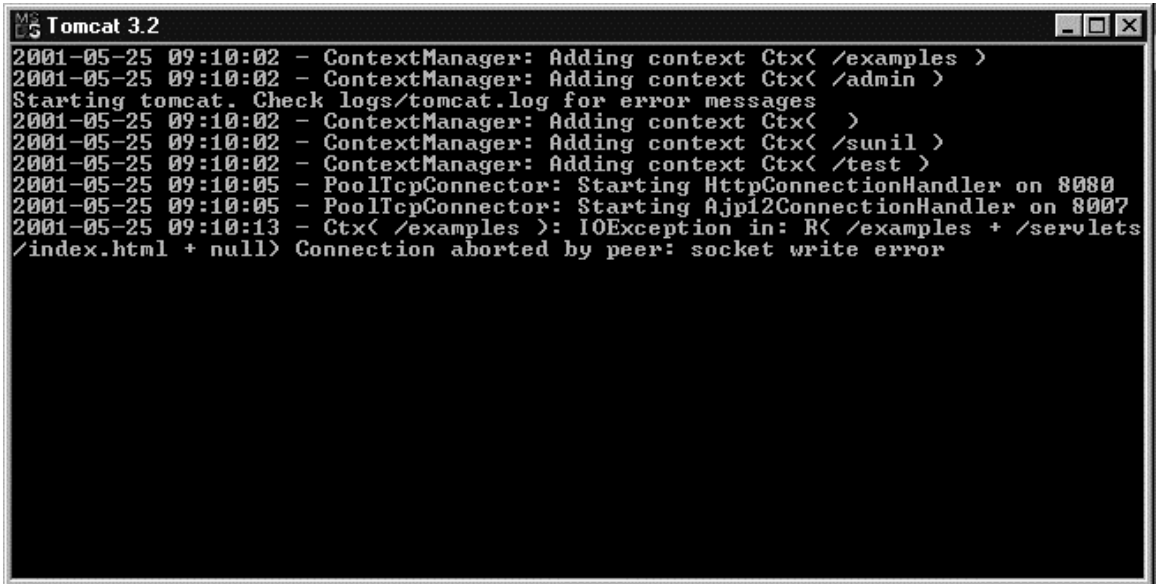


Figure B-9: The starting window for Tomcat Web server

After starting Tomcat, enter the following in the URL of your browser.

`http://localhost: 8080/`

You then see the screen in Figure B-10.

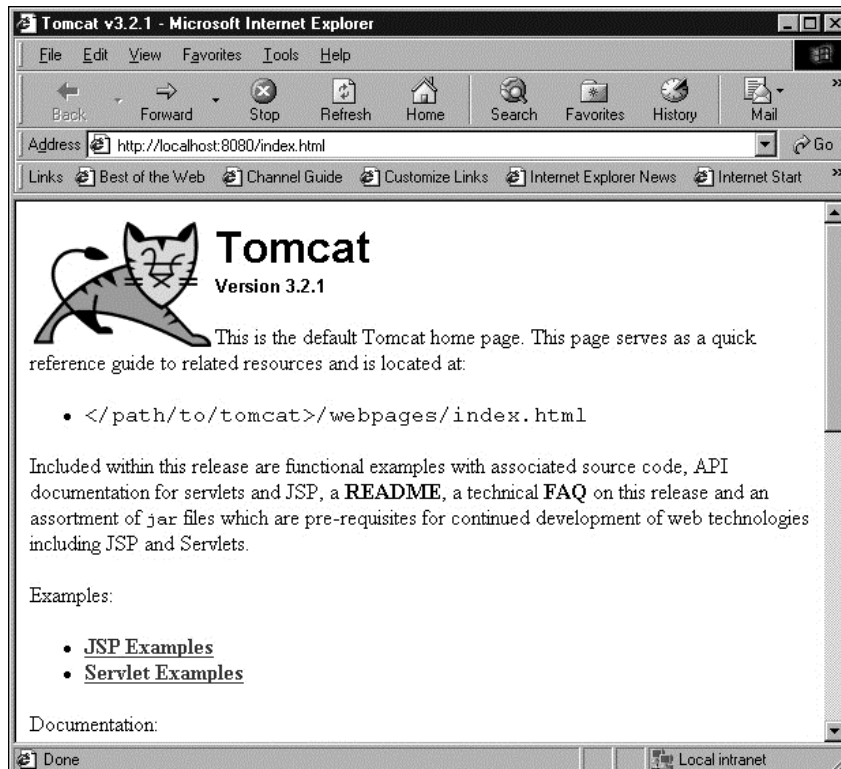


Figure B-10: Home page for Tomcat Web server

484 Appendix B: Tomcat Installation and Configuration

To verify the installation of JDK, you may execute any of the examples given in the home page of the Tomcat server. As shown in Figure B-10, choose JSP Examples. You get a page similar to that shown in Figure B-11.

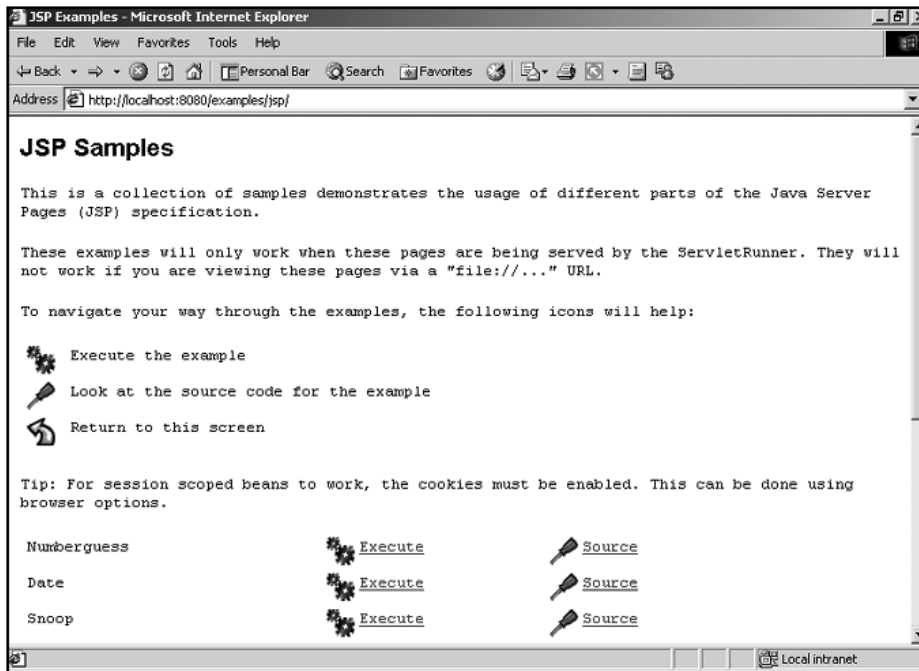


Figure B-11: The JSP Samples page

Now click any of the execute links, for example, Date. You get the results shown in Figure B-12.

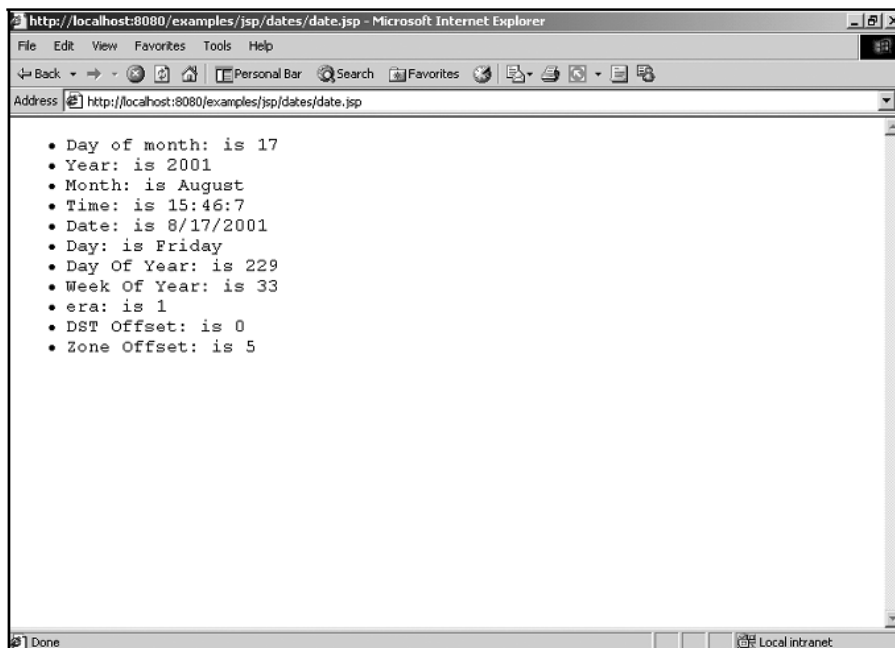


Figure B-12: The JSP Date page

If you get an error message instead of Figure B-12, it means that the path of JAVA_HOME has not been set properly. Check the value of JAVA_HOME and the location of the JDK.

You can stop the Tomcat server by clicking on the shutdown.bat file. You then see the screen in Figure B-13 within a few seconds.

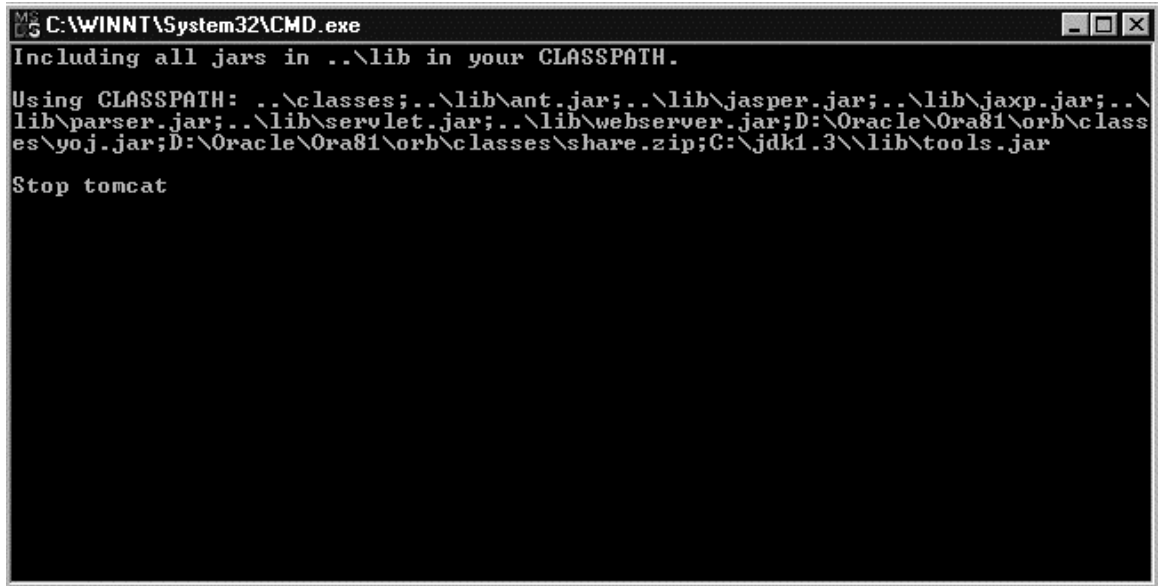


Figure B-13: Stopping window for Tomcat Web server

The default port to run the Tomcat server is 8080 and can be changed. For example, you can change the HTTP port to **80**, from port **8080** by making the following changes in TOMCAT_HOME/conf/server.xml file and then restarting the Tomcat server.

Change

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
<Parameter name="handler"
value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
<Parameter name="port"
value="8080"/>
</Connector>
```

to

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
<Parameter name="handler"
value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
<Parameter name="port"
value="80"/>
</Connector>
```

To use the new setting of Tomcat, shut down the server first by clicking shutdown.bat file and restart by clicking startup.bat.

Now type the following in the URL:

```
http://localhost/
```

You will see results similar to those in Figure B-7.

Deploy Web Applications to Tomcat

After Tomcat is installed and is running, we can deploy a Web application to it. To deploy a Web application, first understand the directory structure of Tomcat. Table B-2 describes the directories that make up a Tomcat installation. It is assumed that the value of `TOMCAT_HOME` precedes each of these directories.

Because we are using a beta release of Tomcat, these directories can change without notice.

Table B-2: The Tomcat Directory Structure

<i>Directory</i>	<i>Explanation</i>
<code>/bin</code>	Contains the startup and shutdown files for Windows as well as Linux OS
<code>/conf</code>	Contains the main configuration files for Tomcat, such as <code>server.xml</code> and <code>web.xml</code>
<code>/server</code>	Contains the Tomcat Java Archive files
<code>/lib</code>	Contains Java Archive files. Tomcat is dependent upon these Java Archive files.
<code>/logs</code>	Contains log files of Tomcat
<code>/src</code>	Contains the source code used by the Tomcat server, which will probably contain only interfaces and abstract classes at the time of Tomcat's release
<code>/webapps</code>	Contains all Web applications and the WAR file
<code>/work</code>	After you execute the JSP file the first time, the servlet generated by the JSP is placed in this directory

Web Application

A Web application is defined as a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors. In other words, a Web application is anything that resides in the Web layer of an application.

The main feature of a Web application is the relationship with `ServletContext`. To avoid clashing between two Web applications, different `ServletContext` are used for different Web applications. The servlet container controls this relationship. Besides servlets, a Web application can have JSP pages, utility classes, static documents (such as HTML, images, and so forth), client-side classes, and other meta information describing the Web application.

Directory structure

To create a Web application, you have to first create the directory structure in which it exists. For example, we are creating the Web application by the name of the Web. The Web contains each and every component as discussed in Table B-3.

Table B-3: The Web Application Directory Structure

<i>Directory</i>	<i>Explanation</i>
<code>/web</code>	The root directory of the Web application and contains all JSP and XHTML files
<code>/web/WEB-INF</code>	Contains all resources related to the application that are not in the document root of the application. It contains your Web application deployment descriptor. It is not part of the public document, so the files of this directory can be served directly to a client.

/web/WEB-INF/classes	Contains servlet and utility classes
/web/WEB-INF/lib	Contains Java Archive files. Tomcat is dependent upon these Java Archive files. For example, you can place a JAR file that contains a JDBC driver.

You can store the classes either in `/WEB-INF/classes` or in `/WEB-INF/lib` directories. The class loader loads the classes of `/classes` directory first followed by the JARs in the `/lib` directory. If both the folders have the class file with the same name, then classes of the `\classes` will be used.

Deployment descriptor of Web application

Deployment descriptor is the heart of all Web applications. It is an XML file by the name `web.xml` stored in the `<SERVER_ROOT>/applicationname/WEB-INF/` directory. `web.xml` carries information for the entire Web application. You can edit or modify the configuration of a `web.xml` file. For our application the location of the `web.xml` file is in the `<SERVER_ROOT>/web /WEB-INF/` directory. The deployment descriptor contains information about the following elements:

- ◆ ServletContext Init Parameters
- ◆ Session Configuration
- ◆ Localized Content
- ◆ Servlet / JSP Definitions and Mapping
- ◆ Mime Type Mappings
- ◆ Welcome File list
- ◆ Security
- ◆ Error Pages

The following code has some of the elements of a Web application deployment descriptor.

```
<web-app>
  <display-name>The web App</display-name>
  <session-timeout>45</session-timeout>
  <servlet>
    <servlet-name>TryServlet</servlet-name>
    <servlet-class>com.web.TryServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </init-param>
  </servlet>
</web-app>
```

where

- ◆ `<display-name>`: The first of the application level elements, it describes the name of the Web application and is functionally inoperative.
- ◆ `<session-timeout>`: The second Web application level element; it controls the lifetime of the application's `HttpSession` object. The `<session-timeout>` value that we have used in the previous code tells the JSP/Servlet container that the `HttpSession` object will become invalid after 30 minutes of inactivity.
- ◆ `<servlet>`: The last application level element that we have defined and this element defines a servlet and its properties.

Create the Web application directory structure

The name of our Web application, `web`, is the root of our directory structure. It contains all the files and subdirectories, as discussed in Table B-3.

Create the directory directly in the `Tomcat /webapps` directory while in development. When it comes for deployment, you can package your Web application into a WAR file and go through the production deployment process.

The last step in creating the Web application directory structure is adding a deployment descriptor. At this point you will be creating a default `web.xml` file that contains only the DTD, describing the `web.xml` file, and an empty element. Listing B-1 contains the source for a default `web.xml` file.

Listing B-1: `web.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
</web-app>
```

Now copy this file to the `TOMCAT_HOME/web/WEB-INF/` directory. We will begin adding web application components to it in the following sections.

Deploy a Web Application to Tomcat

First create the `web` folder in the `TOMCAT_HOME/webapps` folder and then add a new context entry to the `TOMCAT_HOME/conf/server.xml` file, setting the values for the `path` and `docBase` to the name of your Web application.

```
<Context path="/web" docBase="webapps/web" debug="0 "
reloadable="true" />
```

Restart Tomcat after completing these steps. Your application should now be running.

The previously described application can be accessed by pointing your browser at:

```
http://localhost/web
```

Figure B-14 does not show any contents, as the `web` folder does not carry any file (such as HTML, JSP, and so on).



Figure B-14: Tomcat Web server on default port 80

Appendix C

SQL Server 2000 Installation and XML Support Configuration

In this appendix, we will explain how to install SQL Server 2000 and XML support configuration. Before this, we give a brief introduction of SQL Server 2000.

About MS SQL Server 2000

SQL Server 2000 is the database engine product by Microsoft. It is equipped with the power of Relational Database Management System (RDBMS). SQL Server 2000 runs on Windows NT (with Service Pack 5 or above), Window 98, and Windows 2000. Microsoft has also launched SQL 2000 on Windows CE. It is called SQL 2000 CE.

Different editions of SQL Server 2000

SQL Server is available in the following editions:

Enterprise Edition

SQL Server 2000 Enterprise Edition can be used as a production database server for big plants, factories, corporations, and so forth. It supports all features that are available in SQL Server 2000, as well as the very good performance required in large Web sites. It also supports the OLTP (Online transaction processing) feature and data warehousing systems.

Standard Edition

This edition is used as a database server for smaller businesses with smaller databases (as compared to large corporations). This edition is also good for small work groups and individual departments of larger businesses.

Personal Edition

This edition is meant for users who spend more time disconnected from their network of computers. It is capable of running a stand-alone application that requires a local database at a client computer.

Developer Edition

Programmers use the Developer Edition to develop applications that use SQL Server 2000 as a database. It contains all the features of the Enterprise Edition, and it is only licensed for testing and development purposes for the programmers. It is not used as a production server.

Windows CE Edition

This edition is used to store data on Windows CE devices. It can replicate data pertaining to any edition of SQL Server 2000 to synchronize Windows CE data with the primary database.

SQL Server 2000 Enterprise Evaluation Edition

You can download the full-featured version of SQL 2000 from the Microsoft site for the evaluation purpose. The evaluation period for this edition is 120 days.

Platform Choice

SQL Server 2000 runs on Microsoft Windows NT because Microsoft Windows NT (the operating system required for SQL Server) is capable of running on different platforms.

SQL server 2000 can run on the following platforms:

- ◆ Intel x86 based processor
- ◆ Digital Alpha processor

HAL (Hardware Abstraction Layer) makes Window NT 4.0 platform independent so that it can run on an Intel x86 based processor as well as a Digital Alpha Server. Windows NT can support 32 processors in a single server. Now large queries can be divided into different processes that make the server performance faster.

Minimum Installation Requirements

The minimum requirements for installing Microsoft SQL Server 2000 are as follows:

Hardware requirements

Following are the minimum hardware requirements for installing Microsoft SQL Server 2000 or SQL Server client management tools and libraries:

- ◆ Computer: Intel Pentium 166 or higher; DEC Alpha and compatible systems
- ◆ Memory (RAM)
 - Enterprise Edition: 64MB minimum but 128MB is highly recommended
 - Standard Edition: 64MB minimum
 - Personal Edition: 64MB minimum on Windows 2000 and a minimum of 32MB on all other operating systems
 - Developer Edition: 64MB minimum
 - Desktop Engine: 64MB minimum on Windows 2000 and a minimum of 32MB minimum on all other operating systems
- ◆ Hard disk space
 - SQL Server database components: 95 to 270MB although 250MB is generally used
 - Analysis Services: 50MB minimum although 130MB is generally used
 - English Query: 80MB
 - Desktop Engine only: 44MB
- ◆ Monitor: VGA or higher resolution of 800 x 600 because a higher resolution is required for the SQL Server graphical tools
- ◆ Pointing device: Microsoft mouse or compatible
- ◆ CD-ROM drive: Required

Operating system requirements

The various editions or components of Microsoft SQL Server 2000 runs on the computer installed with following operating systems:

- ◆ Enterprise Edition
 - Microsoft Windows NT Server 4.0
 - Microsoft Windows NT Server Enterprise Edition 4.0
 - Windows 2000 Server
 - Windows 2000 Advanced Server
 - Windows 2000 Data Center Server
- ◆ Standard Edition
 - Microsoft Windows NT Server 4.0
 - Windows 2000 Server
 - Microsoft Windows NT Server Enterprise Edition
 - Windows 2000 Advanced Server
 - Windows 2000 Data Center Server
- ◆ Personal Edition
 - Microsoft Windows Me
 - Windows 98
 - Windows NT Workstation 4.0
 - Windows 2000 Professional
 - Microsoft Windows NT Server 4.0
 - Windows 2000 Server
 - All the more advanced Windows operating systems
 - Developer Edition
 - Microsoft Windows NT Workstation 4.0
 - Windows 2000 Professional
 - All other Windows NT and Windows 2000 operating systems
- ◆ Client Tools Only
 - Microsoft Windows NT 4.0
 - Windows 2000 (all versions)
 - Windows Me and Windows 98
- ◆ Connectivity Only
 - Microsoft Windows NT 4.0
 - Windows 2000 (all versions)
 - Windows Me
 - Windows 98

- Windows 95

Complete Installation of SQL Server 2000 (Setup)

1. Insert the SQL Server 2000 CD in CD-ROM.

Insert the SQL Server 2000 compact disc into your CD-ROM drive. The CD that contains SQL Server 2000 automatically starts up after you put it in. If the CD ROM doesn't start automatically, it could mean that the auto start functionality of your CD ROM is disabled. Consequently, you can file the `AUTORUN.exe` file in the root directory of SQL Server 2000. You may also execute this file by double-clicking on it, which starts the installation wizard for the installation of SQL Server 2000. Thus, SQL Server 2000 Components installs the SQL 2000 on your server. (See Figure C-1).

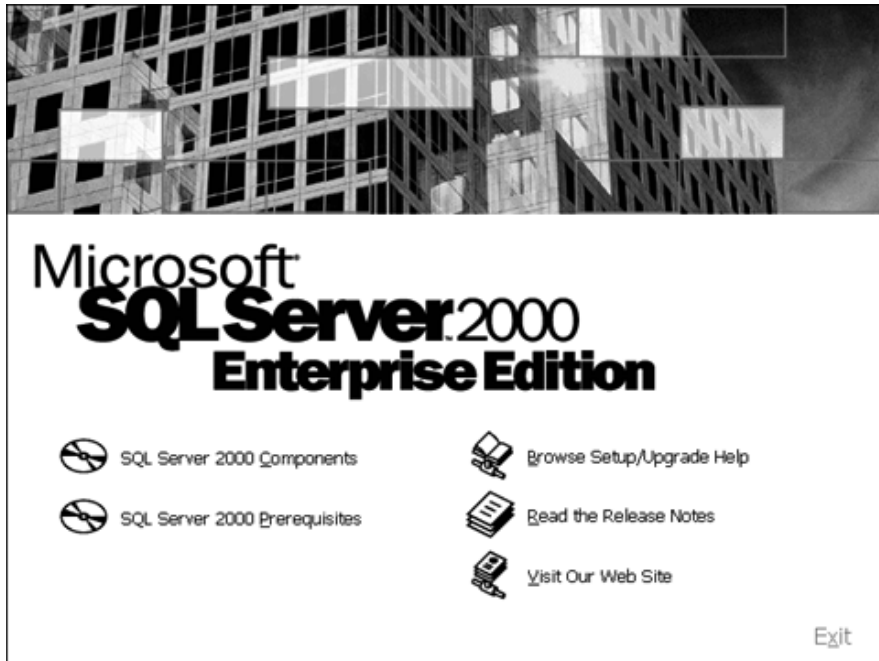


Figure C-1: Auto run menu for SQL 2000

The elements in Figure C-1 are explained as follows:

- ◆ SQL Server 2000 Components installs the SQL 2000 on your server.
- ◆ If you are running Microsoft Windows 95, click SQL Server 2000 Prerequisites and then click Install Common Controls Library Update.
- ◆ The Browse Setup/Upgrade Help opens the books available online with SQL 2000.
- ◆ Read the Release Notes opens the release notes.
- ◆ Visit Our Web Site takes you to the Microsoft site, provided you are connected to the Internet. After you're there, you can choose SQL Server 2000 Components.

2. Install components.

The Install Components screen is shown in Figure C-2. They are explained as follows:

- ◆ Install Database Server installs the SQL 2000 database.
- ◆ Install Analysis Services installs the analysis server, which helps in Online Analytical Processing (OLAP) and data mining applications.

- ◆ Install English Query allows the developer to build the application that provides the end-user with the ability to pose questions in English rather than asking questions with SQL statements.

Select Install Database Server and setup prepares the SQL Server Installation Wizard at the Welcome screen. Then click Next.



Figure C-2: Install Component for SQL 2000

3. In the Welcome screen, click Next.

Figure C-3 shows the Welcome dialog box.

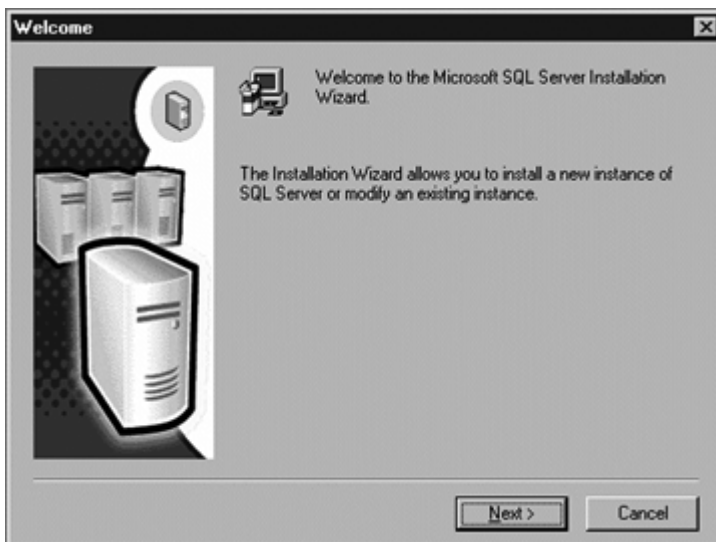


Figure C-3: Welcome screen of SQL 2000

4. Choose the computer name.

Figure C-4 shows the Computer Name dialog box. The individual options are explained as follows:

- ◆ Local Computer: The default option is the name of the computer on which the setup is running. For a local installation, accept the default and click Next.
- ◆ Remote Computer: If you are installing the SQL server 2000 in a network, enter the name of the computer for a remote installation, or click Browse to locate the remote computer.
- ◆ Virtual Server: This option is available in case MSCS (Microsoft Cluster Service) is detected on a Windows 2000 Enterprise or the Windows NT operating system. You can enter the name of a new Virtual SQL Server or that of the existing one..

After you've finished, click Next.

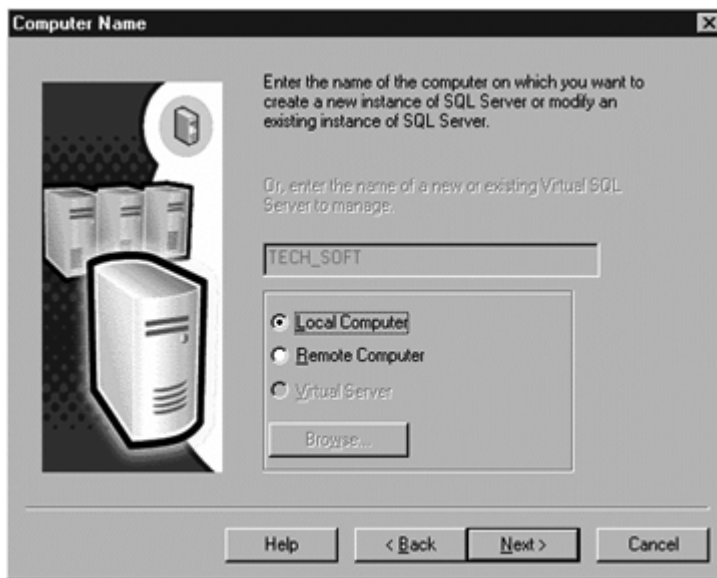


Figure C-4: Choosing the computer name for SQL 2000

5. Installation selection.

Figure C-5 shows the Installation Selection dialog box. The options are as follows:

- ◆ Create a new instance of SQL Server, or install Client Tools: This option is used to create a new installation of SQL 2000 and creates a new instance for SQL 2000 server.
- ◆ Upgrade, remove, or add components to an existing instance of SQL Server: This option allows you to upgrade, remove, or add components to an existing instance of SQL 2000. You can upgrade, remove, or add components in the earlier versions (SQL Server version 6.5 and SQL Server version 7.0) as well as instances of SQL Server 2000.
- ◆ Advanced options: This option is used for unattended setup, cluster maintenance, and registry rebuild.

In the Installation Selection dialog box, click Create a new instance of SQL Server, or install Client Tools, and then click Next.

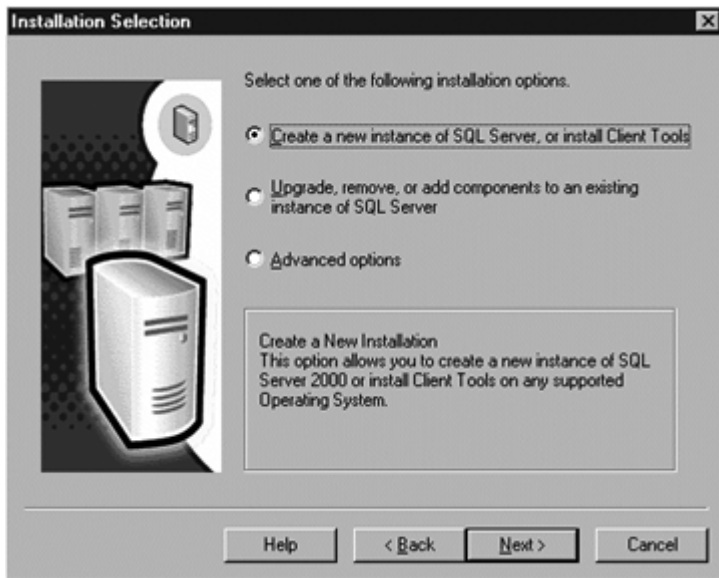


Figure C-5: Installation selection for SQL Server 2000

6. Gather user information.

Figure C-6 shows the User Information dialog box. Enter your name and the name of your company. Here we are using the name *techno campus* and the company name *techno campus ltd.* Then click Next.

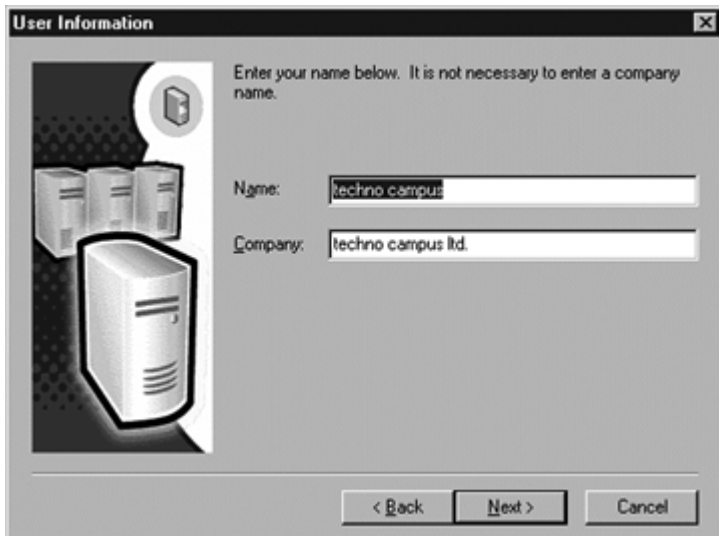


Figure C-6: User Information for SQL Server 2000

7. Software license agreement for SQL Server 2000.

Figure C-7 shows the Software License Agreement dialog box. Read the agreement and then click Yes.



Figure C-7: Software License agreement for SQL Server 2000

8. Select the Installation type.

Figure C-8 shows the Installation Definition screen. The options are as follows:

- ◆ **Client Tools Only:** This option installs the data access components, network libraries, and management tools for SQL Server. You should choose the option Client Tools Only if you intend to use this computer to manage an existing SQL Server remotely.
- ◆ **Server and Client Tools:** This is the default option and performs a complete installation of SQL Server.
- ◆ **Connectivity Only:** This option installs the Microsoft data access components and network libraries but not the management tools. You may use this selection if you want the computer you are installing on to participate in SQL Server communication. If you make this selection, you cannot use your computer as a server or for other management purposes.

In the Installation Definition dialog box, click Server and Client Tools and then click Next.



Figure C-8: Installation Type for SQL 2000

9. Select the instance name.

Figure C-9 shows the Instance Name dialog box. If the Default check box is available, you can install either the default or a named instance. If the Default check box is not available, it means that a default instance has already been installed, and you can install only a named instance.

To install the default instance, select the Default check box and click Next.

To install a named instance, clear the Default check box and type a new named instance in the Instance name edit box. Click Next.

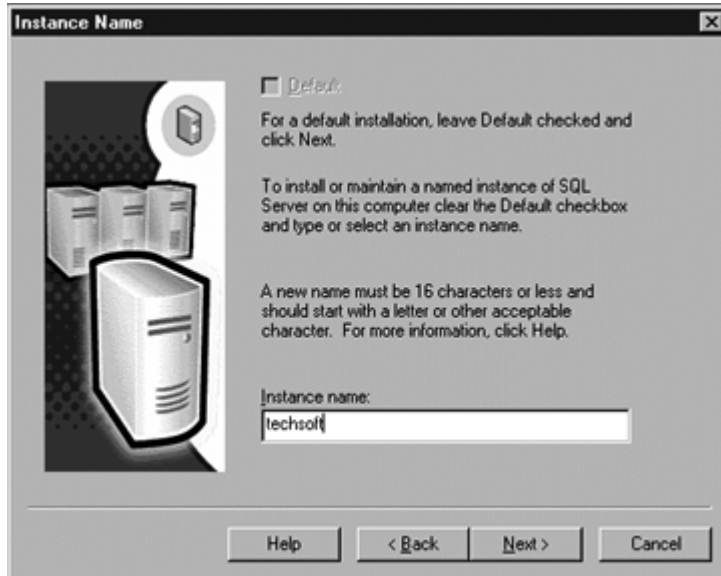


Figure C-9: The Instance Name dialog box for SQL Server 2000

10. Select setup type.

Figure C-10 shows the Setup Type dialog box. The options are as follows:

- ◆ **Typical:** This is the default option for installation. It is generally recommended for users at all levels.
- ◆ **Minimum:** This option installs the minimum components required in a computer to run SQL Server 2000. If your computer doesn't have much space, you can choose this option.
- ◆ **Custom:** This option allows you to choose components and subcomponents or to change settings for collations, service accounts, authentication, or network libraries. If you know all the components and subcomponents with their features and application, then this particular option is recommended for you.

The default location for the installation of SQL Server 2000 is C:\Program Files\Microsoft SQL Server\, for both program as well as data files. You can change the destination for the program and data files by clicking the Browse button.

In the Setup Type dialog box, click Typical or Minimum and then click Next.

If you want to select the components and subcomponents, change character set, network libraries, or other settings, click Custom and then click Next.

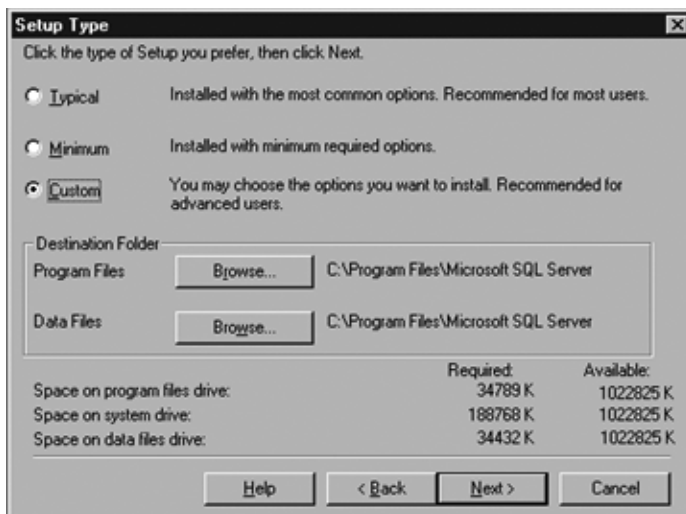


Figure C-10: Selecting the Setup Type dialog box

11. Select the services accounts.

Figure C-11 shows the Services Accounts dialog box. The options are as follows:

- ◆ Use the same account for each service. Auto start SQL Server Service: This option uses one account for both the SQL Server and the SQL Server Agent. These services start automatically when the operating system starts. This is the default option.
- ◆ Customize the settings for each service: This option allows you to use different settings for the two services.
- ◆ Services: If you want, you can select a service for customize settings. The two options available here are SQL Server and SQL Server Agent.
 - SQL Server: This option allows you to select customize settings for the service, Microsoft SQL Server.
 - SQL Server Agent: This option selects customize settings for the service, Microsoft SQL Server Agent.
- ◆ Service Settings: Under this option, you have two sub-options:
 - Use the Local System account: This option does not require a password, does not have network access rights in Windows NT 4.0, and may restrict your SQL Server installation from interacting with other servers for the local system.
 - Use a Domain User account: This option uses Windows Authentication to set up and connect to SQL Server for domain user. By default, it provides for the currently logged on user. It contains the options Username, Password, and Domain.
- ◆ Username: You can either accept or change the username for the domain.
- ◆ Password: Allows you to enter the password for the domain.
- ◆ Domain: Here you can either accept or change the name of the domain.
- ◆ Auto Start Service: If you choose this option, a service starts automatically when you start your computer. By default this option is unavailable and is only available when customizing the settings for each service.

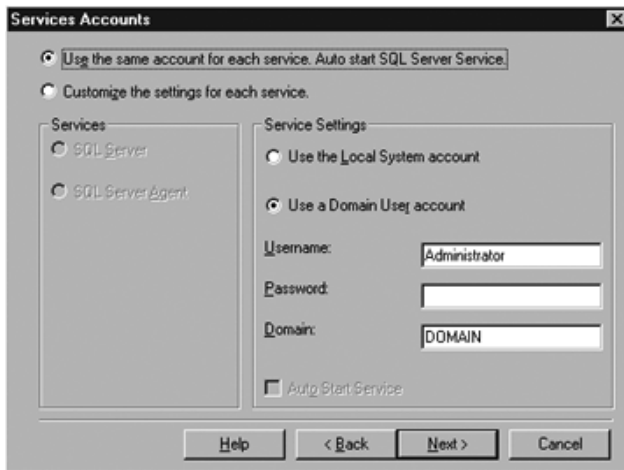


Figure C-11: The Services Accounts dialog box

The SQL Server Agent Service is dependent on the SQL Server service. So you can only autostart the SQL Server Agent Service if you autostart the SQL Server service as well.

In the Service Accounts dialog box, accept the default settings, enter your domain Password, and then click Next.

12. Select the authentication mode.

Figure C-12 shows the Authentication Mode dialog box. The options are as follows:

- ◆ **Windows Authentication Mode:** This is the default option and allows a user to connect using a Windows 2000 or Windows NT 4.0 user account. In this option, SQL Server validates the account name and password by using the Windows NT 2000 or Windows 4.0 for the information.
- ◆ **Mixed Mode:** When a user connects with a specified login and password, SQL Server authenticates the login name and password. It allows the user to connect if the information matches the previously recorded login account set information in the server; otherwise, it sends the error message to the user.

You can choose the Windows Authentication Mode or the Mixed Mode. In the Authentication Mode dialog box, accept the default setting and click Next.

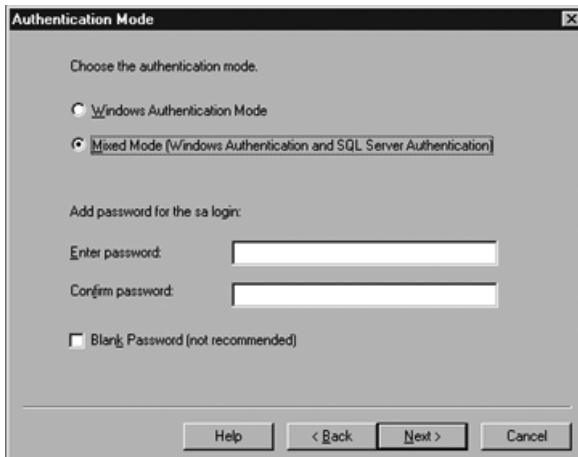


Figure C-12: Dialog box for selecting the Authentication Mode for connection with SQL 2000

13. Start copying the files for the installation of SQL Server 2000.

Figure C-13 shows the Start Copying Files dialog box. After you have finished specifying the options, click Next in the Start Copying Files dialog box.

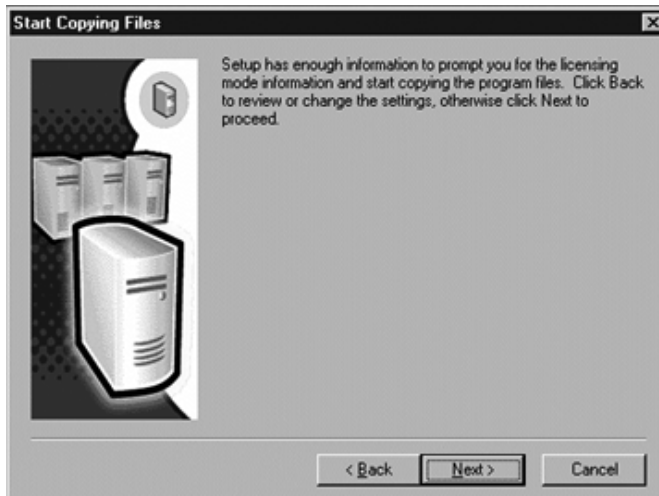


Figure C-13: Start copying the files for installation

14. Installation complete

Figure C-14 shows the Setup Complete dialog box. Here, click Yes, I want to restart my computer now and then click Finish.



Figure C-14: Setup Complete dialog box for SQL Server 2000

XML Support in SQL Server 2000

XML stands for the Extensible Markup Language that describes data. We can describe the data using XML and display it with the help of HTML using style sheets, XSL, and so on. In XML, we can create our own tags instead of using predefined tags, such as those used in HTML. So you have no restriction in

creating the tags. Examples of HTML tags are `<p>` `</p>`, and so forth whereas examples of XML tags are `<user>` `</user>`, `<book>` `</book>`, and so on. The following are a few facts regarding XML:

- ◆ XML was developed in 1996 by an XML Working Group (W3C) chaired by Jon Bosak of Sun Microsystems. In the beginning, the XML group was known as the SGML Editorial Review Board.
- ◆ XML is used to transfer the data over the Internet and is supported by various other applications. For example, if we create a table `exm_emp` containing the empcode, name, address, and phone in SQL, we may save the file as `.xml` (as in `emp.xml`) and view it in the browser.
- ◆ The Document Type Definition (DTD) acts as a rulebook that defines the legal elements of an XML document. DTD defines the document structure with a list of legal elements.
- ◆ A *schema* (in an XML document) is a description of the way in which a document is marked up. The description can be on its grammar, vocabulary structure, datatypes, and so forth.

Using IIS (Internet Information Server) for Accessing SQL Server 2000

The first step for accessing the SQL Server is installing the Virtual Directory for SQL Server.

System Requirement for IIS Virtual Directory Management

You can run the IIS Virtual Directory Management for SQL Server utility on any computer running Microsoft Windows NT 4.0 or Microsoft Windows 2000. For computers running Windows NT 4.0, the following are the essential requirements:

- ◆ Microsoft Internet Information Server 4.0 or higher
- ◆ Microsoft Management Console 1.2

For computers running Microsoft Windows 2000 Professional, the Administrative Tools pack (`Adminpak.msi`) is required. You can locate this file in the `%windir%\System32` folder of the Windows 2000 Server editions.

Steps for Creating a Virtual Directory

1. Go to Start ⇒ Programs ⇒ Microsoft SQL Server program group and click Configure SQL XML Support in IIS (see Figure C-15).

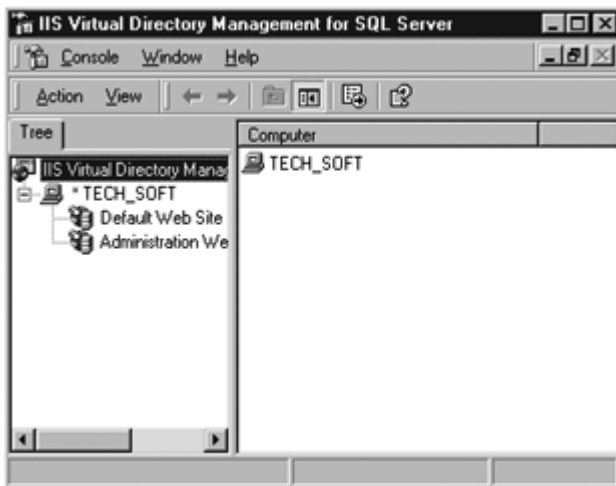


Figure C-15: Setting IIS Virtual Directory Management for SQL 2000

2. Click the plus (+) sign of the server (TECH_SOFT) and then click the Web site that you want. For example, here we are creating a virtual directory on the default Web site. On the Action button, point to New and then click Virtual Directory. You get the New Virtual Directory Properties, as shown in Figure C-16.

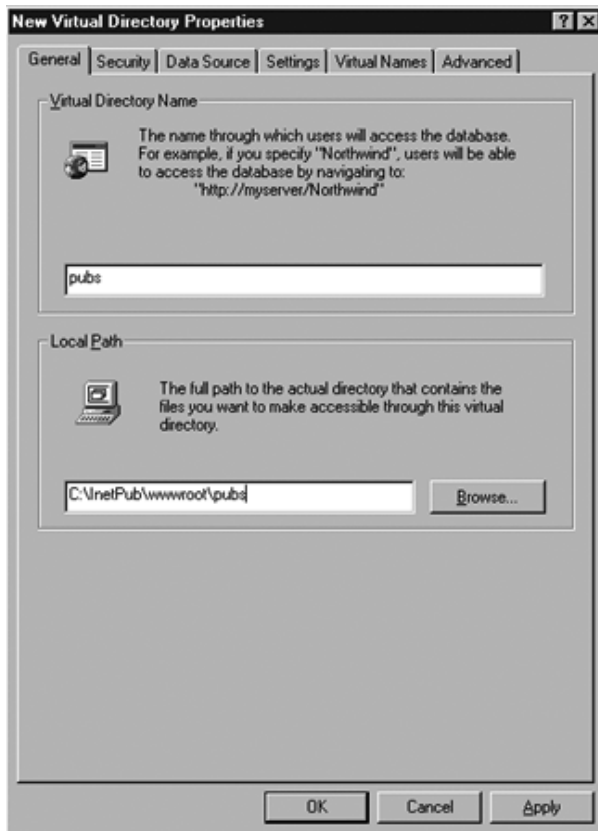


Figure C-16: General Properties Tab IIS Virtual Directory Management

3. Click the General tab of the New Virtual Directory Properties dialog box and enter the name of the virtual directory. Here, we have entered the name **pubs**. Now type the local path or your physical directory or click the Browse button to locate it. For example, we have created a sub-directory with the name pub in the C:\inetpub\Wwwroot\, so you can enter **\inetpub\Wwwroot\pub** in the Local Path.
4. Click the Security tab (Figure C-17) and select the authentication method to access the data. You can either give the login name and a password or use the same name that you use to access the SQL Server 2000. In Figure C-17, we select the Windows Integrated Authentication Option.
5. Click the Data Source tab (Figure C-18). You can either enter the name of a server, such as local or the name of an instance of SQL Server 2000 if more than one instance is installed on the specified computer. In the Database box, enter **pubs** as the name of the default database.

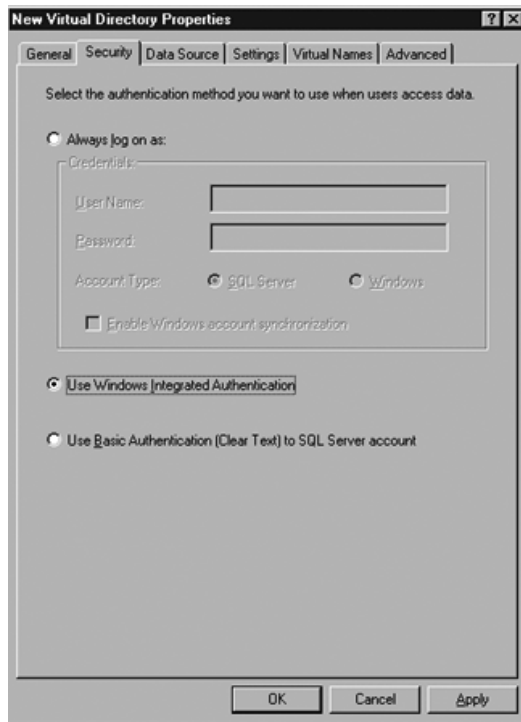


Figure C-17: Security Properties Tab for IIS Virtual Directory Management

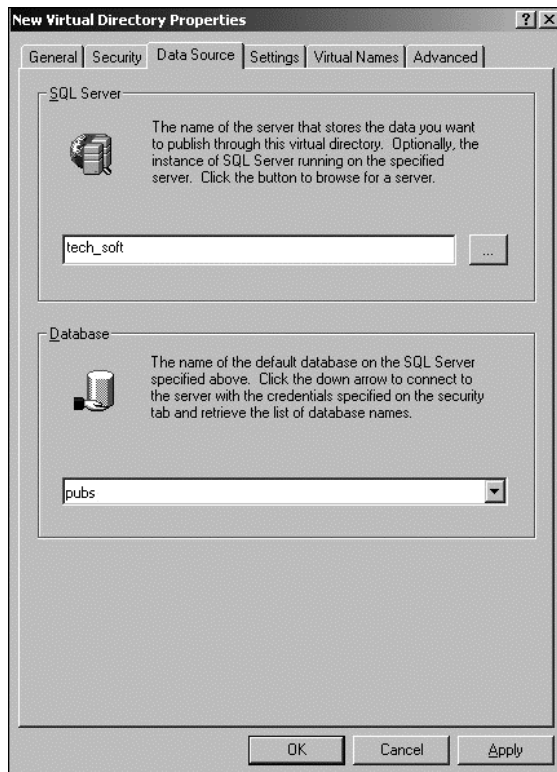


Figure C-18: Data Source for IIS Virtual Directory Management

6. Click the Settings tab (Figure C-19); you can select any or all of the following options:

Allow URL queries allows the user to execute the SQL statement to run directly in the URL, such as `http://tech_soft/pubs?SQL=select+*+from+authors+for+xml+auto`

Allow template queries allows the use and execution of the queries defined in the template file. For example, `http://tech_soft/pubs/myquery.xml`

Allow XPath allows the user to use the XPath query language to submit queries that user SQL Server views. Schema mapping occurs between the XML queries and the SQL server databases. For example: `http://tech_soft/pubs/cust/custlist`

Allow POST allows the user to create an HTML form and then send the data of the form to the server by the post method. It specifies the maximum size of the data passed through the post method.

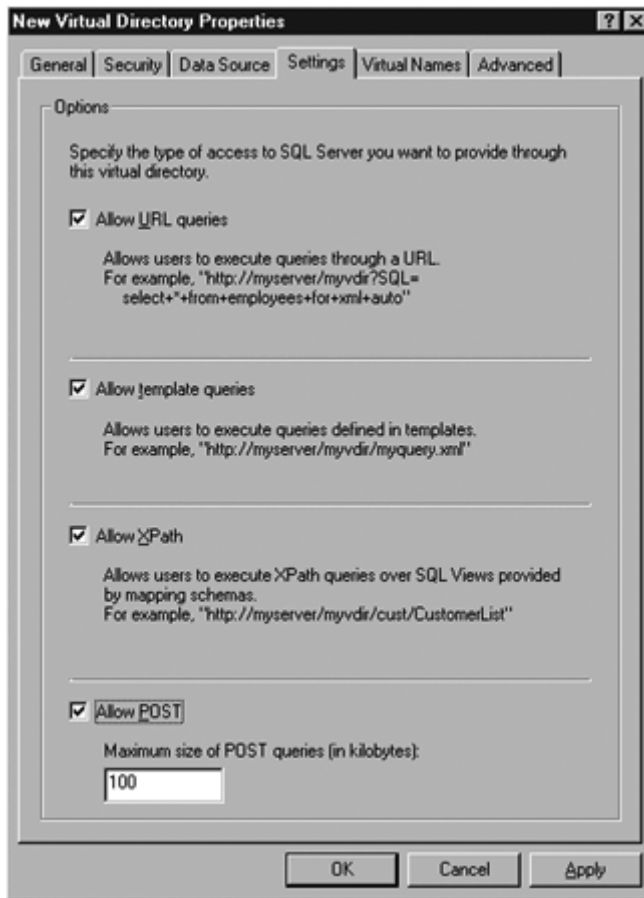


Figure C-19: Settings Tab IIS Virtual Directory Management

7. Click the Virtual Names tab (Figure C-20) and then click New to create the virtual name for the template type.



Figure C-20: Virtual Names IIS Virtual Directory Management

In the Virtual Name Configuration dialog box (Figure C-21), enter the name of the template in the Virtual name box. We use the name template. You can also define any other name of the template. In the Type list, select template. Enter the path (for example, C:\inetpub\Wwwroot\pubs\template, assuming that there is a subdirectory template in the physical directory associated with the virtual directory, however, the existence of the path is not checked). Click Save to save the virtual name.

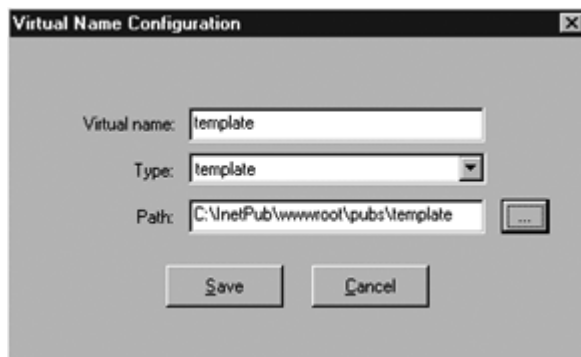


Figure C-21: Creating a template by using Virtual Name Configuration IIS Virtual Directory Management

8. By clicking the Advanced tab (Figure C-22), the user can set the advanced option for the virtual directory. Do not make any changes and click OK to finish creating and configuring the virtual directory.

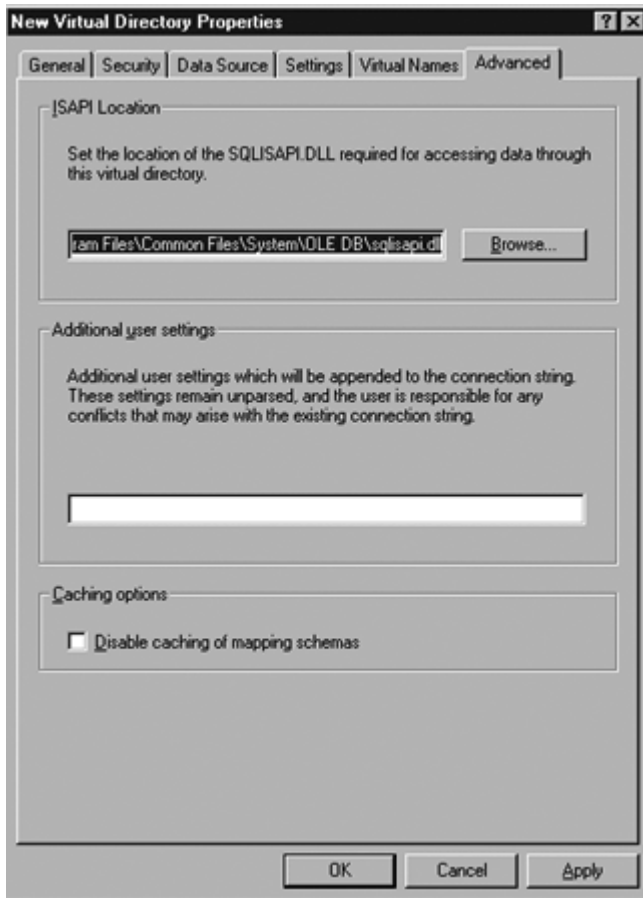


Figure C-22: Advanced Properties tab for the IIS Virtual Directory Management

Appendix D

Bluetooth Reference and Resources

The following table lists the various site addresses furnishing information related to Bluetooth, with a brief statement of what each one provides.

<i>URL</i>	<i>Description</i>
www.anywhereyougo.com/bluetooth	Site provides latest information on wireless technologies including product information, resources for developers including online testing, and a free newsletter
www.bluetooth.com	Official site of Bluetooth Special Interest Group (SIG)
www.comtec.sigma.se	Site of Sigma ComTec AB, Sweden, the distributor of Ericsson's Bluetooth Application Tool Kit.
www.cstack.com	"The Bluetooth site by engineers for engineers"
www.developer.axis.com	An open source implementation of Bluetooth protocol stack
www.ericsson.com/bluetooth	Bluetooth information area of Ericsson
www.homerf.org	Web site for Home RF standards and products
www.irda.org	Web site of Infrared Data Association
www.ivtcorporation.com	Web site of International Validation and Testing Corporation
www.lesswire.de	German firm specializing in location-aware services using Bluetooth
www.lucent.com/micro/bluetooth	Bluetooth information area of Lucent Technologies
www.mecel.se	Web site of Mecel, supplier of Bluetooth protocol stack for embedded systems
www.motorola.com/bluetooth	Bluetooth information area of Motorola
www.nokia.com/bluetooth/index.html	Bluetooth information area of Nokia
www.palowireless.com/bluetooth/devtools.asp	Bluetooth resource center
www.semiconductors.philips.com/bluetooth	Site of Philips Semiconductors, a leading supplier of Bluetooth modules
www.telelogic.com	The site of Telelogic, Sweden based firm specializing in Bluetooth pre-qualification testing

www.topsitelists.com/bestsites/bluetooth	Provides links to the top 25 commercial and non-commercial Bluetooth sites.

Also see the *Bluetooth PC Reference Stack User's Manual* by Ericsson, Release R1C, January 2001.

Appendix E

3G Reference and Resources

The following table lists the various site addresses furnishing information related to 3G, with a brief explanation of what each one provides.

URL	Description
www.the3gportal.com	Web site with rich information on 3G resources
www.3gpp.org	The official site of 3G partnership program. You can find a wealth of information on 3G technologies based on W-CDMA and GSM standards.
www.cdg.org	Web site of CDMA development group
www.de.infowin.org/ACTS	Web site of Advanced Communications Technologies and Services (ACTS) gives information about the pan-European research projects on advanced wireless communications technologies
www.ipn.org	Web site for interplanetary Internet
www.locationforum.org	Web site of Location interoperability forum
www.nttdocomo.com	NTTDoCoMo's site gives information on I-mode and FOMA
www.openwap.org	Web site provides links to downloads of WAP tool kits and discussion lists
www.sss-mag.com/w-cdma1.html	Articles and resources on CDMA
www.tiaonline.org/standards/sfg/imt2K	IMT2000 standards and other resources from Telecommunications Industry Association
www.uwcc.org	Web site of Universal Wireless Communications Consortium
www.wapforum.org	Technology and standards of WAP
http://winweb.rutgers.edu/pub/	Information about Wireless Information Network Laboratory, which participates in the "precompetitive stage of technology creation" for wireless Internet. Offers information on workshops, seminars, online seminars and news of wireless Internet

General technology sites

URL	Description
www.acm.org	Web site of Association for Computing Machinery

www.ieee.org	Web site of Institute of Electrical and Electronics Engineers (IEEE)
www.techonline.com	Education portal with online and off-line lectures and virtual laboratories

Books

J. Korhonen, *Introduction to 3G Mobile Communications*. Artech House, 2001.

T. Ojanpera, R. Prasad (Ed), *WCDMA: Towards IP Mobility and Mobile Internet*. Artech House, 2001.

Index

Numbers & Symbols

- 1G wireless networks, 328
 - 2.5G wireless networks, 1
 - 2G wireless networks, 1
 - CDMA 95B, 341
 - GPRS, 340-341
 - GSM wireless systems
 - architecture, 330-334
 - AuC (Authentication Center), 333
 - base station controller, 332
 - base station subsystems, 332
 - base transceiver, subsystems, 332
 - EIR, 334
 - features, 329
 - Gateway MSC, 334
 - HLR, 333
 - mobile station, 331
 - MSC, 334
 - MXE, 334
 - network areas, 334
 - OMC, 334
 - overview, 328
 - PLMN, 331
 - principles of operation, 335
 - services, 330
 - specifications, 329-330
 - TDMA format, 333
 - VLR, 334
 - HSCSD, 339-340
 - Internet access
 - difficulties of, 336
 - i-mode, 337-338
 - overview, 335-336
 - WAP limitations, 336-337
 - Internet content, 339
 - overview, 328-339
 - wireless devices, 338-339
 - 3G application development, 349
 - languages, 350-352
 - 3G partnership program Web site, 510
 - 3G programming
 - ASP, location-based services, 360-363
 - Java and, 352
 - WML and, 349-350
 - animation example, 352-355
 - XHTML and, 349-352
 - animation example, 355-356
 - audio files example, 356-358
 - location-based services, 360-363
 - video files example, 358-359
 - XML and, 350-351
 - animation display, XSL and, 364-366
 - XSL
 - audio files, 367-369
 - video files, 369-370
 - 3G wireless networks, 1
 - applications, 345-346
 - CDMA2000, 343
 - content development, 345
 - example architecture, 346-347
 - global roaming, problems with, 343
 - Internet access, requirements for, 344
 - location-based services and, 346
 - overview, 341-342
 - W-CDMA, 343
 - Web sites, 510-511
 - wireless devices, requirements for, 343-344
 - 4G systems, 472
 - 802.11 wireless LAN, Bluetooth and, 132
- ## A
- access code, link controllers (Bluetooth), 138
 - access networks, 461
 - ACL (Asynchronous Connection Less), 130
 - ACL links, link controllers (Bluetooth), 138
 - ACM (Association for Computing Machinery Web site, 510
 - ACTS (Advanced Communications Technologies and Services) Web site, 510
 - addPBEntry function, WTA, WML Script file, 59
 - addresses
 - Bluetooth, 139-140
 - WAP with Bluetooth, 153-154
 - advertisements, 89
 - 3G wireless networks and, 345
 - BREW application, 399-402, 407
 - airport kiosk application, 154-155
 - creating the DSN, 156
 - testing, 156-157
 - AMPS (Advanced Mobile Phone System), 328
 - analog cellular systems, disadvantages of, 328
 - animation

- BREW application, 384-391
- GIF format, 355
- WML and, 350
- WML, 3G programming, 352-355
- XHTML, 3G programming, 355-356
- XML and, XSL, 364-366
- anywhereyougo.com Web site, 508
- API (Application Programming Interface)
 - Bluetooth, 147
 - JMF and, 423-424
- applets, Java
 - BREW, 376
 - downloading, 3
- application development, 3G programming, 349
 - languages, 350-352
- applications
 - 3G wireless networks, 345-346
- animation
 - BREW, 384-
 - WML 3G programming, 352-355
 - XHTML 3G, programming, 355-356
- audio, XHTML 3G programming, 356-358
- BREW, 377-383
 - downloading music, 393-394
- chat (Bluetooth), client, 271-323
- database, BREW, 409, 417-418
- deploying to Tomcat Web server, 488
- file transfer (Bluetooth), 212-271
- mobile advertisements, BREW, 399-402, 407
- mobile advertising, wireless toolkit, 370-375
- push technology, 85-86
 - airport kiosk example, 154-157
 - development, 95
 - shopping mall kiosk example, 158-162
 - shopping cart with advertisement, 107-113, 120-124
 - stock quotes, 96-101, 105-106
- Quiz, Cold Fusion, 29-47
- Restaurant Application, WML Script, 9, 15-24
- SMS, 88
- using WAP with Bluetooth, 148-150
- video, XHTML 3G programming, 358-359
- Web, Tomcat Web server and, 486
- WTA, 49-50
- architecture, WTA, 50-51
- arithmetic operators, WML Script, 7
- ASP (Active Server Pages)
 - 3G programming, location-based services, 360-363
 - airport application, 154
 - shop.asp (shopping mall kiosk application), 158

- shopnew.asp (shopping mall kiosk application), 160
- asynchronous channels, voice communication and, 131
- attributes
 - PAP, 90
 - Service Indication, 93
 - XHTML, 351
- AuC (Authentication Center), GSM wireless systems and, 333
- audio
 - WML, 350
 - XHTML, 3G programming, 356-358
 - XSL, 367-369
- audio broadcasting application, 434-440
- audio-video broadcasting application, 446-458
- authentication
 - LMP, 140
 - WTA server, 51
- automobile uses for Bluetooth devices, 129
- automobiles
 - navigation services, 3G wireless networks and, 346

B

- base station controller, GSM wireless systems, 332
- base station subsystems, GSM wireless systems, 332
- base transceiver subsystems, GSM wireless systems, 332
- Baseband/Link Manager Protocol. *See* LMP
- batteries, Bluetooth devices, 127
- bearers, 148
- BID files (BREW), 377-379
 - animation application, 384
- binary encoding, PPG, 94
- bitmap resources, BREW, 379
- bitmaps, WBMP, 350
- bitwise operators, WML Script, 8
- blanket permission, WTA, 52
- Bluetooth, 127, 148
 - 802.11 wireless LAN and, 132
 - addressing, 139-140
 - APIs, 147
 - car uses, 129
 - chat application, 271
 - chat application client module, 272-295
 - chat application server module, 295-323
 - classes, 130
 - classes, declaring (HCI programming), 181-183
 - classes, declaring (SDP programming), 194-204
 - COM components, declaring (HCI programming), 183-194
 - command to discover nearby devices, 177

- commands and messages (HCI programming), 166-181
- commercial solutions, 132-134
- data rates, 131
- data services, 134
- declaring constants, classes, and methods (HCI programming), 164-166
- defining constants (HCI programming), 181-183
- development kit, 163
- devices that can be enabled, 128
- e-mail
 - browsing with mobile phone, 148
 - sending through mobile phone and laptop computer setup, 148
- event handlers, 176
- event handlers (SDP programming), 203, 239-270
- file transfer application, 212
- file transfer application client module, 216-245
- file transfer application common module, 213-216
- file transfer application server module, 245-271
- HCI programming overview, 163-164
- home uses, 128
- Host Control Interface, 153
- importing files (SDP programming), 194-204
- IrDA comparison, 131
- kiosks and, 149
- link controllers
 - data connections, 138-139
 - device states, 137-138
 - time slots, 136-137
 - voice connections, 138-139
- LMP, 140-142
- office uses, 128
- operating frequency, 130
- operating range, 130
- PANs and, 127
- Personal Operating Space and, 129
- piconets and, 134
 - client initiation, 149-150
 - server initiation, 150
- profiles, 133
- radio transmission, 127
- radio transmission hardware, 136
- resources, Web sites, 508-509
- RS232 variables, declaring (SDP programming), 208-212
- SDP programming overview, 194
- security, 135
- services, 130

- SIG, 127
- system architecture, 135-146
- TCS, 144
- variables, declaring (SDP programming), 204-208
- voice services, 134
- WAP
 - addressing, 153-154
 - client/server protocol stack, 150-153
 - implementation, 153
 - interoperability, 153
 - typical applications, 148-150
- Bluetooth Application Tool Kit (Ericsson) Web site, 508
- Bluetooth protocol stack (opensource) Web site, 508
- BREW (Binary Runtime Environment for Wireless), 3, 376
 - advertisement application, 399-407
 - animation application, 384-391
 - applets, 376
 - application development, 377-383
 - BID files, 377-379
 - bitmap resources, 379
 - database application, 409, 417-418
 - dialog controls, 379
 - dialog resources, 381
 - downloading music, 393-394
 - GUI, 376
 - MIF files, 377-379
 - overview, 376, 377
 - Resource Editor, 379-382
 - animation application, 386-387
 - string controls, 379
- BREW Emulator, 376
- BREW SDK, 376

C

- call forwarding, 466
- call log model, WTA, 53
- Call.java listing, 72-76
- callback binding, WTA, 60
- car uses for Bluetooth devices, 129
- CDMA, 3
- CDMA 95B, 341
- CDMA articles and resources Web site, 510
- CDMA development group Web site, 510
- cdma2000 systems, 1
- CDMA2000 wireless network, 343
- cell phones. *See* mobile phones
- cell splitting (mobile communications design issues), 327

- cellular mobile communications, 325-328
 - cellular networks, early years, 1
 - CFCCONTENT tag, 28
 - cfm file extension, 28
 - CFML (Cold Fusion Markup Language), 26
 - CFOUTPUT tag, 28-29
 - CGI (Common Gateway Interface), 477
 - channel spacing, GSM, 329
 - channels, 53
 - frequency allocation and reuse, 327
 - multi-cell wireless networks, 326
 - single cell wireless networks, 325
 - chat application (Bluetooth), 271
 - client module, 272-295
 - server module, 295- 323
 - classes, declaring
 - HCI programming, 164-166, 181-183
 - SDP programming, 194-204
 - ClassID, BID files, 377
 - client capability negotiation, PAP, 91
 - client/server, protocol stack, 150-153
 - clients
 - sessions, creating, 91
 - WAP/Bluetooth piconets
 - initiation, 149-150
 - clock offset request, LMP, 140
 - clusters, frequency allocation and reuse within, 327
 - Cold Fusion, 26
 - content type specification, 28
 - overview, 26-29
 - Quiz application, 29-47
 - tags, 26, 28
 - WAP browser text display, 28-29
 - Cold Fusion Server, 26
 - Cold Fusion Studio, 26
 - COM (Component Object Model)
 - components, declaring (HCI programming), 183-194
 - command button functions, HCI programming, 176
 - commands
 - HCI, 166-181
 - Tomcat Web server, startup and shutdown, 482
 - WML, 5-7
 - WML Script, 7-8
 - commercial uses of Bluetooth, 132-134
 - communications, serial, RFCOMM, 143-144
 - compatibility, platforms (SQL Server 2000), 491
 - configuration
 - engine, WML servlets, 95
 - handsets, SMS, 88
 - SMS-C, 88
 - confirmed data push, WSP, 91
 - connectionless push, 91
 - connections
 - Bluetooth module to PC, 163
 - LMP, 141
 - constants
 - declaring (HCI), 164-166
 - defining (HCI programming), 181-183
 - content development tools, 468
 - content transmission, PPG, 94
 - content type, Cold Fusion, specifying WAP, 28
 - context permission, WTA, 52
 - control structures, WML Script, 8
 - convergence, 460-462
 - of networks, 461, 462
 - of services, 462-464
 - convergence technologies, 460-464
 - converting HTML to WML, 3, 350
 - converting text to speech, 464
 - cordless phones, Bluetooth, 133
 - Cordless telephony profile, Bluetooth, 133
 - cstack.com Web site, 508
 - CTI (computer technology integration), 465
 - CVSD (Continuously Variable Slope Delta Modulation), 135
- ## D
- data connections, link controllers (Bluetooth), 138-139
 - data rates, Bluetooth, 131
 - data services, Bluetooth, 130-134
 - databases
 - BREW application, 409, 417-418
 - GPS, 360-363
 - location-based systems, 360-363
 - stock quote push technology application, 98
 - DDX Control functions
 - HCI programming, 175
 - SDP programming, 239
 - declarations
 - classes (SDP programming), 194-204
 - COM components (HCI programming), 183-194
 - DTD, 351
 - HCI programming, 164-166
 - RS232 variables (SDP programming), 208-212
 - variables (SDP programming), 204-208
 - deployment descriptor file, 487
 - design issues, cellular systems, 327
 - cell splitting, 327
 - frequency allocation and reuse, 327
 - geographical cells and radio surveys, 327

- PSTN trunk capacity, 327
- shadow regions, 327
- traffic analysis, 328
- desktop PCs, Bluetooth enabled, 133
- destroy() method, Java servlets, 65
- developer.axis.com Web site, 508
- development, content development tools, 468
- development kit (Bluetooth), 163
- dialog controls, BREW, 379
- dialog resources, BREW, 381
- Dialogs library, WML Script, 7-9
- Dial-up networking profile, Bluetooth, 133
- digital mobile communication networks. *See* 2G wireless networks
- directories
 - application directory structure, 486-488
 - IIS Virtual Directory Management, 502-507
 - structure, Tomcat Web server, 486
- DNS (Domain Name Server), 84
- downloading
 - applets, Java, 3
 - BREW SDK, 376
 - music, BREW application, 393-394
 - Tomcat Web server, 478
- DSN (Data Source Name), creation, 363
- DTD (Document Type Definition), 5, 90
- DTD declarations, 351
- duplex distance, GSM, 329
- dynamic content technologies, WWW, 477
- dynamic WML content
 - JSP, 68
 - servlets (Java), 67-68

E

- EIR (Equipment Identity Register)
 - GSM wireless systems, 334
- electronic business cards, 3G wireless networks and, 345
- electronic wallets, 3G wireless networks and, 345
- e-mail, 466
 - 3G wireless networks and, 345
 - browsing with mobile phone, 148
 - handset to PC, 88
 - PCs to mobile, 88
 - sending through mobile phone and laptop computer setup, 148
- emerging technologies
 - CTI, 465
 - speech recognition, 464
 - text-to-speech conversion, 464
- encryption, LMP, 140

- engine configuration, WML servlets, 95
- entertainment services, 3G wireless networks and, 345
- environment variables, Tomcat Web server, 482
- Ericsson Web site, 508
- Ericsson's PC reference stack (Bluetooth development kit), 163
- error messages, Tomcat Web server, 485
- European systems, 1
- event binding, WTA, 56
 - callback binding, 60
 - global, 60
 - temporary, 60
- event handlers
 - Bluetooth (SDP programming), 203
 - HCI programming, 176
 - SDP programming, 239-241, 268-270
- event management, WTA, 60
- Event model, WTA, 54-56
- events
 - HCI application module, 164
 - SDP application module, 194
 - WML commands, 6

F

- FA (Foreign Agent), 472
- fax mail, 466
- Fax profile, Bluetooth, 133
- file extensions, cfm, 28
- file transfer, HomeRF, 131
- file transfer application (Bluetooth), 212
 - client module, 216-245
 - common module, 213-216
 - server module, 245-271
- File transfer profile, Bluetooth, 133
- files, importing (SDP programming), 194-204
- first generation wireless networks. *See* 1G wireless networks
- fixed devices' obsolescence, 4
- flight.asp code (airport kiosk application, 154-155)
- Float library, WML Script, 7-8
- for statements, WML Script, 8
- formatting commands, WML, 6
- forms, WML commands, 6
- frequency allocation and reuse, mobile communications design issues, 327
- frequency band, GSM, 329
- frequency of operation, HomeRF, 132
- frequency of operation, Bluetooth, 130
- FSK (Frequency Shift Keying), 329
- functions
 - addPBEntry, WTA WML Script file, 59

- makeCall, WTA WML Script file, 58
- Network common WTAI library, 56
- sendDTMF, WTA WML Script file, 58
- WML Script, 8
- WTAI functin call example, 60-62

G

- Gateway MSC, GSM wireless systems, 334
- Generic access profile, Bluetooth, 133
- geographical cells
 - cell splitting, 327
 - mobile communications design issues, 327
 - shadow regions, 327
- GFSK (Gaussian Frequency Shift Keying), 136
- global binding, WTA, 60
- global roaming, problems with, 343
- Global System for Mobile Communication. *See* GSM.
- GMSC (Gateway MSC), SMS, 87
- GPRS (General Packet Radio Service), 1, 340-341
- GPS, 360-363
- GPS (Global Positioning System), 360
 - Bluetooth and, 129
- graphics
 - WBMP, 2
 - WML, 350
- GSM (Global System for Mobile Communication)
 - wireless systems, 328
 - architecture, 330-334
 - AuC (Authentication Center), 333
 - base station controller, 332
 - base station subsystems, 332
 - base transceiver subsystems, 332
 - EIR, 334
 - features, 329
 - Gateway MSC, 334
 - HLR, 333
 - mobile station, 331
 - MSC (Mobile Switching Center), 334
 - MXE (Message Center), 334
 - network areas, 334
 - OMC, 334
 - overview, 328
 - PLMN, 331
 - principles of operation, 335
 - services, 330
 - specifications, 329-330
 - TDMA format, 333
 - VLR, 334
- GUI (graphical user interface)
 - BREW, 376
 - HCI application module, 164
 - SDP application module, 194

H

- H.323 standards, voice/video, 422-423
- HA (Home Agent), 472
- handsets
 - e-mail to PC, 88
 - SI messages, 92
 - SMS configuration, 88
 - SMS messages, 86
- hardware
 - radio transmission, Bluetooth, 136
 - SQL Server 2000 system requirements, 491
- HCI (Host Controller Interface), 145-146
 - Bluetooth event handlers, 176
 - classes, declaring, 181-183
 - COM components, declaring, 183-194
 - commands and messages, 166-181
 - declaring constants, classes, and methods, 164-166
 - defining constants, 181-183
 - events module, 164
 - GUI module, 164
 - programming overview, 163-164
 - remote device module, 164
- Headset profile, Bluetooth, 133
- headset, Bluetooth, 133
- hidden computing scenario, 148
- high-speed data services, 2.5G networks and, 1
- HLR (Home Location Register)
 - GSM wireless systems, 333
 - SMS, 87
- hold mode, LMP, 140
- hold() function, network specific WTAI library, 57
- home uses for Bluetooth devices, 128
- HomeRF, 131-132
- hop frequency, masters and, 131
- horoscopes, 85
- Host Control Interface (Bluetooth), 153
- Host Controller Interface. *See* HCI
- HSCSD (High Speed Curcuit Switched Data), 339-340
- HTML (HyperText Markup Language), 2
 - converting to WML, 3
 - Web servers, handling requests, 477
 - WML conversion, 350
 - XHTML comparison, 351

I

IE (Internet Explorer), 2
 IEEE (Institute of Electrical and Electronics Engineers)
 standards, 131
 Web site, 511
 if-else statements, WML Script, 8
 IIS (Internet Information Server)
 Virtual Directory Management, 502-507
 images
 inserting in WML commands, 6
 sun.ico file, Information Master, 15
 i-mode, Internet access, 337-338
 I-Mode, 2
 importing, files (SDP programming), 194-204
 IMT2000 standards and resources Web site, 510
 Information Master, WML Script, 9-15
 Information.wml file, Information Master, 9-11
 Infrared Data Association Web site, 508
 initialization, Java servlets, 65
 installation
 SQL Server 2000, 493-501
 Tomcat Web server, 478-485
 instant messaging, 465
 interactive voice response systems, 466
 Intercom profile, Bluetooth, 133
 interfaces, WTA libraries, 52
 Interim Standard (IS) 54/136 based wireless systems, 328
 International Validation and Testing Corporation
 Web site, 508
 Internet, accessing
 2G wireless networks, 335-336
 3G wireless networks, requirements for, 344
 difficulties of, 336
 i-mode, 337-338
 WAP limitations, 336-337
 Internet, content
 2G wireless networks, 339
 pull technology and, 84
 push technology and, 84
 Internet Protocol. *See* IP
 interoperability, WAP with Bluetooth, 153
 Inter-planetary Internet Web site, 510
 intrusion levels, push messages, 93
 IP (Internet Protocol), 2
 client/server setups, 151
 Mobile IP, 472
 IP networks
 video, 420

 voice communications, 420
 IP version 6, 471, 472
 IR (infrared) communication, 131
 SIR (Serial IR), 131
 IrDA (InfraRed Data Association) standards, 131
 Bluetooth comparison, 131
 IS (Interim Standard), 1
 IS 136 standards, 1
 IS 95A based wireless systems, 328
 IS 95A standards, 1
 IS 95B systems, 1
 ISDN (Integrated Services Digital Network)
 H.323 standards, 422
 ISP (Internet Service Provider), 460
 ITU (International Telecommunications Union),
 422-423
 ITU-R (ITU Radio Communications Standardization
 Sector Task Group 8/1), 342
 IVR (Interactive Voice Response), 465

J

J2ME (Java 2 Micro Edition), 3, 344, 352
 Java, 3
 3G programming and, 352
 servlets, 65
 Java technologies, 63
 JavaScript, WMLScript and, 2
 javax.servlet, 65
 javax.servlet.http, 65
 JMF (Java Media Framework)
 voice/video, 423-424
 JSP (Java Server Pages), 66
 dynamic WML content, 68
 servlets, compiling to, 66
 WAP application, 68-83
 JVM (Java Virtual Machine), 3, 352
 H.323 standards, 422

K

kiosks, WAP and Bluetooth enabled, 149
 KVM (Kilobyte Virtual Machine), 3, 352
 Java code, downloading, 3

L

L2CAP (Logical Link Control and Adaptation Protocol), 142
 client/server setups, 150
 Location interoperability forum Web site, 510
 LAN access profile, Bluetooth, 133
 Lang library, WML Script, 7-8

- languages, 3G application development, 350
 - Java, 352
 - WML, 350
 - XHTML, 351-352
 - XML, 350-351
- LAN access point, Bluetooth, 133
- laptop computers/mobile phones, communicating
 - between, 148
- last inch, wireless communications, 462
- last mile, wireless communications, 461
- LCD projector, Bluetooth, 133
- lesswire Web site, 508
- levels of intrusion, push messages, 93
- libraries
 - function libraries (WTAI), 56-59
 - interfaces, WTA, 52
 - WML Script, 7-9
 - WTAGSM, 58
- link controllers, Bluetooth
 - device states, 137-138
 - time slot, 136-137
 - voice/data connections, 138-139
- links, supervision (LMP), 140
- listings
 - Accessing emergency services through WTAI (WML code), 61
 - action.cfm, Quiz application (Cold Fusion), 34-35
 - Advertise.c (BREW), 402-407
 - Animation through XHTML, 356
 - Animation through XSL, 364-365
 - Animation.c (BREW), 387-391
 - answer.cfm, Quiz application (Cold Fusion), 44
 - appl.wmls WML Script file (WTA), 59
 - ASP code (flight.asp) for Airport Kiosk, 154-155
 - ASP code (shop.asp) for Shopping Mall Kiosk, 158-159
 - LMASP code (shopnew.asp) for Shopping Mall Kiosk, 160
 - Audio through XSL, 367-368
 - AudioCapture.java, voice messaging, 427-432
 - AudioReceiveStreams.java, audio broadcasting application, 440-445
 - AudioVideoReceive.java, 452, 457-458
 - AudioVideoTransmit.java, 446-451
 - bingo.cfm, Quiz application (Cold Fusion), 45
 - Call.java, 72-76
 - checkvalue.cfm, Quiz application (Cold Fusion), 38-39
 - CommandDlg.cpp, 196-202
 - CommandsDlg.h, 194-196
 - ConnectionInfo.cpp (file transfer application), 215
 - ConnectionInfo.h (file transfer application), 214-215
 - Database application code (BREW), 409, 417-418
 - Events.cpp, 184-187
 - Events.h, 183
 - HCI Information CommandsDlg.cpp, 166, 175
 - HCI Information CommandsDlg.h, 164-166
 - HTML registration code
 - shopping cart application, push technology, 109
 - stock quote application, push technology, 96
 - Index.cfm, Quiz application (Cold Fusion), 32, 33
 - Information.wml file (Information Master), 10-11
 - Java program, preferences storage in database (shopping cart), 111
 - Java program, preferences storage in database (stock quote), 98
 - Java program, saving order in database (shopping cart), 121
 - Java program, validating and displaying on browser (shopping cart), 113
 - login.cfm, Quiz application, Cold Fusion, 36-37
 - MIDlet for mobile advertising, 371-375
 - Movie.wml file (Information Master), 11-12
 - PrintProfile.cpp, 205-207
 - PrintProfile.h, 204-205
 - Professor.java, 451-452
 - questiondisplay.cfm, Quiz application (Cold Fusion), 40-42
 - RadioChatClientDlg.cpp (chat application), 274, 294
 - RadioChatClientDlg.h (chat application), 272, 274
 - RadioFileClientDlg.cpp (file transfer application), 219, 239
 - RadioFileClientDlg.h (file transfer application), 216-218
 - RadioFileServerDlg.cpp (file transfer application), 248, 267
 - RadioFileServerDlg.h (file transfer application), 245-248
 - readallvalue.cfm, Quiz application (Cold Fusion), 39-40
 - RemoteDevice.cpp, 182
 - RemoteDevice.h, 181
 - Report.java, 77-78
 - ResScripts.wmls file (Restaurant App), 23
 - Restaurant.wml file (Restaurant App), 16-17
 - RS232.cpp, 209-210
 - RS232.h, 208-209
 - Service Indication generated by the Tool Kit for Airport Kiosk Push Message, 157
 - Service Indication generated by Tool Kit for Shopping Mall Push Message, 161
 - Service.cpp (file transfer application), 213-214
 - Service.h (file transfer application), 213

- Servlet to push stock quote information, 101
- Snacks.wml file (Restaurant App), 21-22
- Soft.wml file (Restaurant App), 20-21
- Solution.java, 79-81
- Sound.c (BREW), 394-398
- Source Code for RadioChatServerDlg.cpp (chat application), 298-323
- Source Code for RadioChatServerDlg.h (chat application), 295-298
- South.wml file (Restaurant App), 18-19
- Student.java, audio broadcasting application, 445-446
- submit.cfm, Quiz application (Cold Fusion), 42-43
- TestWML.Java, 69-70
- TrialJsp.JSP, 68
- TrialServlet.java, 67-68
- tryagain.cfm, Quiz application (Cold Fusion), 46-47
- UserInter.java, voice messaging, 432-434
- video play, XSL, 369-370
- Weather.wml file (Information Master), 13-14
- Weather.wmls file (Information Master), 14-15
- web.xml, 488
- Welcome.c (BREW), 382-383
- WML code for animation using timer, 352-354
- WML code for calling servlet/placing order (shopping cart), 123-124
- XHTML code for displaying retrieved values from database, 361-363
- XHTML code for latitude/longitude input, 360-361
- XHTML code to play audio file, 357
- XHTML code to play video files, 358-359
- XML code for animation, 366
- XML code for audio, 368-369
- XML code for video, 370
- P (Baseband/Link Manager Protocol)
 - client/server setups, 150
- LMP (Link Manager Protocol), 140-142
- location, precise location-based services, 467
- location() function, network specific WTAI library, 57
- location-based services
 - 3G programming
 - ASP, 360-363
 - XHTML, 360-363
 - 3G wireless networks and, 346
- logical indicator model, WTA, 53
- Logical Link Control and Adaptation Protocol. *See* L2CAP
- logical operators, WML Script, 7-8
- low bit rate coding, voice/video, 421
- low-speed networks, WAP and, 2
- Lucent Technologies Web site, 508

M

- m advertising, 86
- mail notification, push technology, 85
- makeCall function, WTA, WML Script file, 58
- masters
 - hop frequency, 131
 - PAN, 131
 - piconet, 134
 - switching roles, LMP, 140
- m-commerce, 3G wireless networks and, 345
- MD (Mobile Device), 472
- Mecel Web site, 508
- medical services, 3G wireless networks and, 346
- messages
 - HCI, 166-181
 - length, SMS and, 89
 - PI and, 89
 - push, processing, 89-90
 - voice messaging application, 424-434
- messaging
 - instant messaging, 465
 - unified messaging, 465-466
 - call forwarding, 466
 - e-mail, 466
 - fax mail, 466
 - interactive voice response system, 466
 - short messaging service, 466
 - video messaging, 467
 - voice dialing, 466
 - voice messaging, 466
- meta languages, 3
- methods, declaring (HCI), 164-166
- MIDlets, 352, 370
- MIDs (Mobile Information Devices), 352, 370
- MIF files (BREW), 377-379
- MIP (Mobile IP), 2
- mobile advertising application, 370-375
- mobile banking, SMS and, 88
- mobile communications
 - cellular design issues, 327-328
 - multi-cell wireless networks, principles of operation, 326-327
 - principles of operation, 325-326
- mobile devices, 467
 - GPS, 360
- Mobile IP, 472
- mobile phones/laptop computers, communicating between, 148
- mobile stations, GSM wireless systems, 331
- mobile-originated SMS, 86
- mobile-terminated SMS, 87

- modulation, GMSK, 329
- Motorola Web site, 508
- Movie.wml file, Information Master, 9
 - code output, 12
 - listing, 11, 12
- MPEG (Moving Picture Experts Group) files, 358
- MSC (Mobile Switching Center)
 - GSM wireless systems, 334
 - mobile-originated SMS and, 86
 - mobile-terminated SMS and, 87
- multi-cell wireless networks, principles of operation, 326-327
- music, downloading (BREW application), 393-394
- MXE (Message Center)
 - GSM wireless systems, 334

N

- name request, LMP, 140
- namespaces, XML references, 351
- navigation, WML commands, 6
- navigation services for automobiles, 3G wireless networks and, 346
- nested tags
 - XHTML, 351
 - XML, 351
- network common WTAI library, 52
- Network common WTAI library
 - functions, 56
- network message model, WTA, 53
- network specific WTAI library, 52, 57-58
- networks
 - access networks, 461
 - BREW and, 376
 - convergence of, 461-462
 - piconets, 131
- news services, 3G wireless networks and, 345
- NN (Netscape Navigator), 2
- nodes, PANs, 127
- Nokia Web site, 508
- non-confirmed data push, WSP, 91
- Nordic Mobile Telephony (NMT), 328, 450, 900
- North American systems, 1
- notebook PCs, Bluetooth enabled, 133
- NTTDoCoMo Web site, 510

O

- Object push profile, Bluetooth, 133
- ODBC connectivity, 363
- office uses for Bluetooth devices, 128
- OMC (Operation and Maintenance Center)

- GSM wireless systems, 334
- opening/closing tags
 - XHTML, 351
 - XML, 351
- operating frequency
 - Bluetooth, 130
 - HomeRF, 132
- operating range, Bluetooth, 130
- operating systems
 - 3G wireless networks, requirements for, 344
 - SQL Server 2000, system requirements, 492-493
- operators, WML Script, 7, 8
- Originating Call model, WTA Event model, 54
- OTA (Over The Air) protocol, push technology, 89-91
- overloading SMS, 89

P

- palowireless.com Web site, 508
- PANs (Personal Area Networks)
 - Bluetooth, 127
 - masters, 131
 - overview, 127
 - slaves, 131
 - topology, 131
- PAP (Push Access Protocol), 89-91
- park mode, LMP, 140
- parsing, XML, 351
- PC reference stack (Bluetooth development kit), 163
- PCM (Pulse Code Modulation), 135, 421
- PCs
 - e-mail to wireless, 88
 - SMS and, 86, 88
- performance, 3G wireless network requirements, 344
- peripherals, 3G wireless network requirements, 344
- permissions
 - policy files, 425
 - WTA, 51
- Personal Digital Cellular (PDC) wireless system, 328
- Personal operating space, Bluetooth and, 129
- Philips Semiconductors Web site, 508
- phone book model, WTA, 53
- PI (Push Initiator), 89
- piconets, 131, 134
 - WAP and Bluetooth enabled, 149-150
- platform compatibility, SQL Server 2000, 491
- PLMN (Public Land Mobile Network), 331, 460
- Point to Point Protocol. *See* PPP.
- policy files, 425
- ports, Tomcat Web server, 485
- power
 - Bluetooth devices, 127

- LMP, 140
 - power classes, radio transmission, 136
 - power consumption, 3G wireless network
 - requirements, 344
 - power of transmission, HomeRF, 132
 - PPG (Push Proxy Gateway), 86-95
 - PPP (Point to Point Protocol), client/server setups, 151
 - PQR, banking, 88
 - precise location-based services, 467
 - PRINTSERVICE, SDP application module, 194
 - profiles, Bluetooth, 133
 - programming
 - classes, declaring (HCI), 181-183
 - COM components, declaring (HCI), 183-194
 - commands and messages (HCI), 166-181
 - declaring constants, classes, and methods (HCI), 164-166
 - defining constants (HCI), 181-183
 - HCI overview, 163-164
 - protocol conversion, PPG, 94
 - protocols, 470
 - client/server stack, 150-153
 - H.232 standards, 422
 - IP, 151
 - IP version 6, 471-472
 - L2CAP, 150
 - LMP, 150
 - multiplexing, L2CAP, 142
 - OTA, 91
 - PPP, 151
 - push messages, 89
 - RFCOMM, 151
 - RTCP, 421
 - RTP, 2, 421
 - SDP, 149-150
 - SMS, 152
 - TCP, 2
 - UDP, 2, 151
 - video, 421
 - voice communications, 421
 - WAP bearers, 148
 - WDP, 151
 - WSP, 152
 - WTLS, 151
 - WTP, 152
 - PSTN (Public Switched Telephone Network), 460
 - mobile communications design issues, 327
 - principles of operation, 335
 - Public Land Mobile Network. *See* PLMN.
 - Public WTAI, 58
 - public WTAI library, 52
 - pull technology, Internet content access and, 84
 - push applications
 - airport kiosk example, 154-155
 - creating the DSN, 156
 - testing, 156-157
 - shopping mall kiosk example, 158-160
 - creating the DSN, 160
 - testing, 160-162
 - push cancellation, PAP, 91
 - push framework, 89-94
 - pros/cons, 125
 - push initiation messages, PPG, 94
 - Push Message Simulator, 156
 - push messages
 - levels of intrusion, 93
 - PPG and, 89
 - protocols, 89
 - push initiation messages, 94
 - SI, 92
 - push submission, PAP, 90
 - push technology, 84
 - application, development, 95
 - applications for, 85-86
 - binary encoding, 94
 - confirmed data push, 91
 - connectionless push, 91
 - content transmission, 94
 - Internet content access and, 84
 - message processing, 89-90
 - non-confirmed data push, 91
 - OTA protocol, 89-91
 - overview, 84-85
 - PAP, 90-91
 - PAP protocol, 89
 - protocol conversion, 94
 - service indication, 85
 - Service Loading, 92-94
 - shopping cart with advertisement, 107-113, 120-124
 - SIA, 91
 - stock quote application, 96-98, 101, 105-106
 - WDP, 91
 - WSP, 91
- ## Q
- QoS
 - L2CAP, 142
 - parameters exchange, LMP, 140
 - Question of the Day WAP page, 68
 - Quiz application, Cold Fusion, 29-30
 - action.cfm listing, 34-35
 - answer.cfm listing, 44
 - bingo.cfm listing, 45

- checkvalue.cfm listing, 38-39
- function, 31-47
- Index.cfm listing, 32-33
- login.cfm listing, 36-37
- questiondisplay.cfm listing, 40-42
- readallvalue.cfm listing, 39-40
- submit.cfm listing, 42-43
- tryagain.cfm listing, 46-47
- quotation marks, XHTML, 351

R

- radio surveys, mobile communications design
 - issues, 327
- radio transmission
 - Bluetooth, 127
 - hardware, 136
 - HomeRF, 131
 - range of operation and, 130
- range of operation
 - Bluetooth, 130
 - HomeRF, 132
- registration, services (SDP programming), 194-212
- reject() function, network specific WTAI library, 57
- relational operators, WML Script, 7-8
- remote devices
 - HCI application module, 164
 - SDP application module, 194
- Report.java listing, 77, 78
- request and response features of LMP, 140
- resolution, graphics (WBMP), 350
- Resource Editor (BREW), 379-382
 - advertisement application, 401-402
 - animation application, 386-387
- resources, 53
- Restaurant Application, WML Script, 9, 15-24
- result notification, PAP, 90
- RF standards and products Web site, 508
- RFCOMM protocol, 143-144
 - client/server setups, 151
- roaming globally, problems with, 343
- root element, XML, 351
- RS232 variables
 - declaring (SDP programming), 208-212
 - SDP application module, 194
- RTCP (Real Time Control Protocol), 421
- RTP (Real Time Transport Protocol), 2, 421

S

- SCO (Synchronous Connection Oriented), 130
- SCO links

- link controllers, Bluetooth, 138
- TCS, 144
- scores, sports, 85
- scripting, WTAI interface, 56-59
- SDP (Service Discovery Protocol), 142-143
 - classes, declaring, 194-204
 - client/server setups, 150
 - DDX Control functions, 239
 - events module, 194
 - files, importing, 194-204
 - GUI module, 194
 - piconets and, 149
 - PRINTSERVICE module, 194
 - programming overview, 194
 - remote device module, 194
 - RS232 module, 194
 - RS232 variables, declaring, 208-212
 - variables, declaring, 204-208
- second generation wireless networks. *See* 2G wireless networks
- security
 - Bluetooth, 135
 - permissions, policy files, 425
 - WTA, 51-52
- segmentation, L2CAP, 142
- sendDTMF function, WTA, WML Script file, 58
- sendUSSD() function
 - network specific WTAI library, 57
- Serial communication profile, Bluetooth, 133
- serial communications, RFCOMM and, 143-144
- serial ports
 - defining parameters (HCI programming), 175
 - firing success/failure messages (HCI programming), 178
 - interface constants, defining (SDP programming), 239
- ServletRequest object, 65
- servers
 - Cold Fusion, 26
 - WAP/Bluetooth piconets initiation, 150
 - WTA, 51
- servers, 477. *See also* Web servers
- Service discovery application profile, Bluetooth, 133
- Service Discovery Protocol. *See* SDP
- service indication, push technology, 85
- service() method, Java servlets, 65
- services
 - convergence of, 462-464
 - registering (SDP programming), 194-212
- Servlet Response object, 65
- servlets
 - Java, 65-83

- WML content, 95
 - sessions, client creating, 91
 - shadow regions (mobile communications design issues), 327
 - shop.asp (shopping mall kiosk application), 158-159
 - shopnew.asp (shopping mall kiosk application), 160
 - shopping cart with advertisement application, push technology, 107-113, 120-124
 - shopping mall kiosk application, 158-160
 - creating the DSN, 160
 - testing, 160-162
 - Short Messaging Service. *See* SMS
 - shutdown command, Tomcat Web server, 482
 - SI (Service Indication) messages, 92
 - SIA (Service Initiation Application), 91
 - SIG (Bluetooth Special Interest Group) Web site, 508
 - Sigma ComTec AB Web site, 508
 - signaling, GSM, 329
 - Signaling System 7 (SS7), 330
 - SIGs (Special Interest Groups), Bluetooth, 127
 - single action permission, WTA, 52
 - SIR (Serial IR), 131
 - SL (Service Loading), 92, 94
 - slaves
 - PANs, 131
 - piconet, 134
 - switching roles, LMP, 140
 - SMS (Short Messaging Service), 3, 86-89, 152, 466
 - SMS-C (SMS Center)
 - configuration, 88
 - handset configuration, 88
 - mobile-originated SMS and, 86
 - mobile-terminated SMS and, 87
 - Snacks.wml file, Restaurant App, 15
 - code output, 23
 - listing, 21-22
 - Soft.wml file, Restaurant App, 15
 - code output, 21
 - listing, 20-21
 - Solution.java listing, 79-81
 - South.wml file, Restaurant App, 15
 - code output, 19
 - listing, 18-19
 - speakers, Bluetooth, 133
 - speech coding, GSM, 329
 - speech recognition, 464
 - sports scores, 85
 - SQL Server 2000
 - Developer Edition, 490
 - Enterprise Edition, 490
 - Enterprise Evaluation Edition, 491
 - installation, 493-501
 - overview, 490
 - Personal Edition, 490
 - platform compatibility, 491
 - Standard Edition, 490
 - system requirements, 491-493
 - Windows CE Edition, 490
 - XML support, 501-502
 - standards
 - IS 136, 1
 - IS 95A, 1
 - startup command, Tomcat Web server, 482
 - state management, WTA, 60
 - states, devices, 137-138
 - status codes, PAP, 90
 - status query, PAP, 91
 - stock quote application, push technology, 96-106
 - stock quotes, 85
 - storage, WTA, 53-54
 - store-and forward mechanism, SMS and, 89
 - string controls, BREW, 379
 - String library, WML Script, 7-8
 - sun.ico file, Information Master, 9
 - code output, 15
 - SWAP (Shared Wireless Access Protocol), 132
 - synchronization, 4
 - Synchronization profile, Bluetooth, 133
 - SyncML (Synchronization Markup Language), 470
 - syntax
 - WML, 5-7
 - WML Script, 7-8
 - XHTML, 351
 - system architecture, Bluetooth, 135-146
 - system requirements
 - IIS Virtual Directory Management, 502
 - SQL Server 2000
 - hardware, 491
 - operating systems, 492-493
- ## T
- tables, commands (WML), 6
 - tags
 - CFCCONTENT, 28
 - CFOUTPUT, 28-29
 - Cold Fusion, 26-28
 - WML, 5
 - XHTML, 351
 - XML, 351
 - TCP (Transmission Control Protocol), 2
 - TCP layer, voice/video, 421

TCS (Telephony Control Protocol Specification), 144
 TDMA (Time Division Multiple Access)
 GSM, 329, 333
 technoline.com Web site, 511
 Teleca Comtec Web site, 163
 Telelogic Web site, 508
 temporary binding, WTA, 60
 Terminating Call model, WTA Event model, 54
 testing
 airport kiosk example, application, 156-157
 shopping mall kiosk application, 160-162
 TestWML.Java listing, 69-70
 text, Cold Fusion (WAP browser), 28-29
 text messages, SMS, 86
 text-to-speech conversion, 464
 the3gportal.com Web site, 510
 time slots, link controllers (Bluetooth), 136-137
 timing accuracy information request, LMP, 140
 Tomcat Web server
 application directory, 486, 488
 applications, deploying, 488
 directory structure, 486
 installation, 478-485
 overview, 478
 startup and shutdown commands, 482
 Web applications, 486-487
 topology
 PAN, 131
 PANs, 127
 topsitelists.com Web site, 509
 Total Access Communication System (TACS), 328
 traffic analysis (mobile communications design issues), 328
 transfer() function, network specific WTAI library, 57
 transmission power, HomeRF, 132
 transmit data rate, GSM, 329
 TrialJsp.JSP, 68
 TrialServlet.java, 67-68
 trunk capacity (PSTN), mobile communications design issues, 327
 trusted services, 51

U

UDP (User Datagram Protocol), 2, 421
 client/server setups, 151
 unified messaging, 465-466
 call forwarding, 466
 e-mail, 466
 fax mail, 466
 interactive voice response systems, 466
 short messaging service, 466
 video messaging, 467
 voice dialing, 466
 voice messaging, 466
 Universal Wireless Communications Consortium
 Web site, 510
 URL library, WML Script, 7, 9
 URLs (Uniform Resource Locators)
 WAP, addressing with Bluetooth, 153-154
 USB (Universal Serial Bus), interface constants, defining (SDP programming), 239
 user agents, WTA, 51, 60
 User Datagram Protocol. *See* UDP
 user permissions, WTA, 51

V

variables
 declaring (SDP programming), 204-208
 environment, Installing Tomcat Web server, 482
 WML commands, 6
 video, 420
 H.323 standards, 422-423
 IP network applications, 420
 JMF, 423-424
 low bit-rate coding, 421
 PCM, 421
 protocols, 421
 WML, 350
 XHTML, 3G programming, 358-359
 XSL, 369-370
 video conferencing, 3G wireless networks and, 345
 video mail, 3G wireless networks and, 345
 video messaging, 467
 video players, 3G wireless networks and, 346
 videophones, 3G wireless networks and, 345
 Virtual Directory Management (IIS), 502-507
 VLR (Visitor Location Register)
 GSM wireless systems, 334
 voice call model, WTA, 53
 voice communication, 420
 asynchronous channels, 131
 audio broadcasting application, 434, 438-440
 audio-video broadcasting application, 446-458
 H.323 standards, 422-423
 IP network applications, 420
 JMF, 423-424
 low bit rate coding, 421
 PCM, 421
 protocols, 421
 voice connections, link controllers (Bluetooth), 138-139
 voice dialing, 464, 466

- voice messaging, 466
- voice messaging application, 424-434
- voice services (Bluetooth), 130-134
- Voice XML (Voice Extensible Markup Language), 468-470

W

- WAE (Wireless Application Environment), 5
- WAP (Wireless Application Protocol), 2
 - Bluetooth, 148-154
 - browser text display (ColdFusion), 28-29
 - Internet access, limitations, 336-337
 - JSP-based application, 68-83
 - kiosks and, 149
 - low-speed networks, 2
 - piconets and, 149-150
 - protocol bearers, 148
 - servlets-based application, 68-83
- WAP Forum, 2
- WAP gateway, WTA server and, 51
- WAP technology and standards Web site, 510
- WAP toolkits and discussion lists Web site, 510
- WAP-enabled phones, 3
- WBMP (Wireless Bitmap), 2
 - images, 350
- W-CDMA (Wideband Code Division Multiple Access) systems, 1
 - wireless network, 343
- WDP (Wireless Datagram Protocol), 91
 - client/server setups, 151
- Weather Report WAP page, 68
- Weather.wml file, Information Master, 9
 - code output, 14
 - listing, 13-14
- Weather.wmls file, Information Master, 9
 - code output, 15
 - listing, 14-15
- Web servers
 - overview, 477
 - PI, 89
 - principles of operation, 477
 - Tomcat, 478-486
- Web sites
 - 3G partnership program, 510
 - 3G wireless networks, 510-511
 - ACM, 510
 - ACTS, 510
 - Bluetooth Application Tool Kit (Ericsson), 508
 - Bluetooth protocol stack (opensource), 508
 - Bluetooth resources, 508-509
 - CDMA articles and resources, 510
 - CDMA development group, 510
 - cstack.com, 508
 - developer.axis.com, 508
 - Ericsson, 508
 - IEEE, 511
 - IMT2000 standards and resources, 510
 - Infrared Data Association, 508
 - International Validation and Testing Corporation, 508
 - Inter-planetary Internet, 510
 - Location interoperability forum, 510
 - lesswire, 508
 - Lucent Technologies, 508
 - Mecel, 508
 - Motorola, 508
 - Nokia, 508
 - NTTDoCoMo, 510
 - palowireless.com, 508
 - Philips Semiconductors, 508
 - RF standards and products, 508
 - SIGs, 508
 - Sigma ComTec AB, 508
 - technoline.com, 511
 - Teleca Comtec, 163
 - Telelogic, 508
 - the3gportal.com, 510
 - topsitelists.com, 509
 - Universal Wireless Communications Consortium, 510
 - WAP technology and standards, 510
 - WAP toolkits and discussion lists, 510
 - Wireless Information Network Laboratory, 510
 - wireless technologies information, 508
- Web-based learning, 3G wireless networks and, 345
- WECA (Wireless Ethernet Compatibility Association), 132
- while statements, WML Script, 8
- Windows NT/2000, Tomcat Web server, 479-482
- wireless applications, evolution of, 1
- wireless communications
 - last inch, 462
 - last mile, 461
- Wireless Datagram Protocol. *See* WDP
- wireless devices
 - 2G wireless networks and, 338-339
 - 3G wireless networks, requirements for, 343-344
- Wireless Information Network Laboratory Web site, 510
- wireless networks, evolution of, 1
- wireless protocols, evolution of, 1
- Wireless Session Protocol. *See* WSP
- wireless toolkit, 370-375
- Wireless Transaction Protocol. *See* WTP
- Wireless Transport Layer Security. *See* TLS

WML (Wireless Markup Language), 2
 3G programming and, 349-350
 animation example, 352-355
 animation, 350
 audio, 350
 commands, 5-7
 converting from HTML, 3
 dynamic content
 JSP, 68
 servlets, 67-68
 graphics, 350
 HTML conversion, 350
 program structure, 5
 servlets, 95
 syntax, 5-7
 toolkits, 2
 video, 350

WML Script
 commands, 7-8
 control structures, 8
 functions, 8
 Information Master, 9-15
 libraries, 7-9
 operators, 7-8
 Restaurant, 9
 Restaurant Application, 15-24
 syntax, 7-8

wml tag, 5

WMLBrowser library, WML Script, 7, 9

WMLScript, 2

WSP (Wireless Session Protocol), 91
 client/server setups, 152

WTA (Wireless Application Environment), 49
 Accessing emergency services through WTAI
 (WML code), 61
 applications, 49-50
 architecture, 50-51
 call log model, 53
 event binding, 56, 60
 event management, 60
 Event model, 54-56
 functions, WML Script file, 58-59
 interface libraries, 52
 logical indicator model, 53
 Network command WTAI library, 56
 network message model, 53
 network specific WTAI library, 57-58
 permissions, 51
 phone book model, 53
 Public WTAI library, 58
 security, 51, 52

state management, 60
 storage, 53-54
 user agent, 51, 60
 voice call model, 53
 WML Script, appl.wmls listing, 59

WTA server, 51

WTAGSM library, 58

WTAI
 function calls example, 60-62
 function libraries, 56-59
 scripting interface, 56-59

WTLS (Wireless Transport Layer Security)
 client/server setups, 151

WTP (Wireless Transaction Protocol) client/server
 setups, 152

X

XHTML (eXtensible HyperText Markup Language), 3
 3G programming and, 349-352
 location-based services, 360-363
 animation example, 355-356
 audio files example, 356-358
 video files example, 358-359
 attributes, 351
 HTML comparison, 351
 nested tags, 351
 opening/closing tags, 351
 quotation marks, 351
 syntax, 351
 tags, 351

XML (eXtensible Markup Language), 2-3
 3G programming and, 350-351, 364-366
 deployment descriptor files, 487
 entries, PAP push submission, 90
 namespace references, 351
 nesting tags, 351
 opening/closing tags, 351
 parsing, 351
 root element, 351
 SQL Server 2000, support, 501-502
 WML definition, 5

XSL
 3G programming
 audio files, 367-369
 video files, 369-370
 animation display in XML, 364-366

Hungry Minds, Inc.

End-User License Agreement

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book (“Book”). This is a license agreement (“Agreement”) between you and Hungry Minds, Inc. (“HMI”). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

- 1. License Grant.** HMI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the “Software”) solely for your own personal and non-commercial purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.
- 2. Ownership.** HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM (“Software Media”). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensors.
- 3. Restrictions on Use and Transfer.**
 - (a)** You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.
 - (b)** You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.
- 4. Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in Appendix A of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in Appendix A and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.
- 5. Limited Warranty.**
 - (a)** HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.
 - (b)** **HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.**

- (c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. Remedies.

- (a) HMI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: *WAP, Bluetooth, and 3G Programming: Cracking the Code*, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.
- (b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.
- (c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. U.S. Government Restricted Rights. Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

8. General. This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

Sun Microsystems, Inc.

Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. **LICENSE TO USE.** Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.
2. **RESTRICTIONS.** Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.
3. **LIMITED WARRANTY.** Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.
4. **DISCLAIMER OF WARRANTY.** UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.
5. **LIMITATION OF LIABILITY.** TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.
6. **Termination.** This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.
7. **Export Regulations.** All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).
9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.
10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.
11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

Java(TM) 2 Software Development Kit (J2SDK), Standard Edition, Version 1.3 SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 4 (Java(TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software complete and unmodified for the sole purpose of designing, developing and testing your Java applets and applications intended to run on the Java platform ("Programs").
2. License to Distribute Software. Subject to the terms and conditions of this Agreement, including, but not limited to Section 4 (Java (TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the Software in binary code form only, provided that (i) you distribute the Software complete and unmodified and only bundled as part of, and for the sole purpose of running, your Programs, (ii) the Programs add significant and primary functionality to the Software, (iii) you do not distribute additional software intended to replace any component(s) of the Software, (iv) you do not remove or alter any proprietary legends or notices contained in the Software, (v) you only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.
3. License to Distribute Redistributables. Subject to the terms and conditions of this Agreement, including but not limited to Section 4 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the binary form of those files specifically identified as redistributable in the Software "README" file ("Redistributables") provided that: (i) you distribute the Redistributables complete and unmodified (unless otherwise specified in the applicable README file), and only bundled as part of Programs, (ii) you do not distribute additional software intended to supersede any component(s) of the Redistributables, (iii) you do not remove or alter any proprietary legends or notices contained in or on the Redistributables, (iv) you only distribute the Redistributables pursuant to a license agreement that protects Sun's interests consistent with the terms contained in the Agreement, and (v) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection

with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

4. **Java Technology Restrictions.** You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you create an additional class and associated API(s) which (i) extends the functionality of the Java platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create, additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.
5. **Trademarks and Logos.** You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use you make of the Sun Marks inures to Sun's benefit.
6. **Source Code.** Software may contain source code that is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.
7. **Termination for Infringement.** Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303

License Agreement: Forte for Java, release 2.0 Community Edition for All Platforms

To obtain Forte for Java, release 2.0, Community Edition for All Platforms, you must agree to the software license below.

Sun Microsystems Inc., Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. **LICENSE TO USE.** Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.
2. **RESTRICTIONS.** Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.
3. **LIMITED WARRANTY.** Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.
4. **DISCLAIMER OF WARRANTY.** UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.
5. **LIMITATION OF LIABILITY.** TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.
6. **Termination.** This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.
7. **Export Regulations.** All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You

agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).
9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.
10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.
11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

JAVA™ DEVELOPMENT TOOLS FORTE™ FOR JAVA™, RELEASE 2.0, COMMUNITY EDITION SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java(TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software complete and unmodified for the sole purpose of designing, developing and testing your [Java applets and] applications intended to run on the Java platform ("Programs").
2. License to Distribute Redistributables. In addition to the license granted in Section 1 (Redistributables Internal Use and Development License Grant) of these Supplemental Terms, subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute those files specifically identified as redistributable in the Software "README" file ("Redistributables") provided that: (i) you distribute the Redistributables complete and unmodified (unless otherwise specified in the applicable README file), and only bundled as part of your Programs, (ii) you do not distribute additional software intended to supersede any component(s) of the Redistributables, (iii) you do not remove or alter any proprietary legends or notices contained in or on the Redistributables, (iv) for a particular version of the Java platform, any executable output generated by a compiler that is contained in the Software must (a) only be compiled from source code that conforms to the corresponding version of the OEM Java Language Specification; (b) be in the class file format defined by the corresponding version of the OEM Java Virtual Machine Specification; and (c) execute properly on a reference runtime, as specified by Sun, associated with such version of the Java platform, (v) you only distribute the Redistributables pursuant to a license agreement that protects Sun's interests consistent with the terms contained in the Agreement, and (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any

claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

3. **Java Technology Restrictions.** You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you create an additional class and associated API(s) which (i) extends the functionality of the Java platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create, additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.
4. **Java Runtime Availability.** Refer to the appropriate version of the Java Runtime Environment binary code license (currently located at <http://www.java.sun.com/jdk/index.html>) for the availability of runtime code which may be distributed with Java applets and applications.
5. **Trademarks and Logos.** You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use you make of the Sun Marks inures to Sun's benefit.
6. **Source Code.** Software may contain source code that is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.
7. **Termination for Infringement.** Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303