DEVELOPING Intranet Applications with Java

by Jerry Ablan

C O N T E N T S

Introduction

I
Planning an Intranet

Chapter 1 Intranets and Java

- What Is an Intranet?
- Using Web Services on Your Intranet
- Why Build an Intranet?
  - Intranets Are Affordable
  - Intranets Save Money
  - Intranets Are Highly Efficient
- Java in Brief
  - Java: The Early Days
  - Java Gets the Official Stamp of Approval

- Java in Action
- Rapid Development with Java

- Using Java on an Intranet
  - Using a Java Application to Track Employee Files
  - Using a Java Application to Schedule Appointments, Meetings, and Conferences
  - Using a Java Application to Track Who Is in the Office
- Summary

Chapter 2 Designing Intranet Applications with Java

- Intranet Programming with Java
  - Applets versus Applications
  - Conceptualization and Design of Intranet Applications
- Creating Applets
  - The Benefits and Drawbacks of Applets
  - Browsers for Your Applets
- Placing Applets in HTML Documents
  - Introduction to HTML
  - The Java Extensions to HTML
- Creating Applications
  - More Options with Applications
  - Applications Have a Different Structure
  - Applications Have Different Security Considerations
- Running Applications and Applets
- Summary

Chapter 3 Planning Your Intranet Environment

- Building Blocks for Creating a Perfect Intranet
  - Managing Expectations
  - Managing Perceptions
  - Managing Strategies
  - Managing Goals
  - Managing Rules
  - Managing Behavior

- Determining the Best Organization for Your Intranet
  - Learning from the Past
  - Applying the Past to Your Intranet's Future
- Creating Content for Your Intranet
- Intranet Development Tools
  - Implementing TCP/IP Networking
  - Creating Web Services with HTTP
- Intranet Developer's Resource Tools
  - HTML Development Tools
- Mapping Your Intranet in Four Easy Steps
  - Step 1: Determining Requirements
  - Step 2: Planning
  - Step 3: Design
  - Step 4: Implementation
- Summary

II
Development Concepts and Environments

Chapter 4 Java Development Environments

- Introduction
- Selecting an IDE
  - GUI Development Tools
  - Portability of Code
  - IDE Experience
  - Multiple Language Development
  - The Bottom Line
- Symantec Café
  - System Requirements
  - Overview
  - Pricing and Additional Information
- SunSoft Java WorkShop
  - System Requirements

- Overview
- Pricing and Additional Information

- SourceCraft NetCraft
  - System Requirements
  - Overview
  - Pricing and Additional Information
- Other Offerings
  - Borland C++ 5.0 with Java Enhancements
  - MetroWerks CodeWarrior
  - Java WebIDE
  - Kalimantan
  - Natural Intelligence Roaster
  - Microsoft Visual J++
  - RogueWave JFactory
  - Cosmo Code
  - Summary

Chapter 5 Intranet Security

- Introduction
- Why Security?
  - What Are the Security Features of an Intranet?
  - It's Your Call
- Security on Your Web Server
  - Controlling Access Globally and Locally
  - Username/Password Authentication
  - Authentication Based on Network Hostname or Address
  - Combined Authentication
- Secure/Encrypted Transactions
  - Secure HTTP (S-HTTP)
  - Secure Sockets Layer (SSL)
- The Common Gateway Interface (CGI) and Intranet Security
- Your Intranet and the Internet
  - Firewalls

- Virtual Intranet
- Summary

Chapter 6 Database Connectivity Options

- Introduction
- Database Overview
- Database Terminology
- Database Locations
  - Local and Remote
  - Tiering 1-2-3
- Database Access Methods
  - Native Drivers
  - ODBC
  - SQL
- Databases and Java
  - Access via Web Server
  - Access via Proprietary Server
  - Network Access
  - Direct Access
  - JDBC
  - JDBC Goals
  - JDBC Overview
  - JDBC Vendor Support
- Summary

III
Extending Java for Intranets

Chapter 7 A Model Intranet Application

- Introduction
- A Quick Overview of Intranet Applications
  - Configuration File Processing
  - Logging to Disk or Screen

- Database Connectivity
- Look and Feel
- Coding Style Notes
- Code Layout
  - Parentheses and Code Blocking
  - Using Tabs Versus Spaces
  - Liberal Use of Spaces
  - Multiple Lines Per Statement
- Comments
- Code Order
- Summary

Chapter 8 Utility Classes

- Introduction
- Timers
  - Timer Operations
  - Callbacks
  - Event Timers
  - Why Have Two Timers?
- Java Extensions
  - Extending Java's Date Class
  - Application Configuration Parameters
- Summary

Chapter 9 Logging Classes

- Introduction
- The Log
  - The Log Entry
  - The Log Interface
- The Logging Classes
  - DiskLog
  - ScreenLog
- A Sample Logging Program

- Summary

Chapter 10 Database Classes

- Introduction
- JDBC in Depth
  - The DriverManager Class
  - The Driver Class
  - The Connection Class
  - The Statement Class
  - The ResultSet Class
  - A JDBC Sample Program
- Making JDBC Easy to Use
  - The Connector Interface
  - The SQLFactory Interface
- The Classes
  - The DBConnector Class
  - OracleSequence
- Summary

Chapter 11 User Interface Classes

- Introduction
- 3-D Effects
  - The Effects Interface
- The JifPanel Class
  - JifPanel Design
  - Constructing a JifPanel
  - Smoke and Mirrors
  - Drawing 3-D Borders
  - Tabbing Between Components
- SQL Generation
- The JifPanel Descendants
  - The CalendarPanel Class
  - The ImagePanel Class

- The JifLabel Class
- The JifTabPanel Class
- The StatusBar Class
- The JifDialog Class
- The MessageBox Class
- The PickList Class
- Java TextComponent Extensions
- Change Detection
- Summary

Chapter 12 Putting Them All Together

- Introduction
- Java Compilation Basics
- Java Source Code Files
- Have You Got the Package?
- Making Java Packages
- Introducing the Java Intranet Framework
- Packaging the JIF Classes
- Extending the Framework
- Java Applets
- Making JIF Easy to Use
- The JifApplication Interface
- The Jiflet Class
- Instance Variables
- Constructors
- Methods
- Wrapping Up Jiflets
- Programming with Jiflets
- The Smallest Jiflet
- The HelloWorld Jiflet
- Extending Jiflets for Real-World Use
- DBRecord
- SimpleDBUI

- SimpleDBJiflet
- Summary

IV
Applications Developing with JIF

Chapter 13 Employee Files

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Chapter 14 Human Resources: Benefits Maintenance

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Chapter 15 Conference Room Scheduling

- Introduction
- Application Design
- Database Design
- Implementation

- Building the User Interface
- Interacting with the User
- Database Access
- Reading the Existing Schedule
- Storing Your Schedule
- Generating the SQL

- Programming Considerations
- Summary

Chapter 16 Online In/Out Board

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
  - A Refresh Timer
- Programming Considerations
- Summary

Chapter 17 Online Employee Phonebook

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Chapter 18 News & Announcements

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Chapter 19 Product Maintenance

- Introduction
- Who Would Use This Application?
  - Johnston, Ulysses, Norman, and Kaiser
- Application Design
  - Using a Pick List
- Database Design
- Implementation
  - User Interface
  - The Product Pick List
  - Database Access
- Programming Considerations
- Summary

Chapter 20 Customer Support Maintenance

- Introduction
- Application Design
- Database Design
- Implementation
  - Building the User Interface
  - Database Access
  - Retrieving the Product and Problem Lists

- Product and Problem Selection Changes
- Programming Considerations
- Summary

Chapter 21 Extending the Java Intranet Framework

- Introduction
- jif.util
  - ConfigProperties
  - FileDate
- jif.log
- jif.sql
- jif.awt
  - JifPanel
  - JifTabPanel
  - StatusBar
  - Miscellaneous
- jif.jiflet
- Extending the Applications
  - Benefits Maintenance
  - Conference Room Scheduling
  - News and Announcements
- Summary

appendixes

appendix A Java Resources

- Sun's Java Sites
- Java Information Collection Sites
- Java Discussion Forums
- Notable Individual Java Webs
- Java Index Sites
- Object-Oriented Information
- Java Players and Licensees

appendix B JDK Tools Reference

- JDK Tools Reference
- javac-The Java Compiler
  - Synopsis
  - Description
  - Options
  - Environment Variables
- java-The Java Interpreter
  - Synopsis
  - Description
  - Options
- jdb-The Java Debugger
  - Synopsis
  - Description
  - Options
- javah-C Header and Stub File Generator
  - Synopsis
  - Description
  - Options
- javap-The Java Class File Disassembler
  - Synopsis
  - Description
  - Options
- javadoc-The Java API Documentation Generator
  - Synopsis
  - Description
  - Options
- appletviewer-The Java Applet Viewer
  - Synopsis
  - Description
  - Options

appendix C Java API Reference

- Reserved Words
- Comments
- Literals
- Variable Declaration
- Variable Assignment
- Operators
- Objects
- Arrays
- Loops and Conditionals
- Class Definitions
- Method and Constructor Definitions
- Packages, Interfaces, and Importing
- Exceptions and Guarding

appendix D Java Class Reference

- java.lang
  - Interfaces
  - Classes
- java.util
  - Interfaces
  - Classes
- java.io
  - Interfaces
  - Classes
- java.net
  - Interfaces
  - Classes
- java.awt
  - Interfaces
  - Classes
- java.awt.image

    - Interfaces
    - Classes
- java.awt.peer
- java.applet
    - Inetrfaces
- Classes

appendix E Differences Between Java and C/C++

- The Preprocessor
- Pointers
- Structures and Unions
- Functions
- Multiple Inheritance
- Strings
- The goto Statement
- Operator Overloading
- Automatic Coercions
- Variable Arguments
- Command-Line Arguments

appendix F Java Intranet Framework Reference

- jif.awt
    - CalendarPanel
    - Effects
    - ImagePanel
    - JifCheckbox
    - JifDialog
    - JifLabel
    - JifPanel
    - JifPanePanel
    - JifTabPanel
    - JifTabSelector
    - JifTextArea

    - JifTextField
    - MessageBox
    - PickList
    - ResponseDialog
    - SimpleDBUI
    - StatusBar
- jif.jiflet
    - JifApplication
    - JifMessage
    - Jiflet
    - SimpleDBJiflet
- jif.log
    - DiskLog
    - Log
    - ScreenLog
- jif.sql
    - CodeLookerUpper
    - Connector
    - DBConnector
    - DBRecord
    - MSQLConnector
    - MSSQLServerConnector
    - ODBcconnector
    - OracleConnector
    - OracleSequence
    - SequenceGenerator
    - SQLFactory
    - SybaseConnector
- jif.util
    - CallbackTimer
    - ConfigProperties
    - EventTimer
    - FileDate
    - TimeOut

appendix G What's on the CD-ROM

- Windows Software
  - Java
  - mSQL
  - Servers
  - HTML Tools
  - Graphics, Video, and Sound Applications
  - Explorer
  - Utilities
- About Shareware

Credits

To Kathryn. A dedication haiku:
"Love is a great thing, like puddles after the rain, or a stroll with you." -Jerry Ablan

FIRST EDITION

International Standard Book Number: 1-57521-166-1

HTML conversion by :
M/s. LeafWriters (India) Pvt. Ltd.
Website : http://leaf.stpn.soft.net
e-mail : leafwriters@leaf.stpn.soft.net

Trademarks

President, Sams Publishing	*Richard K. Swadley*
Publishing Manager	*Mark Taber*
Managing Editor	*Cindy Morrow*
Director of Marketing	*John Pierce*
Assistant Marketing Managers	*Kristina Perry, Rachel Wolfe*

Acquisitions Editor	*Beverly M. Eppink*	**Development Editor**	*Kelly Murdock*
Software Development Specialist	*Bob Correll*	**Production Editors**	*Mary Inderstrodt, Ryan Rader*
Copy Editors	*Brice Gosnell, Stacey Houston,Kristen Ivanetich, Howard Jones*		
Indexer	*Johnna VanHoose*	**Technical Reviewers**	*Will Kelly, Billy Vernon*
Editorial Coordinator	*Bill Whitmer*	**Technical Edit Coordinator**	*Lorraine Schaffer*
Resource Coordinator	*Deborah Frisby*	**Editorial Assistants**	*Carol Ackerman, Andi Richter, Rhonda Tinch-Mize*
Cover Designer	*Tim Amrhein*	**Book Designer**	*Alyssa Yesh*
Copy Writer	*Peter Fuller*	**Production Team Supervisor**	*Brad Chinn*
Production	*Debra Bolhuis, Mona Brown, Kevin Cliburn, Betsy Deeter, Jason Hand, Susan Knose, Clint Lahnen, Carl Pierce, Casey Price, Laura Robbins, Ian Smith, Susan Van Ness, Marvin Van Tiem*		

Acknowledgments

Thanks to Sun Microsystems for creating a very cool language!

Thanks to the people at Sams-most importantly, Beverly Eppink. This book would not have been possible without the idea she originally gave to me. I'd also like to thank Kelly Murdock for keeping me, and the book, on track. Thanks!!

I'd like to thank my Internet Service Provider (again!) for providing me with excellent Internet service throughout the writing process. Thanks to Karl Denninger and the folks at MCSNet in Chicago. Keep up

the good work!

Thanks to all my friends at work who helped and encouraged me, especially Eric Reiner and Nick Athanas. Thanks also to Maureen Smith for putting up with me while I did this again!

I'd also like to thank my close friends. With their support, I was able to hide in my office at home and write. I missed many good Friday evenings at George and Alex's because of this book. So thanks to Tom and Nancy Lynch, Tom and Karen Kenny, George Walker, Alex Weismantel, and Jim Burck. (Have I been more sociable lately, Alex?)

I'd like to thank my animals for staying out of my hair: Grendl (Great Dane), Cecil (Dachshund), Buttons (Calico Cat), T.C. (Tabby Cat), and Kato (Tabby Cat).

Lastly, I'd like to thank my wife. Without her support, an endeavor such as this would not be possible. Thanks, Kathryn!

Jeen Velly

& nbsp; ; &nb sp; & nbsp; -**Jerry Ablan**

About the Author

Jerry Ablan (munster@mcs.net) is best described as a computer nut. Jerry has been involved in computers since 1982. He has worked on and owned a variety of microcomputers including several that are no longer manufactured. He has programmed in many languages, including several that are not cool (such as RPG II). Jerry is a Senior Software Engineer at the Chicago Board Options Exchange. There he creates client/server systems for IBM, HP, and microcomputer platforms. He (and his wife) can't believe that people pay him money to program computers!

Jerry is a member of TeamJava (http://www.teamjava.com), the Java Developer's Organization (http://www.jade.org), and the Illinois Java User's Group (http://www.xnet.com/~rudman/java.html). The Chicago Java User's Group was a little too snooty for him.

Jerry lives in a Chicago suburb with his wife Kathryn, their two dogs (Grendl and Cecil), three cats (Uncle Pat, T.C., and Kato), and a tank full of fish. When not working, writing, or otherwise cavorting, Jerry and his brother Dan (dma@mcs.net) operate NetGeeks (http://www.netgeeks.com), an Internet consulting firm in Chicago, Illinois.

Jerry is coauthor of the *Web Site Administrator's Survival Guide* from Sams.net and a contributing author to *Using Java* and *Platinum Edition: Using HTML, Java, and CGI* from Que, as well as *Java Unleashed* and *Intranets Unleashed* from Sams.net.

William R. Stanek (director@tvp.com) is a leading Internet technology expert and a working professional who directs an Internet start-up company called The Virtual Press (http://tvp.com/ and mirror site http://www.tvpress.com/). As a publisher and writer with over 10 years experience on networks, Stanek brings a solid voice of experience on the Internet and electronic publishing to his many projects. He has been involved in the commercial Internet community since 1991 and was first introduced

to Internet e-mail in 1988 when he worked for the government. His years of practical experience are backed by a solid education, Master of Science in Information Systems and a Bachelor of Science in Computer Science. In addition to authoring best-sellers such as Sams.net's *Web Publishing Unleashed* and *Microsoft FrontPage Unleashed*, Stanek advises corporate clients and develops hot new Web sites.

Rogers Cadenhead (rcade@airmail.net) is a Web developer, computer programmer, and writer who created the multiuser games Czarlords and Super Video Poker. Coauthor of *Teach Yourself SunSoft Java Workshop in 21 Days*, he also writes an advice column, "Ask Ed Brice," in the Fort Worth Star-Telegram, and has programmed Java applications for Tele-Communications, Inc. and other clients.

Tell Us What You Think!

As a reader, you are the most important critic of and commentator on our books. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way. You can help us make strong books that meet your needs and give you the computer guidance you require.

Do you have access to CompuServe or the World Wide Web? Then check out our CompuServe forum by typing `GO SAMS` at any prompt. If you prefer the World Wide Web, check out our site at http://www.mcp.com.

Note
If you have a technical question about this book, call the technical support line at (800) 571-5840, ext. 3668.

As the team leader of the group that created this book, I welcome your comments. You can fax, e-mail, or write me directly to let me know what you did or didn't like about this book-as well as what we can do to make our books stronger. Here's the information:

Fax: (317) 581-4669

E-mail: newtech_mgr@sams.mcp.com

Mail: Mark Taber
Comments Department
Sams Publishing
201 W. 103rd Street
Indianapolis, IN 46290

Introduction

Welcome

Hello, and welcome to *Developing Intranet Applications with Java*. I hope you enjoy this book as much as I enjoyed writing it. This is a book that represents a lot of work in the area of Java application programming and design. It is designed to hold your hand while you journey through the hills and valleys of Java application programming. You needn't be creating Intranet applications specifically to use this book. It does, however, focus on corporate and Intranet application development. More importantly, its focus is

Java application programming. Even though the applications might not be your cup of tea (or coffee!), the concepts and source code presented in this book will be of value.

After reading this book, you should have a good understanding of programming Java applications and how to apply that toward creating applications of your own, Intranet-specific or not. This knowledge will help you at work or at play, but most of all it will help you be a better Java programmer and provide you with a rich source code base to use as your coding foundation.

Who Should Read This Book

Although this book is geared toward experienced programmers, beginning programmers will find it useful as well. However, you should have some experience with C, C++, or Java. It would help if you had some database programming experience as well.

How This Book Is Designed

This book is divided into four distinct parts. I'll go over each section in detail and give you a little background on its purpose and content.

Part I: Planning an Intranet

The first part gives you a little background information on intranets. Many readers might not be familiar with the term *intranet* and its implications. Some of the topics covered in this section are

- What an intranet is
- Why build an intranet
- How Java can be used to enhance an intranet
- Intranet application design considerations
- Developing intranet applications
- Java development environments

After finishing the first part of this book, you should have some good, solid knowledge about integrating Java applications into your intranet plans. You will also gain a little insight about developing intranet applications.

Part II: Development Concepts and Environments

This part examines some of the more pressing issues in intranet development with Java. Areas covered in this section are

- Intranet security
- Database connectivity

Part III: Extending Java for Intranets

This part introduces you to an application framework for building intranet applications. This framework is called *JIF*. JIF stands for Java Intranet Framework. JIF is a made up of several

Java packages and is included on the CD-ROM. You can use JIF to create your own applications, or you can modify it for your own needs. It's up to you!

This section covers the foundations of a framework and builds upon it. Before any classes are discussed, however, a model intranet application is presented. This application becomes the driving force of the class creation for the rest of this section. It is also the model used for the sample applications in the next section.

Part IV: Applications Development with JIF

This section presents eight sample intranet applications. These applications are real-world examples of using Java to create database-aware intranet applications. The applications presented are

- Employee files
- Benefits maintenance
- Conference room scheduling
- Online in/out board
- Online employee phonebook
- News and announcements
- Product maintenance
- Customer support maintenance

These are fully functioning applications that really do work! The source code for them is included on the CD-ROM.

Conventions Used in This Book

The following type conventions are used throughout this book:

Italic type is used for

- New terms when they are used

`Monospaced` type is used for

- Source code listings
- Commands to be entered
- Any representation of computer output

`Monospaced Italic` type is used for

- Placeholders within source code-for example, function arguments.

There are also some special conventions used when discussing parameters of classes and functions in their declarations. These conventions are similar to most other programming books that you've seen. They are as follows:

- Square brackets (`[]`) `are used to surround optional arguments or parameters.`

Note
Throughout this book I use several terms interchangeably. Specifically the terms *object* and *class*, and the terms *application* and *program*. Some say that an object is only an instantiation of a class. However, I feel that a class is an object in any stage of life. So, don't be too mad. Also, you will see the terms *method* and *function* used to refer to the same thing, as well as *member* and *instance variable*. They all mean the same things; however, different books call them different things. I have no idea what you, the reader, call them, so I'm trying to please everyone.

Chapter 1

Intranets and Java

CONTENTS

- What Is an Intranet?
- Using Web Services on Your Intranet
- Why Build an Intranet?
  - Intranets Are Affordable
  - Intranets Save Money
  - Intranets Are Highly Efficient
- Java in Brief
  - Java: The Early Days
  - Java Gets the Official Stamp of Approval
  - Java in Action
  - Rapid Development with Java
- Using Java on an Intranet
  - Using a Java Application to Track Employee Files
  - Using a Java Application to Schedule Appointments, Meetings, and Conferences
  - Using a Java Application to Track Who Is in the Office
- Summary

***intra net \intra-net\ n**: an internal network that is sheltered from the outside world*

Networks aren't the same since the Java programming language and intranets burst onto the scene. A few years ago, no one had heard of the programming language named after the pervasive liquid that energizes millions. Today, not only is the Java programming language a buzzword in the computer industry, it is fueling a period of tremendous growth and enthusiasm for networked platform-independent applications.

Intranets, like Java, seemingly sprang to life overnight. The key to intranets is that they apply the best Internet technologies to the internal networks of organizations the world over. When you combine the object-oriented programming language of the future with a networked publishing solution you can only

dream of until recently, you have a powerful toolkit for boosting productivity, enhancing information distribution, dramatically reducing costs, increasing efficiency, and much more.

This chapter discusses the basics of intranets. You learn what an intranet is, how intranets differ from Web sites, and why organizations build intranets. You also learn something about the Java programming language-like how Java can enhance an intranet.

What Is an Intranet?

Millions of people browse the World Wide Web and if you are reading this book, odds are you are one of them. The Web is a networked information system based on hypertext. Hypertext allows you to navigate through networked resources at the click of a button. Using a client application called a browser, you can select highlighted keywords or specified hot areas within a document to quickly and automatically navigate to a new document. Browsers are in fact your window to everything the World Wide Web has to offer. Using Web technologies, you have instant access to anything your company publishes and it is this information-on-demand feature that makes the Web such a hot commodity.

Beneath the system of hypertext documents and the wonderful graphical interface that makes it all work is a complex network-the Internet. The Internet is a global network of millions of computers. Many different technologies are used on the Internet to find, send, and retrieve information:

- E-mail is used to send electronic mail.
- Gopher, Archie, and Veronica can be used to find information.
- FTP is used to send and retrieve files.
- Telnet is used to log into remote hosts.
- The Web is used to browse hypertext resources.

Trillions of research dollars went into developing the Internet and the tools that make it work. Since some of it was paid for with your tax dollars, wouldn't it be nice to put this technology to work for you and your company? This is where intranets come in.

An intranet is a network within an organization-an internal network-that adapts Internet technologies for use in its information infrastructure. Worldwide, the most common Internet technology put to use within organizations is the Web's hypertext system. For this reason, many developers associate Web publishing on an internal network with intranets.

Using your intranet as a publishing solution, employees throughout your organization can quickly find answers to questions. They don't have to search massive policy manuals or learn the commands to interface with the company database. To find information, all they have to do is click on a hypertext reference or enter a word or two at a prompt.

Ideally, your intranet puts to use many different Internet technologies including Internet e-mail, FTP, Telnet, and Web services. You might be wondering why you might want to use all these Internet services. After all, most networks are set up for file transfer with FTP, remote host logins, and e-mail. However, your internal network probably uses commercial software designed for a specific operating system. Further, this software is probably not entirely user and administrator friendly.

Take e-mail for example. Most networks use an e-mail system. In a large organization, mail gateways and servers are needed to transfer e-mail from one area of the network to another. These gateways and servers are responsible for translating or encapsulating the protocol of the e-mail software so your messages are readable on the receiving end.

Maintaining this maze of servers and gateways is the responsibility of the network administrator. When the system fails, as it inevitably does from time to time, users may lose mail and the administrator may lose sleep. By using an e-mail system designed for the Internet, you can end the nightmare. Users on any platform, be it UNIX, Mac, or Windows 95, can use the same software to send and receive messages. But best of all, you eliminate the necessity for e-mail servers.

Using Web Services on Your Intranet

Setting up a Java-ready intranet with Web services is a key focus in this book. To get started, you need three things:

- Server software
- Java-capable browser software
- A Java development environment

Note
Browsers that you can use with Java are discussed in Chapter 2, "Designing Intranet Applications with Java." Although the Java Development Kit includes a basic developer's environment, many developers prefer more advanced development tools. The best development tools currently available are discussed in Chapter 4, "Java Development Environments."

Generally, all networks have computers designated as workstations and servers. The workstations are the computers used by your end-users. The servers are the computers that provide services to the workstations. For your intranet, you need one or more servers to provide essential services, which includes Web server software and applications.

The Web server software is used to display hypertext documents that you publish on your intranet. Intranet publishing is very different from Web publishing. When you publish on the Web, you are making information and products available to the world community. Yet when you publish on your intranet, the information and products are only available to those with access to the internal network.

Using security mechanisms built in to most server software, you can restrict access to information and products published on your intranet, which allows you to selectively disseminate information within the company. For example, your Java spreadsheet application may only be accessible to personnel in the finance department. You may further restrict access by adding login names and passwords. In this way, corporate financial records are only accessible to those who need to know.

Your intranet also needs applications and this is where Java comes in. With Java you can create powerful applications that run on virtually any operating system on your network. Unlike programs coded in other

programming languages, Java programs are not system resource hogs. A typical advanced application for Windows 95 coded in C++ requires 8-16MB of memory. If you follow sound object-oriented design techniques, the same application coded in Java may require only 2-4MB memory.

Why Build an Intranet?

If you are a software developer, programmer, or network administrator, management has probably asked you a hundred different questions about intranets. Sure, thousands of companies are racing to set up intranets but does your organization need one? The answer is a resounding yes, even for a small organization, and here are the top three reasons why:

- Intranets are affordable.
- Intranets save money.
- Intranets are highly efficient.

Intranets Are Affordable

Intranet services generally do not require a substantial amount of system resources. You probably don't need to use a high-performance computer to provide the services. In fact, on most networks, you find that you can use an existing computer to provide the necessary services. This computer may be a server with other roles within your existing network or a workstation sitting on someone's desk. Because you can use your existing network as the basis of your intranet, the cost of setting up an intranet is negligible.

More good news is that you may not need full time developers or administrators to maintain the intranet. Your current network should have a system administrator capable of taking on the additional role as the intranet administrator. Primarily this is because intranet servers are easy to manage once they are up and running.

The company also needs someone to create the wonderful Java applications for the intranet, which is probably where you fit into the picture. This book is loaded with information on Java applications you can develop for intranets.

Intranets Save Money

You may be surprised to learn that creating an intranet can actually save you and your company hundreds, thousands, and possibly millions of dollars. If you don't think this is possible, find out how much the company spent on software purchases in the last 2-3 years. Wouldn't it be great to drastically reduce the need to purchase commercial software? By developing your own applications for the intranet, you can do just that.

An intranet can offer immediate savings in other areas as well. The cost of a browser that includes an e-mail program is about $25 to $40 per license. Typical Web server software costs between $99 and $999. Using the Web server and browsers on your intranet, you can publish documents and send e-mail.

Compare the price of the intranet solution to the hefty prices of the software currently used on most internal networks to put documents into electronic format and send e-mail. Generally, you need to purchase the document creation tool and the e-mail tool separately. A popular tool for creating

documents on UNIX systems costs more than $1000 for each license. Many commercial e-mail packages require server software as well as client software. So, you not only have to pay $60 to $100 per license for the e-mail software, you also need to purchase e-mail server software at a cost of hundreds of dollars.

Intranets Are Highly Efficient

Traditional ways to spread information throughout an organization are through meetings, memos, newsletters, and the postal system. In recent years, e-mail has played an increasing role in disseminating information in the workplace. Your e-mail message can reach users in milliseconds. Still, e-mail is not the most efficient way to spread information. For example, you don't want to distribute a 200-page policy document via e-mail. Generally, large or important documents are distributed through the company's mail room or the postal system.

You can take the same 200-page policy document and publish it on your intranet, making the document instantly accessible to anyone in the organization. Anything you publish on your intranet is easily searched, indexed, and cross-referenced. Because the document can be fully indexed and easily searched, employees are able to quickly find the areas of the policy document they need to read or are interested in.

Java in Brief

You have probably read all about Java in other books and after reading the basic and intermediate topics those books cover, you are ready to move on to more advanced topics. So instead of spending a dozen pages to tell you about Java's features, I summarize Java's features and history during a brief tour of the Web.

Java: The Early Days

If you have browsed the World Wide Web, you have probably seen Java in action. As unbelievable as it seems, Java was formally introduced to the world in May 1995 and it has been the hottest buzzword in the computer industry ever since. Java was born at JavaSoft (`www.javasoft.com`) and if you are a current Java programmer, you probably visit the home site featured in Figure 1.1 regularly. Most programmers and Java developers want to go straight to the Developer's Corner section of the Web site, which is where you can download the latest version of the Java Developer's Kit and extension Application Programmer's Interfaces (APIs).

Figure 1.1 : *JavaSoft's site on the Web is a great place to visit frequently.*

Note

JavaSoft is an operating company of Sun Microsystems. Like many Internet technology companies, JavaSoft has played the Internet name game. Back in 1991, JavaSoft was a special technologies group within Sun called Green. The Green group would later become FirstPerson, Inc. and finally, they would become JavaSoft. The name changes go along with the changing role of Java. The initial goal of the Green group was to establish Sun Microsystems in the commercial electronics market. Fortunately, things didn't turn out quite as Sun Microsystems planned and the project the Green group started became the platform-independent programming solution for networks.

Intranet developers want to pay particular attention to the Java Database Connectivity (JDBC) and the Java Intranet Framework (JIF) APIs. JDBC enables developers to write Java applications that access databases. JIF is the key to developing intranet applications complete with user-friendly interfaces. You learn all about JDBC in Chapter 6, "Database Connectivity Options," and the classes and methods of the JIF API are explored in detail in Part III, "Extending Java for Intranets."

Note

The JIF API was developed by the authors of *Developing Intranet Applications with Java* and is added as an extra value to our readers. You will build this API in its entirety in Part III.

The first version of Java released for general use on the Internet was an alpha version. Alpha versions of software applications and programming languages are generally released to developers for review, comments, and bug fixes. The Java programming language has come a long way since that original alpha. A major area of change is the developer's tool kit. The current developer's tool kit includes seven powerful tools capable of meeting the needs of most programmers. Table 1.1 summarizes the tools in the current JDK and describes their uses.

Table 1.1. Tools in the JDK.

Executable	*Tool Name*	*Description*
`appletviewer`	Java applet viewer	Displays applets.
`java`	Java interpreter	Runs Java bytecode.
`javac`	Java compiler	Compiles Java programs into bytecode.
`javadoc`	Java documentation generator	Creates documentation in HTML format from Java source code.
`javah`	Java header and stubs file generator	Creates C language header and stubs files from a Java class.
`javap`	Java class file disassembler	Disassembles Java files and prints out a representation of Java bytecode.
`jdb`	Java language debugger	Finds problems in your Java code.

Note
You should be very familiar with the tools in the JDK, especially the interpreter, compiler, and debugger. If you are not, you may want to refer to appendix B, "JDK Tools Reference." The appendix contains helpful hints and command summaries for each of the JDK tools.

Java Gets the Official Stamp of Approval

In January 1996, JavaSoft officially released Java, version 1.0, and the enthusiasm for Java really took off. Companies from every sector of the business community started putting Java to work on their intranets. These organizations include: banks, distributors, shipping companies, advertising agencies, real estate agencies, publishers, and manufacturers. They are using Java because it is so easy to develop universally usable intranet applications with Java.

On a typical network, you find one or more operating systems. The UNIX operating system may be in use by an engineering or graphics development division. The Windows NT operating system may be in use by the sales division. The Macintosh System 7 may be in use in the advertising, marketing, or desktop publishing department. Increasingly, different operating systems are in use within the same department or office. So what do you do when a person trained only on Windows NT moves to an office that doesn't use Windows NT?

You retrain the person, showing him or her how to deal with the new operating system. You may also need to send the employee to special classes that teach him or her how to use all the applications installed on the new operating system. There is a tremendous difference between a popular desktop publishing program for UNIX, FrameMaker, and a popular desktop publishing program for Windows 95, Microsoft Publisher.

What makes these programs so different is their interface. FrameMaker and MS Publisher have entirely different sets of menus, options, and commands. Why can't you develop an advanced word processor that is usable on any operating system and features a familiar interface on any system where the program is running? Enter Java. With Java, you can develop applications that are usable on any operating system and have the same familiar interface no matter if they are running on a UNIX system or a Windows NT system.

Nowhere is the enthusiasm for Java more evident than at the Gamelan Web site shown in Figure 1.2-the official Java repository. In the early days of Java, Gamelan (`www.gamelan.com`) was *the* place to learn about what others were doing with Java. Today, there is such an incredible volume of Java development that it is impossible for any organization to keep track of it all. Still, the folks at Gamelan make a valiant effort to keep up.

Figure 1.2 : *At Gamelan, you can learn about what others are doing with Java*.

Java in Action

If you visit Gamelan, make your way past the thousands of programs designed for entertainment to the ones designed for their utility. These are the programs that help you develop and add to the usefulness of your intranet. A must see program for anyone tracking the commercial marketplace is the WallStreetWeb. The home page for the WallStreetWeb (`www.wallstreetweb.com`) is shown in Figure 1.3.

Figure 1.3 : *The WallStreet Web home page.*

The WallStreetWeb is the perfect program to demonstrate the versatility of Java. When you access the WallStreetWeb home page, the WallStreetWeb applet is downloaded to your computer. As soon as the download finishes and the applet starts, you see a dialog box prompting you to enter a user name and password. This dialog box is shown in Figure 1.4. If you have a WallStreetWeb account, you can log in and access real-time stock quotes. If you don't have an account, you can log in as a guest. Guests have limited access to the program's features, but can obtain quotes on certain stocks.

Figure 1.4 : *Using this dialog box, you can log onto the WallStreet Web.*

The WallStreetWeb (Figure 1.5) communicates with your system using a Web server that can be located halfway around the world from your location, yet you can still obtain stock quotes instantly. Even more amazing is the fact that the program may have been developed on a completely different operating system than the one you use regularly, yet the program downloads and runs on your system without making any changes to the original program. This true platform independence is what makes Java the most versatile programming language in the world.

Figure 1.5 : *The WallStreetWeb applet in action.*

Another powerful feature of Java is its strict security model. If you refer to Figure 1.4, you see a warning at the bottom of the dialog box stating `Unsigned Java Applet Window`. The warning ensures that the user knows the program does not have a digital signature and is not to be trusted. Java includes many other security mechanisms to ensure the integrity of your network and that systems cannot be compromised.

Note
Security is a major issue in network programming. In the chapters ahead you learn important security issues related to Java programming, intranets, and servers. To learn more about security in Java programming, refer to Chapter 2.

Platform independence and strict security are features that make Java the perfect network programming language. As you develop your intranet, keep these features in mind.

Rapid Development with Java

Not only is rapid development possible with Java, it is the status quo. As most current Java developers know, the Java Application Programming Interface consists of predeveloped code that you can use in your applications. This predeveloped code is organized into packages filled with useful classes and methods designed to make it easy to program with Java.

The core package for the Java programming language is called `java.lang`. Although the `java.lang` package provides the core functionality of the Java programming language, it is not the only package included in the official Java Developer's Kit from JavaSoft. The official developer's kit includes eight basic packages: `java.applet`, `java.awt`, `java.awt.image`, `java.awt.peer`, `java.io`, `java.lang`, `java.net`, and `java.util`. The JDK also includes an add-on package called `sun.tools.debug`. Together the basic and add-on packages provide everything you need to create and debug advanced Java applications.

The API in the developer's kit is only the beginning for Java APIs. Currently, there are eight other APIs in development by JavaSoft:

- The Java Enterprise API provides almost everything you need to create applications that connect to databases.
- The Java Commerce API is the key to enabling electronic commerce.
- The Java Management API is a comprehensive tool for building applications that can manage networks.
- The Java Server API provides almost everything you need to develop Java-powered servers for the Internet and intranets.
- The Java Media API is an advanced toolkit for creating applications that use multi-media.
- The Java Beans API helps you create plug-ins and software modules that interact with existing architectures for object linking and embedding.
- The Java Embedded API defines a minimal set of core functionality that can be used in embedded devices.

The great thing about standardized APIs is that all the packages they contain are thoroughly tested before the API is officially released. What this means is that you have millions of lines of code at your fingertips. You can use this code to develop Java applications without having to re-invent the wheel.

A perfect example of this is an application called Jompanion, which is shown in Figure 1.6. Jompanion is a fairly advanced text editor that I created for *Peter Norton's Guide to Java Programming*. Jompanion includes all the features you want in a text editor-cut, copy, paste, find, replace, replace all, adjustable font type and size, file creation, file save, and the ability to have multiple open files. A similar text editor written in the C programming language has more than 10,000 lines of source code, yet because Jompanion is written in Java, the source code is slightly more than 800 lines.

Figure 1.6 : *Jompanion: A text editor written in Java.*

Using Java on an Intranet

By now, you know that Java is a great programming language for networks. What you may not know is how to put Java to use on your intranet. In this section, you find three specific examples of using Java to enhance an intranet. This is only a starting point meant to get you thinking about the kinds of Java applications you can develop.

Using a Java Application to Track Employee Files

Every company has employees and files related to those employees. Sometimes those files are only a few dozen pages. Other times, the files are hundreds of pages long, especially if the file contains a record of the employee's history with the company. Wouldn't it be great to create a system to track, store, and access those files instantly?

With Java, you can develop an intranet application to do just this. Because you are using Java, the application is able to fully access your databases, even if they are proprietary in nature. But best of all, using the Java Database Connectivity classes and methods, you can access several databases using the same user interface. This means you can use a single application with a standardized and friendly interface to create new entries in multiple databases; to retrieve and collate information; and to update files as necessary.

Note
In Chapter 13, you learn firsthand how to design, code, and implement such an application. In Chapter 14, you learn how to create an intranet application for the human resources department of your organization.

Using a Java Application to Schedule Appointments, Meetings, and Conferences

Life in a business organization can be a rat race of meetings with the staff, conferences with management, and appointments with clients. Tracking, announcing, and scheduling these events requires a great deal of time and effort. Wouldn't it be great to have a universally accessible tool everyone at your organization can use to check schedules and announce meetings?

Using Java as your intranet programming solution, you can create such an application. Because Java has a standard intranet framework API, you can develop the application using fewer resources-time and money-and obtain a better end product. In Chapter 15, you learn all about a Java-powered intranet application for conference room scheduling.

Using a Java Application to Track Who Is in the Office

In Star Trek, the main computer knows the whereabouts of every person on the Starship Enterprise. The traditional way to track who is in the office is with an in/out board. When you remember to walk past the board on your way in or out of the office, the board is accurate. When you forget to update the board, the

board is not accurate.

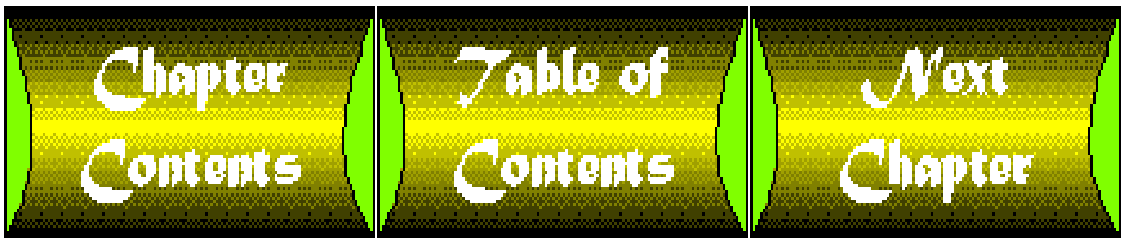
In recent years, some enterprising programmers have created electronic versions of the in/out board. An electronic in/out board can be run automatically when you log in or out, so it gets used. The drawback to existing programs written in traditional programming languages like C/C++ is that they are really only useful as long as you and everyone else in your office uses the operating system for which the program is designed. If some of the office uses the Macintosh System 7 O/S and some of the office uses Windows 95, you cannot easily track or check the status of the office staff.

When you use Java to create an electronic in/out board, you do not have a compatibility problem. Your Java-powered in/out board can be used by anyone in the office. In fact, your program can be used by anyone within the company, which allows you to find out if Susan in marketing is in her office before you leave your desk. For tips on designing and implementing an electronic in/out board using Java, see Chapter 16, "Online In/Out Board."

Summary

In this chapter, you learn about intranets and the Java programming language. With an intranet, you can put the best Internet technologies to use on your organization's network. With Java, you can create powerful applications based on APIs that have been thoroughly tested and proven.

By combining the two hottest technologies in use today, you get the best of both worlds. You can create a network that is affordable, cost effective, and highly efficient. You can build applications for the network that have standard interfaces and are usable on any operating system.



Chapter 2

Designing Intranet Applications with Java

CONTENTS

- Intranet Programming with Java
  - Applets versus Applications
  - Conceptualization and Design of Intranet Applications
- Creating Applets
  - The Benefits and Drawbacks of Applets
  - Browsers for Your Applets
- Placing Applets in HTML Documents
  - Introduction to HTML
  - The Java Extensions to HTML
- Creating Applications
  - More Options with Applications
  - Applications Have a Different Structure
  - Applications Have Different Security Considerations
- Running Applications and Applets
- Summary

---

de sign \di-zin' \ v: to form a plan for

Designing intranet applications involves slightly different methodologies than you may be used to and opens a whole new world of possibilities. This chapter explores the design considerations you make when developing intranet applications with Java. You find many valuable tips that are designed to help you become a successful intranet programmer.

Intranet Programming with Java

Intranet applications are network-aware applications that can retrieve hypertext documents from Web servers, connect to database servers, and examine the contents of file servers. Some intranet applications are considered to be clients, meaning they obtain services from a server. Other intranet applications are considered to be servers, meaning they provide services to clients.

Creating intranet applications without Java is difficult, especially because the programmer must write the networking routines for accessing remote files and databases. Fortunately, Java demystifies intranet application programming. Java has built-in features for working with networks in general. Using these features, you can easily create routines to retrieve and display the contents of a file from any computer on the network. Java also has specific features for intranets. Using these features, you can easily create routines that connect to databases, log network activities, and display friendly interfaces.

Your early plans for your intranet application should focus specifically on your programming language of choice-Java. In Java, there are two types of programs you can create: applets and applications.

Applets versus Applications

When programmers create Java applets, they are generally creating a small application that is designed to be used within the framework of an HTML document. Because applets are designed for use in an HTML document, you need an external viewer to display applets. External viewers you use to display applets include Web browsers or applet viewers.

Note
Note that the size of an applet is relative to its purpose. The real key to applets is they are designed to be used with an external viewer.

When programmers create Java applications, they are generally creating an application that is designed for standalone use. The application does not need to be run within an external viewer, which means you can execute a Java application directly using the Java interpreter. If you follow Java-related discussion groups or have read other Java books, you probably have seen references to Java *apps*. An *app* is a slang term for a standalone application.

While Java applications usually run on a local machine, Java applets usually run on a remote machine. Some Java programs can run both as standalone applications and as applets requiring an external viewer. This is a function of the program's design. Generally, programs that can run as both an app and an applet include a main method required for applications and the `init` and `run` methods used in applets.

Note
Throughout this book, the phrase intranet application is often used to refer to both applets and applications that are designed to be used on an intranet.

Conceptualization and Design of Intranet Applications

Before you create an intranet application, take the time to carefully conceptualize and design the application. The first step is to decide whether you want to create the intranet application as a Java app or as an applet. Three questions that can help you make this decision are

- What is the purpose of the program?
- How is the program going to be used?
- Who is going to use the program?

As you try to answer these questions, look around your workplace and try to determine how best the intranet and this particular application can serve your organization. To help you better understand when to use applications and when to use applets, the next sections examine applet and application creation in depth.

Creating Applets

The most common type of Java program is the applet. Applets are widely used on the World Wide Web. Before you automatically create an intranet application as a Java applet, carefully consider the questions posed in the previous section.

The Benefits and Drawbacks of Applets

When you create an applet, you want the program to be used with an external viewer. The two most common external viewers used with applets are Java-capable browsers and the JDK applet viewer. The benefit of using an external viewer is that the viewer handles starting and exiting the applet. This allows you to create applets without menu bars or dialog boxes. Applets without menu bars and dialog boxes generally have a very simple and friendly user interface.

While applets can have simpler interfaces and sometimes require less code, applets inherit all the overhead of the viewer in which they are running. This means that even if your applet needs only 50KB of memory, you are still constrained by the memory needs of the viewer, which may be an additional 2, 4, or 8MB of memory. On a heavily loaded workstation that is already running multiple applications, such as a word processor or spreadsheet, the additional memory needs of the viewer can seriously affect performance.

Sometimes the overhead involved with applets is not the major issue, especially on intranets where security is a big concern. Using an external viewer, you gain an additional layer of security. This is because the security controls for applets are usually provided by the applet security manager, which can recognize whether applets are running on a local machine or a remote machine.

If an applet exists on the local machine and is in a directory defined in the CLASSPATH environment variable, the applet is loaded by the file system loader. Applets loaded by the file system loader are allowed to read and write files and can access native code.

Note
Code used with Java programs and written in another programming language, such as C or C++, is called native code. Any program that can invoke native code on a computer can gain direct access to protected system areas, which is a major security concern.

If an applet does not exist on the local machine or is not in the right directory, the applet is loaded by the applet class loader and is subject to the restrictions of the applet security manager. These applets are generally restricted from gaining access to the local file system and therefore cannot manipulate files or directories on the local machine in any way. They cannot read or write files. They cannot start another program on the local machine. They cannot make directories. They cannot check file size, modification

date, or type.

Browsers for Your Applets

Browsers are the most popular type of external viewer for applets. Think of browsers as your windows to the Web; change your browser and you get a whole new view of what is out there. Browsers are available for virtually any computer operating system from DOS to Mac to OS/2.

When you set up your intranet, you want to choose a Java-capable browser. The number of Java-capable browsers is steadily increasing. Currently, the most popular browsers that support Java include

- HotJava
- Microsoft Internet Explorer
- Netscape Navigator
- Oracle PowerBrowser

Note
As you read about Java-capable browsers, keep in mind that some versions of a browser may not support Java. In order to use Java on a particular operating system, there must be a Java runtime environment. Currently, there are Java runtime environments for UNIX Solaris, AIX, Macintosh, OS/2, and Windows 95/NT. There are also initiatives to develop Java runtime environments for Windows 3.1, Linux, and Amiga.

HotJava

Not only is HotJava the first browser to support Java, it is also the first browser written entirely in Java. HotJava is being developed by JavaSoft. To say that JavaSoft is developing HotJava slowly is an understatement. Until April 1996, the alpha version of HotJava was the only version available. Unfortunately, HotJava alpha cannot run applets written in beta or later versions of Java.

The current version of HotJava is a beta version. Fortunately, this version supports JDK 1.0 and later. Versions of HotJava are available for Windows 95/NT, Sun Solaris, and soon Macintosh.

Because HotJava is the first Java-capable browser, it is a popular browser. This popularity leads to disappointment for some new users, especially because HotJava is not feature-rich like some of the other Java-capable browsers. You find that HotJava has a rather plain interface and limited extras. Still, HotJava is currently in the testing stages and may yet evolve into a full-featured browser.

JavaSoft's HotJava page is shown in Figure 2.1. You can download an evaluation version of HotJava at the following URL:

Figure 2.1 : *The HotJava page at JavaSoft.*

`http://www.javasoft.com/java.sun.com/HotJava/index.html`

Unique Solutions with HotJava

Although HotJava is not the most advanced Java-capable browser today, there are many great reasons to choose HotJava as an intranet publishing solution, especially if you need to handle unique content or protocols. Traditional browsers are limited in the type of content they can handle and the protocols they can use. When you want to display documents in a unique or proprietary format such as rich text format (RTF), your traditional browser probably will not be able to handle the format directly and will depend on an add-on module or helper application to display the document. If an add-on for the browser is not available, you are out of luck.

With HotJava, you can easily create your own content handler to display the nonstandard format. Creating content handlers is so easy that many developers needing unique publishing solutions turn to HotJava.

HotJava also allows you to create protocol handlers. Traditional browsers support popular protocols, such as HTTP and FTP. If a protocol is not supported directly in the browser, there is no practical way to extend the browser to support the protocol. For example, most browsers do not directly support the Telnet protocol to access remote hosts. If you want to use Telnet, you start a separate application that is designed for remote host access with Telnet.

Using HotJava, you can create a protocol handler that allows HotJava to remotely access hosts using Telnet. Direct support for a protocol eliminates error messages and confusion when users try to access nonstandard protocols linked in your published documents.

Internet Explorer

The Internet Explorer is a feature-rich browser from Microsoft. As one of the most powerful browsers currently available, Internet Explorer took the Web by storm when it was first released in 1995 and quickly moved to the number two browser on the market. You can obtain free versions of Internet Explorer for Macintosh, Windows 95, and Windows NT.

The popularity of Internet Explorer stems largely from its support for existing HTML standards and unique extensions to HTML. Internet Explorer supports HTML 3.2; all Netscape 1.0 and 2.0 extensions; and powerful multimedia extensions including document soundtracks, scrolling marquees, and inline video. Versions 3.0 and later also feature support for Java and ActiveX. As of July 1996, Internet Explorer was also the only browser to fully support the expanded HTML table model specification.

The Internet Explorer home page at Microsoft is shown in Figure 2.2. From the IE home page, you can access the most current version of the browser and obtain upgrade modules like Internet Explorer VR. You can access the IE home page at:

Figure 2.2 : *Web browsing with the Internet Explorer.*

`http://www.microsoft.com/ie/default.htm`

Netscape Navigator

Although the Internet Explorer is vying with the Netscape Navigator for its coveted position as king of the browsers, the Navigator remains the hands-on favorite. The Netscape Navigator is available for Macintosh, Windows, Windows 95/NT, and UNIX.

Netscape Navigator supports Java and has many features that make it a great choice for your intranet. These features include support for HTML 3.2, plug-ins, and JavaScript. With HTML 3.2, you get support for the advanced features of HTML like tables and client-side image maps. Plug-ins allow you to add modules for inline video, sound, and multimedia. Using JavaScript, you can create client-side scripts for your HTML documents.

Netscape has also introduced unique extensions to HTML with every major release of the Navigator. Currently, Netscape and Microsoft are playing a game of one-up-manship. Netscape Navigator is the first browser to support HTML tables and is the model for the table standard adopted in HTML 3.2. By supporting the expanded HTML table model specification, Internet Explorer went one better in version 3.0. Netscape Navigator 2.0 introduced frames, which are mini-windows within documents. Internet Explorer 3.0 went one better and introduced frames without borders and frames that can float on the page. Netscape added new extensions to the beta release of Navigator 3.0 that include support for frames without borders.

You can download free versions of the Netscape Navigator at the Netscape Web site, which is shown in Figure 2.3. The URL to this site is

Figure 2.3 : *Web browsing with the Netscape Navigator.*

`http://home.netscape.com/`

Oracle PowerBrowser

Although PowerBrowser is a fairly new browser on the market, it has all the features you expect in a browser created by Oracle. Oracle is known for its powerful databases and not surprisingly, PowerBrowser includes a local database called Blaze. With Blaze, you can store and manage large amounts of data efficiently. Because Blaze and PowerBrowser can communicate, you can easily create HTML pages that access the Blaze database. Other features of the browser include support for HTML 3.2 and Java.

Currently, PowerBrowser is available for Windows 3.1 and Windows 95/NT. You can obtain an evaluation version of the browser at the site shown in Figure 2.4. You access this site by pointing your browser to

Figure 2.4 : *Learning about the Oracle PowerBrowser.*

`http://www.oracle.com/products/websystem/powerbrowser/`

Placing Applets in HTML Documents

When you create an applet, you need to create an HTML document to display the applet. This section provides an introduction to HTML and the specific extensions in HTML for Java.

Introduction to HTML

The Hypertext Markup Language is based on a system of markup codes called tags. Each tag provides specific instructions to your browser. These instructions tell the browser how to display the contents of the document.

Most tags are used in pairs. A tag called the begin tag marks the start of an element such as `<P>` that marks the beginning of a paragraph. A tag called the end tag marks the end of an element such as `</P>` that marks the end of a paragraph. The difference between a begin tag and an end tag is the forward slash before the element name. Because HTML is not case-sensitive, `<p>` and `<P>` mean the same thing.

Some characters in HTML are reserved, like the open and close brackets `<>`, which denote an HTML element. To display reserved characters in your HTML page, you tell the browser this is a special character and not a reserved character. To do this, you use a special element name that begins with an ampersand and ends with a semicolon such as `&lt;` for the less than symbol `<`. When browsers see a special character, they display the corresponding symbol if possible.

In general, all HTML documents follow a basic structure that includes:

- A document identifier
- A header element designator
- A body element designator

The document identifier identifies the type of document you are creating. Because you are creating an HTML document, your document begins with the tag `<HTML>` and ends with the tag `</HTML>`. The begin tag `<HTML>` tells the browser the document is an HTML-formatted document and marks the beginning of the document. The end tag `</HTML>` marks the end of the document and is the last item in any HTML document.

A header element immediately follows the begin document tag `<HTML>`. Headers are used to specify key aspects of a document such as its title. The beginning of the header is specified with the begin header tag `<HEAD>` and the end of the header is specified with the end tag `</HEAD>`. For now, the only element you may want to use in the header is the `TITLE` element, which is used to specify a title for your document.

The body element follows the header element and contains the text and objects you want to display in the reader's browser. Like the header, the body has a begin tag `<BODY>` and an end tag `</BODY>`.

You can add attributes to most markup tags. Attributes are used to assign default formats for text or graphics associated with the tag. An example of a tag with an attribute is: `<P ALIGN=CENTER>`. This `ALIGN` attribute set to the value of `CENTER` tells the reader's browser to center a text or graphic element on the page.

Using the markup tags discussed in this section, you can create the framework for an HTML document as follows:

```
<HTML>

<HEAD>
<TITLE> A Cool Java Applet </TITLE>
</HEAD>

<BODY>

Insert the actual body elements here.

</BODY>
</HTML>
```

The Java Extensions to HTML

The latest HTML specification-3.2-includes two elements that are specifically designed for use with Java applets. These elements are APPLET and PARAM. Like most HTML elements, these new elements have many attributes that you can use when you insert an applet in a page. Ideally, you use only the attributes that you need for a particular applet.

Using the APPLET Element

The APPLET element is used to name the applet you are inserting into the HTML document and to define its characteristics. Like most HTML tags, the APPLET element has a begin tag <APPLET> and an end tag </APPLET>.

Note

If you are familiar with HTML, you might be wondering why there is a begin and end tag for the APPLET element. After all, all the necessary instructions are in the begin applet tag <APPLET>, which makes the end tag </APPLET> seem unnecessary. Between the begin and end APPLET tags, you can define an area of the document that is displayed by browsers that are not Java-capable.

Here is a sample page that has an area for browsers that are not Java-capable:

```
<HTML>

<HEAD>
<TITLE> Using Java </TITLE>
</HEAD>

<BODY>
```

```
<APPLET CODE="AppletFun.class" WIDTH=300
HEIGHT=300>
<P>
<IMG SRC="Champions.gif">
This page contains a Java applet.
You see this message because your browser does
not support Java.
</P>
<P>You should get a Java-capable browser.</P>
</APPLET>

</BODY>
```

The required attributes for the `APPLET` element are: `CODE`, `WIDTH`, and `HEIGHT`. The `CODE` attribute is used to name the applet's primary class file-the class file containing the `Applet` subclass references. The `WIDTH` and `HEIGHT` attributes define the initial width and height of the applet's display area in pixels.

Tip

Testing the display of the applet in your document at various display resolutions is extremely important. The most popular display resolution on Windows-based pcs is 640∞480 with an increasing number of users moving to larger displays such as 800∞600 or 1024∞768. On a Macintosh, screen size determines the pixel size of the screen. Currently, many Mac users have a 13" screen, which offers a display resolution close to 640∞480. On UNIX systems and primarily Sun Microsystems SPARCs, the display resolution is often set at 1154∞864.

Here is a sample HTML document using the required attributes:

```
<HTML>

<HEAD>
<TITLE> Java to the Max </TITLE>
</HEAD>

<BODY>

<APPLET CODE="CoolApplet.class" WIDTH=300 HEIGHT=300>
</APPLET>

</BODY>
</HTML>
```

In the example, the HTML document and the applet `CoolApplet` must be in the same directory. If you

plan to place applets in a different directory, you must use the optional CODEBASE attribute. The CODEBASE attribute lets you indicate the URL path to the applet. In this example, the directory called java/apps/ contains the class files for the applet:

```
<HTML>

<HEAD>
<TITLE> Java to the Max </TITLE>
</HEAD>

<BODY>

<APPLET CODE="CoolApplet.class" CODEBASE="java/apps/"
WIDTH=300 HEIGHT=300>
</APPLET>

</BODY>
</HTML>
```

Note
URL paths are either relative or absolute. In the example, the URL path is relative, meaning the applet is located relative to the current directory. To specify an absolute path, you use a full hypertext reference, such as: http://www.javasoft.com/applets/applets/NervousText

Additional optional attributes for the APPLET element include: ALIGN, HSPACE, VSPACE, NAME, and ALT. You can use these attributes to enhance the display of the applet and the layout of your document.

The ALIGN attribute is used to specify the alignment of the applet within the document. The acceptable values are shown in Table 2.1.

Table 2.1. The alignment attribute values and their meaning.

Attribute Value	*Meaning*
ALIGN=ABSBOTTOM	Align the bottom of the applet with the lowest item on its line.
ALIGN=ABSMIDDLE	Align applet with the middle of the largest item on its line.
ALIGN=BASELINE	Align the bottom of the applet with the baseline of the text associated with the applet.
ALIGN=BOTTOM	Same as ALIGN=BASELINE.
ALIGN=CENTER	Align applet in the center of the document.
ALIGN=LEFT	Align applet with the left side of the document.
ALIGN=MIDDLE	Align applet with the middle of the baseline of the text associated with the applet.

ALIGN=RIGHT	Align applet with the right side of the document.
ALIGN=TEXTTOP	Align applet with the top of the tallest text on its line.
ALIGN=TOP	Align applet with the topmost item on its line.

The HSPACE and VSPACE attributes are used to specify the amount of white space around the applet. HSPACE defines the horizontal space on either side of the applet and VSPACE defines the vertical space above and below the applet. The value assigned to these attributes is defined in pixels. You use HSPACE=10 and VSPACE=10 in the <APPLET> tag to have ten pixels of space around the applet.

The ALT attribute defines alternative text to display when the browser recognizes the <APPLET> tag but is not capable of running applets. Currently, this attribute is not used widely with applets.

The final optional attribute for the APPLET element is the NAME attribute. Using the NAME attribute, you can assign a designator for the applet that allows it to be targeted by other applets on the page. When applets can target each other, they can interact, which allows you to update applets on the page based on what the user is doing with another applet. The value for the NAME attribute is a unique designator, such as NAME=APPLET1 or NAME=APPLET2.

Using the PARAM Element

Using the PARAM element, you can pass general purpose parameters to an applet. The PARAM element is one of the few elements in HTML that uses only a begin tag. Because each <PARAM> tag is used to set a specific parameter to specific value, you can use multiple <PARAM> tags in your document. Place these tags between the begin and end <APPLET> tags. This is the only element you can insert between the APPLET tags that Java-capable Web browsers takes advantage of when they run an associated applet.

Applets access the parameters set in the <PARAM> tag using the getParameter() method. The <PARAM> tag has two required attributes: NAME and VALUE. The NAME attribute assigns the name that the getParameter() method in your applet searches for. The VALUE attribute is used to set a specific value for the parameter.

Here is an example document using the PARAM element that sets a parameter called TEXT:

```
<HTML>

<HEAD>
<TITLE> Java to the Max With Parameters </TITLE>
</HEAD>

<BODY>

<APPLET CODE="MessageBoard" WIDTH=600 HEIGHT=200>
<PARAM NAME=TEXT VALUE= "Thanks for visiting!">
</APPLET>

</BODY>
</HTML>
```

In your Java applet, you read the value set for the `TEXT` parameter as follows:

```
Msg = getParameter ("TEXT");
```

Creating Applications

Unfortunately, Java programmers don't create applications as often as they create applets. One reason for this is that many programmers don't understand when to use applications.

More Options with Applications

Although applications and applets are similar, there are more options available when working with an application. Primarily this is because applications are standalone programs that require you to create the user interface. For example, the most common method for exiting an application is a Quit or Exit option on a pull-down menu.

When you create an application, you need to build a menu bar, add menus to the menu bar, and options for the menus. One of the options on your menu system is the Quit or Exit option. While having to create your own interface outside the context of a browser may seem like a drawback, you actually have more options available to you when you build a unique interface for your application.

Applications Have a Different Structure

Generally, applications are run by the Java interpreter on a local machine, which eliminates the need to create an HTML page to view your Java program. Additionally, because you do not need an external viewer to run applications, applications can provide users with better performance. Better performance translates to improved response times and, often, increased productivity.

All applications require a static `main()` method. Unlike C++, Java's `main()` method must be part of the public class that defines an application. When you run an application, the Java interpreter executes the `main()` method and in turn, the `main()` method creates instances of objects and lets the objects control the execution of the program.

Applications Have Different Security Considerations

Because applications are run using the Java interpreter, they are not subject to the security constraints of the applet security manager. This gives applications a substantial advantage over applets and means applications have unrestricted access to the client's file system. Therefore, unless you specifically set up restrictions otherwise, standalone applications can

- Read from and write to files.
- Create directories.
- Check file system, type, and modification date.
- Start other programs on the client.

Although being able to access the local file system provides an advantage non-local applets do not have, a lack of security constraints can cause problems. Therefore, if security is a major issue with your

program, you want to carefully consider whether you need a strict security model.

Standalone applications can also run native code. This means that if you have source code in other programming languages that you want to use with your intranet applications, you probably want to create an application. Keep in mind that native code, such as C or C++, is compiled for a specific operating system. Thus if you use native code, you lose the platform independence and widespread portability you gain by programming with Java.

Note
Most organizations that develop software have large libraries of native code. Before you automatically decide that using native code is a done deal, study the current Java APIs. You may be pleasantly surprised when you find that functions you've coded in C, C++, SmallTalk, or some other programming language are already available in Java.

Running Applications and Applets

Most intranets have many different types of servers and follow a loose client-server model. In an ideal client-server environment, clients maintain the application front end, the user interface, and the binary executable for the client interface. Servers maintain the application back end, the server software, and data. Using a strict client-server model reduces traffic on the intranet and balances the demand for resources around the network.

In the real world, many network applications use a file server model where the server has both the binary executable for the client interface and the necessary data files for the application. Because clients execute applications by way of the server and retrieve data from the server, you often have dramatically more network traffic and heavy loads on your file servers.

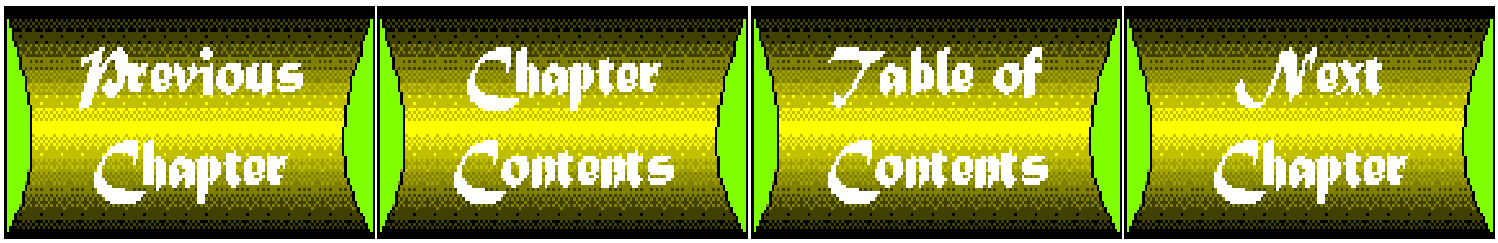
As you design your intranet applications, keep the client server and the file server models in mind. Because Java applications are invoked by the Java interpreter running on a local machine and can retrieve files from remote servers, your Java applications generally follow a client-server model. Java applets, on the other hand, usually run on a remote machine and because of this generally follow a file server model.

Although it seems you should immediately choose the client-server model over the file server model, there are benefits to following a file server model. The two key benefits of the file server model are related to storage and maintenance.

With the client-server model, all workstations generally have a copy of the intranet application on their file system which requires space on the workstation's hard drive. Since there are copies of the application on all your workstations, you must copy new versions of the application whenever you make revisions, which may create a maintenance nightmare. With the file server model, there is usually only one copy of the application and it is on the file server, which makes updating the application very easy.

Summary

After reading this chapter, you should have a clear understanding of the design issues related to Java programming. The fundamental decision you must make is whether to create your intranet application as an applet or as an app. As you've seen, there are many reasons why you may choose to design an applet rather than an application and vice versa. Before you create your intranet applications, consider the underlying issues surrounding applet and app design, including security, file access, the ability to use native code, and network design.



Chapter 3

Planning Your Intranet Environment

CONTENTS

- Building Blocks for Creating a Perfect Intranet
  - Managing Expectations
  - Managing Perceptions
  - Managing Strategies
  - Managing Goals
  - Managing Rules
  - Managing Behavior
- Determining the Best Organization for Your Intranet
  - Learning from the Past
  - Applying the Past to Your Intranet's Future
- Creating Content for Your Intranet
- Intranet Development Tools
  - Implementing TCP/IP Networking
  - Creating Web Services with HTTP
- Intranet Developer's Resource Tools
  - HTML Development Tools
- Mapping Your Intranet in Four Easy Steps
  - Step 1: Determining Requirements
  - Step 2: Planning
  - Step 3: Design
  - Step 4: Implementation
- Summary

In this chapter, you will learn how to plan your intranet. Creating a successful intranet is a matter of planning. Start your intranet on a solid foundation, follow through with good organization, and you can map out an intranet you will be proud of.

This chapter covers the following topics:

- The building blocks for creating a perfect intranet
- Determining the best organization for your intranet
- Creating content for your intranet
- Intranet development and resource tools
- Mapping your intranet in four easy steps

Building Blocks for Creating a Perfect Intranet

Think of the creative process as a building process. Try to build the roof of the house before you lay the foundation, and you are going to have serious problems. Pour the concrete for the foundation of the house before you put in the necessary plumbing for water and sewer access, and you are going to spend more money than you bargained for.

You build a house one step at a time. You ensure the house has a strong foundation. Buildings with strong foundations tend to weather the seasons and time. When you are almost done with the frame of the house, you build a roof. Although the roof of the house is the top of the structure, you do not stop there. It takes more than a covered frame to make a house. You hire an electrician to do the wiring and bring back the plumber to finish the plumbing. Afterward, you hang plaster board, add insulation, finish the exterior, add fixtures, and before you know it, you have a house that you can call home.

You build an intranet in the same way, one step at a time. Your start on the intranet is about as glamorous as the water and sewer pipes waiting for the foundation to be poured around them; for just when you are ready to roll back your sleeves and dive into the intranet creation process with both feet, you might discover you need to conduct research, planning, or consider the requirements of the intranet. When you finally flesh out the foundation of the intranet, you start to build the framework.

The basic components of any intranet are the hardware and software that make it work. The hardware your intranet uses will determine the way the intranet operates. The software your intranet uses will determine what the intranet is used for. Eventually, you finish designing the intranet, but find you still have to develop the hot Java-powered applications for the intranet.

Even when you have completed the design and development processes, the intranet still is not finished. You check the structure of the work for flaws. You make sure you have used the right structure and created the best tools. You examine the fixtures. Once all this is done, you finally have an intranet worthy of the CEO's wholehearted embrace.

Try to build the whole house at once and you will be overwhelmed. The same is true for any creative process. When you are building your intranet and its applications, you need to manage many things on a level of general organization and on a more specific level.

Managing Expectations

If you mismanage expectations, your intranet might not turn out as you planned. Your expectations and the expectations of your superiors might be totally different. Before you start to design the intranet and the Java-powered applications for the intranet, make sure your expectations and the expectations of your supervisors mesh. A good way to do this is to ensure that the communications channels are open and

used.

To ensure that your project is a smashing success, you should discuss expectations throughout the development of the intranet, especially as you develop your intranet applications. If you develop a rapid prototype of key applications, your superiors should be the ones to verify that the designs meet their expectations. If the prototypes do not meet their expectations, maybe the prototypes were an example of what not to do, or maybe the expectations of management are unrealistic. If your prototypes meet or exceed the expectations of your superiors, you have a green light and your project is well on its way to a successful implementation.

You should also manage your personal expectations for the intranet and its applications. Your expectations play a major role in the success of the intranet. The following is a list of do's and don'ts to help you manage expectations:

- Do expect the creation and development process to be challenging and fun.
- Don't expect first efforts to be perfect.
- Don't expect the intranet to be perfect.
- Do expect to revise and improvise as necessary.
- Do expect to learn a lot.

Managing Perceptions

Realistic expectations ensure the success of your intranet. If you perceive the intranet as an impossibly large undertaking, you might cripple yourself mentally. If you perceive the intranet as a trivial undertaking, you will not produce the best possible structure and tools for your organization.

It is best to find a balance in your perceptions about the intranet. As you begin to design the intranet, keep in mind that the intranet creation process is a team effort. Few individuals will be able to handle all aspects of creating the intranet and its applications. For this reason, you should have an accurate perception of your abilities and know when it is in the best interest of the project to delegate tasks.

Managing Strategies

Creating an intranet is exciting and challenging. You'll be breaking new ground, trying new things, and experimenting with new applications. Manage the intranet creation process in whatever way will motivate you. If one way of thinking about the intranet is not motivating you, change tactics. Do whatever it takes to get the job done.

Do not limit yourself to a few strategies or stick with one strategy when it obviously is not working. Make a list of strategies. If one strategy is not working, switch to a new one. If you do not have a new one, create a new one.

The strategy you use can be very basic. A great strategy to start with is to plan to work on the project every day until it is completed. In addition to this strategy, you should add planning to involve both management and users in the development process. The degree of involvement for management and users might need to be adjusted throughout the development process.

Your role in the project should be a part of your strategy. Initially, you might want to work closely with

the development team. Later, you might discover that your best role is to manage the development at a higher level. Or if you are the top programmer or network administrator, you might find that you need to work on application design rather than the actual programming. Adapting your role as necessary can help the project succeed.

Managing Goals

When you start working on the intranet design and creation process, one of the first things you should do is develop goals. Your goals should take into consideration the complexities and nuances of the intranet you plan to develop for your organization. Goals should be clear and relevant to the problem at hand.

Set major goals relevant to the purpose, scope, and audience of the intranet. Also, set minor goals or milestones for the stages of the intranet development and its applications.

Goals and milestones help define the intranet development process as a series of steps or achievements. One major goal could be to complete the planning of the intranet; another major goal could be to complete the design of the intranet.

The series of steps necessary to complete the major goals are the minor goals or milestones. Your first milestone will be to start work on the intranet. Another milestone might be to select and purchase the necessary intranet software, such as Web server software, browser software, and a Java Development environment. Your goals are to complete the major steps of the development process, such as planning and design. You will learn all about these major steps later in this chapter in the section titled, "Mapping Your Intranet in Four Easy Steps."

Managing Rules

As the intranet designer and manager, you will probably create or be provided rules that pertain specifically to the intranet's layout or scope of control, such as the Information Systems department that will have overall responsibility for the intranet after completion. As you start to create the intranet, these rules might seem perfectly acceptable. However, as you conduct planning for the intranet and its applications, you might find that the overall responsibility of the intranet should be divided amongst the departments that will set up intranet servers. If these early rules cannot be modified to fit the current situation, you will have problems. You might encounter delays due to loss of efficiency or the final product might not be what was expected.

No rule should ever be considered absolute. Even the best of rules should be interpreted as guidelines that can vary depending on the situation. Rules for a complex project like your intranet should be flexible and make sense. A rule that conflicts with something you are trying to do should be reexamined. The rule might be inappropriate for the situation you are trying to apply it to.

Managing Behavior

Your intranet will never be implemented if you avoid working on it. Putting off work until something is due is a poor practice. Quitting when things do not go your way or when you seem to have a block is another poor practice.

Even if you thrive on deadlines, plan to work toward intranet's goals and milestones regularly-every day if necessary and possible. You should also plan to work on the intranet and its applications during those times when your thoughts are not flowing. Everyone has bad days and good days. Some days you take more breaks. Some days you work straight through the day and into the night.

You might tend toward other destructive behavior besides avoiding or putting off work. Sometimes programmers go to the opposite extreme. They tear things apart impulsively before letting the work cool off so they can look at it objectively. Never hack your code just because a few users didn't like your application's interface.

Determining the Best Organization for Your Intranet

Managing the aspects of the intranet's design and creation is only the beginning. The next step is to determine the best organization for your intranet. Over the years, three models have developed for information systems like your intranet: centralized, decentralized, and a combination of centralized and decentralized.

Learning from the Past

The three computing models are really driven by the types of computers in use at an organization. Following the centralized model, all computer resources are centered in one location and under the management of one organization. When you think of centralized computing, think of mainframes and computer centers.

With the introduction of file server and client server computing, most organizations moved away from the centralized model toward a decentralized model. In decentralized computing, computer resources are spread throughout the organization and under the management of the departments in which the computers are located. When you think of decentralized computing, think of the high-power workstations and servers.

After the big move to decentralize computer resources and dismantle massive computer centers, many managers had a rude awakening to the anarchy decentralized computing can cause. Imagine an organization where each department sets the rules and decides the standards, like what hardware and software to purchase and how that hardware and software should be set up. Then imagine the nightmare of trying to support the gauntlet of software and hardware installed throughout an organization the size of AT&T.

Because of a lack of control with decentralized computing, many organizations are moving to the happy middle ground of a mixed computing model. In this mixed model, a centralized Information Systems management sets broad policy, such as the direction and purpose of key computing initiatives, and the individual departments are free to work within those guidelines.

Applying the Past to Your Intranet's Future

As you discuss the implementation of the intranet with management, keep the three computing models in mind. While your organization might currently use a specific model, you can apply any of the models to

the design of your intranet and should encourage management to choose the model that will best serve your organization. Ideally, the final decision will be based on the necessary responsibility and control of the intranet resources.

Following a centralized model, a specific department within the organization will be responsible for the intranet. This same department will be responsible for the setup, design, and administration of your intranet servers. The department will also be responsible for creating the necessary publications and applications based on user requests.

With a centralized model, there will usually be a formal approval process for new publications, applications and services. This means that if the Human Resources department wanted an application to track employee files, a formal request would be required. Once the request is approved, the intranet developers would work with Human Resources to create the application. The problem with centralized control and formal approval processes is that they put creativity and timeliness in thumbscrews. Can you imagine having to get formal approval to change the dates in an intranet published memo?

Following a decentralized model, each department within the organization is responsible for its section of the intranet. All departments that want to create intranet services will have to set up, design and administer their own intranet servers. Each department will also be responsible for creating the publications and applications used by the department.

When you use a decentralized model, you cut out the formal approval process for new publications, applications, and services. This means anyone can create intranet resources. Greater freedom and few controls means that new services can be set up quickly by anyone who wants to set them up. This freedom and lack of controls can also lead to abuse of the intranet resources. Who do you blame when someone publishes potentially offensive material or when the usefulness of the intranet deteriorates because so much junk has been created?

By adopting elements of both the centralized and decentralized model that fit the needs of the organization, you might be able to balance the need for strict control with the need for creative freedom. For example, you could create an intranet with a centralized Web server that links together departmental servers. The IS staff would be responsible for maintaining the central server and updating links to resources throughout the organization. The individual departments would be responsible for maintaining their own servers. To ensure the intranet is not abused, one person within each department could be responsible for that department's intranet resources.

Creating Content for Your Intranet

The real stars on your intranet are the applications you plan to develop. Still, you will need content for your intranet. Most of your content will be in the form of hypertext documents that are served by your Web server and displayed by your chosen Web browser.

As you consider the type of content you want to publish on your intranet, think about how you will organize that content. You can organize hypertext documents in many ways. The structure that is best for a particular document depends on the complexity of the material you plan to present. As complexity increases, you manage it by adopting a more advanced structuring method. Specific design models for hypertext documents include

- Linear
- Linear with alternative paths
- Hierarchical
- Combinations of linear and hierarchical
- Integrated web

For a small document with limited complexity, a simple structure is often best. Simple structures include linear and linear with alternative paths. The simplest way to structure a hypertext document is in a *linear* fashion. Using a pure linear structure, you can create a hypertext publication with a structure resembling a traditional print publication. Readers move forward and backward in sequence through the pages of the publication.

An *alternative path* structure gives readers more options or paths through a document. By providing alternative paths, you make the structure of the publication more flexible. Instead of being able to move only forward and backward through the publication, readers can follow a branch from the main path. In a linear structure the branches will rejoin the main path at some point.

The *hierarchical* structure is the most logical structure for a publication of moderate complexity. In this structure, you organize the publication into a directory tree. Readers can navigate through the publication, moving from one level of the publication to the next, more detailed, level of the publication. They can also go up the tree from the detailed level to a higher level and possibly jump to the top level.

The directory tree closely resembles the way you store files on your hard drive in a main directory with subdirectories leading to files. You could also think of the hierarchy as a representation of an actual tree. If you invert the tree, the trunk of the tree would be the top level of the publication. The trunk could be the overview of the publication. The large boughs leading from the trunk would be the next level of the document structure. The boughs could be chapter overview pages. Branches leading from the boughs would be the next level, or the pages within chapters.

A *combined linear and hierarchical* structure is one of the most used forms for hypertext publications. This is because it is an extremely flexible, but still highly structured method. Readers can move forward and backward through individual pages. They can navigate through the various levels of the publication by moving up a level or descending to the next level. They can also follow parallel paths through the document.

The most complex structuring method is the *integrated web*. This method lets the reader follow multiple paths from many options. This is a good method to use when you want the reader to be able to browse or wander many times through the publication you have created. Each time through the publication, readers will probably discover something new.

Intranet Development Tools

After considering the various styles for hypertext documents, you should examine the various tools you will need to develop the intranet. A tool is anything that supports the task you are working on. The tools for unleashing the power of your intranet are based on the existing tools for the Internet itself, which includes protocols, resource tools, and information services.

Implementing TCP/IP Networking

TCP/IP (Transmission Control Protocol Internet Protocol) is the foundation of the worldwide Internet. You must install TCP/IP on your network to enable intranet services.

A protocol is a set of rules for programs communicating on the network. It specifies how the programs talk to each other and what meaning to give to the data they receive. Without TCP/IP setting the rules for your network communications, you cannot use Internet technologies.

The good news is that if your organization already has access to the World Wide Web, you might already have the necessary TCP/IP structure in place. Additionally, TCP/IP is built in to some operating systems, including Windows 95, Windows NT, and most variants of UNIX.

If you have an operating system where TCP/IP is not built in and do not have TCP/IP installed, you will need to purchase TCP/IP software. Fortunately, TCP/IP software is widely available from software vendors. For example, if you want to install TCP/IP on a Macintosh, you can obtain the software directly from Apple or third-party vendors.

Note
If you plan to use a commercial browser, check to see if the software package includes the necessary TCP/IP software.

Creating Web Services with HTTP

An intranet without Web services is like a world without water. The key to the World Wide Web is the hypertext transfer protocol.

HTTP offers a means of moving from document to document, or of indexing within documents. Accessing documents published on your intranet involves communications between browsers and servers.

In a browser, such as the Netscape Navigator, the HTTP processes are virtually transparent to the user. All the user really has to do is activate links to move through your Web presentation. The browser takes care of interpreting the hypertext transfer commands and communicating requests.

Tip
To reduce time spent on training and support, you might want to select a single browser package for use on the intranet. Before selecting a specific browser package for your intranet, you should ensure the developer of the browser makes versions for all the operating systems in use on your network. If the developer does not, you might want to consider another browser.

The mechanism on the receiving end, which is processing the requests, is a program called the Hypertext Transfer Protocol Daemon (HTTPD). A daemon is a UNIX term for a program that runs in the background and handles requests. The HTTP daemon resides on your Web server.

Before setting up or installing server software, you must determine what platform the Web server will run on. Until recently, your choices were limited, but this changed rapidly as the World Wide Web grew in popularity. Today, Web server software and server management tools are available for almost every platform. And, like other software developed for use on the Internet, this software is available as freeware, shareware, and commercial software.

You will find that UNIX platforms have the most options for server software. Until recently, there was only one good choice for the Windows NT environment, but this has changed. There are now many excellent commercial and freeware choices for Windows NT. For other platforms, there is generally only one choice in server software. Having only one choice of server software for your Macintosh or Windows system doesn't mean the quality of the server software is poor. Quite the contrary, the quality of the software is often quite good.

The following listing shows the most popular servers listed according to the platform they run on:

Platform	*Server Software*
Macintosh	WebStar (formerly MacHTTP)
UNIX	Apache
	CERN HTTPD
	ncSA HTTPD
	ncSA S-HTTPD
	Netscape Servers
	Open Market
	WN
Windows	Windows HTTPD
Windows 95	Website
	Microsoft IIS
Windows NT	EMWAC HTTPS
	Netscape Servers
	Purveyor

Note
The best server software for you is most likely the software that will run on the workstation you plan to use as the network's Web server. This ensures your installation and management team are familiar with the server's operating system. An excellent resource for setting up and administering a web site is the *Web Site Administrator's Survival Guide* from Sams.net.

Intranet Developer's Resource Tools

Tools are an essential part of any operation. Resource tools provide the means for sending and retrieving information. There are three basic tools of intranet working:

E-Mail: Electronic mail is a great way to communicate. Think of e-mail as a way to send letters to anyone within the company instantly. Many e-mail programs enable delivery of mail to single users or groups of users. Some e-mail programs even provide ways to automate responses. Most browser packages are packaged with e-mail software.

FTP: File transfer protocol provides the basic means for delivering and retrieving files around the network. The files can be text, sound, or graphics. FTP provides a springboard for many information-based approaches to retrieving information. Many higher level tools that have friendlier interfaces use FTP or a protocol similar to FTP to transfer files. Just about every browser currently available supports FTP.

Telnet: Telnet lets you remotely log in to another system and browse files and directories on that remote system. Telnet is valuable because it is easy to use and basic to the network. When you telnet to another computer, you can issue commands as if you were typing on the other computer's keyboard. On some platforms, like UNIX, telnet is a built-in resource. On other platforms, you will need a telnet tool.

The basic resource tools are indispensable when used for the purpose that they were designed for. They even provide the fundamental basis for many high-level resource tools, but they simply weren't designed for the advanced manipulation of the wealth of information available on the Internet. This is why dozens of information resource tools have been designed to manipulate networked data.

Here is a list of high-level resource tools you might want to use on your intranet:

Archie: A system to automatically gather, index, and serve information on the Internet. Archie is a great tool for searching your intranet's file archives. Once you set up Archie services, users can access Archie resources with their browser.

Gopher: A distributed information service that enables you to move easily through complex webs of network resources. Gopher uses a simple protocol that enables a Gopher client to access information on any accessible Gopher server. Most browsers directly support Gopher.

LISTSERV: An automated mailing list distribution system. Users can subscribe to LISTSERV lists you set up on the intranet, which enables them to read e-mail posted to the list or to post e-mail to the list. Once you set up a LISTSERV server, users can join lists and participate in lists using standard Internet e-mail software. Most browser packages include e-mail software.

Usenet: A bulletin board system of discussion groups called *newsgroups*. Users can participate in newsgroups posting messages to the group and can read messages posted by other newsgroup members. Once you set up a newsgroup server, users can browse newsgroups and post information to newsgroups using a newsgroup reader. Most browser packages include a newsgroup reader.

Wide Area Information Servers (WAIS): A distributed information service for searching databases located throughout the network. It offers indexed searching for fast retrieval and an excellent feedback mechanism that enables the results of initial searches to influence later searches. WAIS servers are best

accessed via CGI scripts, which allow users to search WAIS databases using their browser.

HTML Development Tools

Using HTML development tools, you can quickly and easily create HTML documents for your intranet. There are three basic types of HTML development tools for intranet publishing:

- HTML editors
- HTML templates for word processors
- HTML converters

HTML editors have features similar to your favorite word processor and enable you to easily create documents in HTML format. Typically, these editors enable you to select HTML elements from a pull-down menu. The menu has brief descriptions of elements you can add to the document. The editor places the element in the document in the proper format, which frees you from having to memorize the format. When creating complex forms, you'll find HTML editors especially useful.

Here is a list of some of the most popular HTML editors:

FrontPage
HotDog
HoTMetaL
HTML Author
HTML Writer
Netscape Gold
Pagemill/Sitemill

HTML templates enable you to add the functionality of an HTML editor to your favorite word processor. The great thing about templates is that you can use all the word processor's features, which could include checking grammar and spelling. More importantly, you'll be using the familiar features of your word processor to add HTML formatting to your documents.

Here are several popular HTML templates:

ANT_HTML
EasyHelp/Web
Internet Assistant

Although the task of creating HTML code is fairly complex, some helper applications called *converters* try to automate the task. HTML converters convert your favorite document formats into HTML code and vice versa. At the touch of a button, you could transform a Word for Windows file into an HTML document. Converters are especially useful if you're converting simple documents and are less useful when you're converting documents with complex layouts.

You can find HTML converters for every major word processor and document design application, including BibTeX, DECwrite, FrameMaker, Interleaf, LaTeX, MS Word, PageMaker, PowerPoint, QuarkXPress, Scribe, and WordPerfect. HTML converters are available to convert specific formats, such as ASCII, RTF, MIF, Postscript, and UNIX MAN pages. There are even converters to convert source code from popular programming languages to HTML. You can convert your favorite programs to HTML if they are in these languages: C, C++, FORTRAN, Lisp, or Pascal.

Mapping Your Intranet in Four Easy Steps

Now that you know the basics of intranet organization, content structure, and development tools, you have everything you need to develop a plan that takes you through the creation and implementation of your intranet. The best way to start is to break down the plan into a series of steps, which ensures the intranet development process is manageable.

Here are four steps you should follow:

1. Determining requirements
2. Planning
3. Designing
4. Implementation

Step 1: Determining Requirements

In this step, you try to figure out what you need to complete the intranet design and implementation. You do this by first examining the intranet's purpose, scope, and audience.

Your statement of purpose should identify:

- Why you are building the intranet
- How the intranet will be used within your organization

When you examine the scope for the intranet, think in terms of size and focus. Will the intranet be company-wide? What types of documents, files, and applications will be permitted on the intranet?

Your audience for the intranet is your customer base. Your customers could include all company employees, employees in specific departments, or employees in a single department.

Here's a preliminary plan for an intranet within a specific department:

Intranet for Sales Department

Purpose
Provide support to the regional sales department. Services to include record searches of the customer databases, sales computation, order processing, and automated inventory updates.

Scope
25 computers within the sales department. All resources are to support and promote regional sales. Limited human resource data will be available to management staff.

Audience
All personnel assigned to the regional sales department.

After determining the purpose, scope, and audience, examine your reasonable expectations for the

completed intranet. You translate these needs, goals, and purposes into requirements for the intranet. The basic needs for any intranet are the software development tools that will help you build the necessary intranet services. Software tools for implementing your intranet are examined in the section titled "Intranet Development Tools."

You will want to think beyond your software needs and also look at your hardware needs. Many types of computers are on the market. The IBM pc and pc compatibles have many generations of computer systems based on the different chip sets. Some pcs are based on the 80286, 80386, and 80486 chips. Other pcs are based on the Intel's Pentium chips. The same is true for Macintoshes-you might choose from a whole line of PowerMacs. There is even a Powerpc, a cross between a Mac and a pc. UNIX systems come in many configurations from Sun Microsystems' popular Sparc workstations to Silicon Graphics workstations.

Very often, the best platform for your intranet services is the platform you are most familiar with. The primary reason for this is that different computer platforms use different operating systems and it is the operating system that your intranet services will run under. If you are unfamiliar with the operating system, there will be an extended learning curve as you study both the operating system and the software your intranet runs on.

Here is a sample plan for hardware and software requirements:

Hardware
Web server: Existing 486DX/100Mhz in System Administration area Application Server: Add service to Sun SPARC 10 file server in System Administration area Database Server: Connectivity to existing SYBASE databases

Software
Web Server: Microsoft IIS for Windows NT Browser: Internet Explorer 3.0 E-mail: Add-on module for Internet Explorer

Next, you should consider time, budget, and personnel constraints. If you have only 10 weeks to completely implement the intranet, you might need to hire additional team members to get the intranet finished on time. In this case, hiring a specific number of additional team members would be one of your requirements.

Here's a sample plan for the initial time, budget, and personnel requirements:

Duration
Setup and installation: 30 days Phase-in and testing: 30 days Follow-up training and support: 90 days

Budget
$5,000

Personnel
Management and planning: 1 Installation team: 2 Training and support: 2

If you have a $5000 budget, you will have to scrutinize every aspect of the budget to keep costs down. In this case, you will probably be extremely selective about the development tools you purchase. You will also hire outside help only as necessary. And if the budget constraints are so severe that they would materially affect the success of the intranet, you will want to ensure your superiors are aware of the situation, and possibly make a case for getting a larger budget.

Step 2: Planning

After you determine your requirements for the intranet, you should plan the intranet. An essential part of planning is determining how long the project is going to take and the steps necessary to carry you through the project. For this reason, the planning step can also be a reality check for constraints or requirements.

For example, you determine that it will take a minimum of five months to complete the intranet and install all the necessary services, yet the deadline for project completion given to you by management is two months away. Here, something would have to give and you would have to work hard to manage perceptions and expectations concerning the intranet. You might have to renegotiate the deadline, hire additional team members or eliminate certain time-intensive parts of the intranet.

The more complex your intranet, the more involved your planning will be. The plans for a small intranet could be very basic, a list of steps with deadlines for completion of each step written down in a notepad. The plans for a large intranet could be rendered in detail using a project management tool such as Microsoft Project.

Ideally, your deadlines will not be carved in stone. The best planners use windows for project steps, such as five days for planning or two weeks for preliminary design. There could be hundreds of steps, with multiple steps being performed simultaneously or a handful of steps with each step being performed one after the other. Some steps would be dependent on other steps, meaning they could not be started until certain other aspects of the intranet were completed. Other steps would not be dependent on any other steps and could be performed at any time during the intranet's development.

Tip

Part of your planning should include scheduling necessary training on the intranet and promoting the intranet to company employees. If you don't sell the employees-your customers-on the intranet, your intranet will not succeed.
One way to help sell the intranet is to develop focus groups. Using the focus groups, you can get your customers involved with the development process. Continued involvement in the intranet throughout its development and after its implementation ensures you create an intranet the customers want. It also ensures they have a solid investment in an intranet that they will want to promote and support.

Step 3: Design

The design step is one of the most critical steps. During this step you take your plans to another level of detail. You do this by determining how and where the intranet's hardware and software will be set up. For example, will the intranet's main Web server be located in the computer department or how will the software be distributed?

Use this step as a reminder to sit down with your system administrators and network personnel. You should discuss how you plan to install the hardware and software for the intranet. If there are any misgivings about the intranet, it is better to hear about them before you begin installation. If there are great ideas for improving the planned intranet, you definitely want to consider them before installation.

Tip

For a small intranet, you might be inclined to skip this step altogether. Don't. During this step, you might discover something you overlooked in planning.

Part of your design might be to use a specific section of the current network as a test bed before you deploy the intranet company-wide. In this way, you install the intranet services within a specific department or office. The users in this group are then given access to the intranet for a testing period. Based on the outcome of the testing, you would either continue with the company-wide installation of the intranet or revise your plans accordingly.

Ideally, your intranet team will work closely with the test group. During installation, and when users start using the new services, you should ensure someone is on hand to answer questions and problems that might arise. This individual or group from your intranet team should take notes and make daily progress reports. Based on the input, you could modify your plans as you proceed through the various phases of the implementation for the test group.

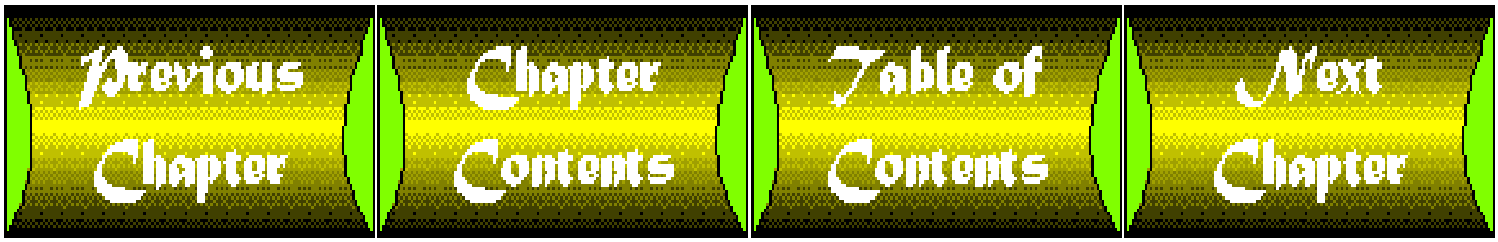
Step 4: Implementation

The implementation step tends to be the longest step in the development of your intranet. During this step, you install your intranet services and create your intranet applications based on the requirements, plans, and designs you created.

Note
Don't stop trying to enhance your plans once you have your intranet blueprint. The key to building a better intranet is to improve your ideas.

Summary

Learning the building blocks for creating a perfect intranet is only the first step toward implementing your intranet. Your intranet will need content, which can be organized in a variety of styles and created with a variety of helper applications. You will also need to set up basic networking protocols, like TCP/IP, and services like the WWW. Once you have selected the basic tools you need to create the intranet and considered how you will organize it, you can map it through completion.



Chapter 4

Java Development Environments

CONTENTS

- Introduction
- Selecting an IDE
  - GUI Development Tools
  - Portability of Code
  - IDE Experience
  - Multiple Language Development
  - The Bottom Line
- Symantec Café
  - System Requirements
  - Overview
  - Pricing and Additional Information
- SunSoft Java WorkShop
  - System Requirements
  - Overview
  - Pricing and Additional Information
- SourceCraft NetCraft
  - System Requirements
  - Overview
  - Pricing and Additional Information
- Other Offerings
  - Borland C++ 5.0 with Java Enhancements
  - MetroWerks CodeWarrior
  - Java WebIDE
  - Kalimantan
  - Natural Intelligence Roaster
  - Microsoft Visual J++

- RogueWave JFactory
- Cosmo Code
- Summary

---

develop \di-'vel-ep\ v: *to make active or promote the growth of*

Introduction

When the Java programming language was introduced in 1995, the only development tool available was the Java Developer's Kit (JDK) from Sun. This set of command-line tools-which is described in appendix B-makes it possible to write, compile, and debug Java programs. However, the JDK is a far cry from the software used to write programs in languages such as Visual Basic and Borland C++. These languages make good use of integrated development environments.

An *integrated development environment* is software that combines several development tools into a single, cohesive package. The assortment usually includes a source code editor, compiler, debugger, and other utilities. These tools work together through the development process and most use windows, drag-and-drop, and other graphical elements. The goal is to make software design faster, more efficient, and easier to debug.

Many IDEs make use of rapid application development (RAD) methods in their approach. *Rapid application development* is a broad strategy to use tools such as an interface designer and prototyping which can speed up the design process. For most of the Java programming environments that have been released, the RAD tools in evidence are graphical user interface builders. Some of these have a direct connection from the interface design tool to the source code so that you can design a component such as a button and go directly to the event-handling code to make something happen when the button is clicked.

As further evidence of Java's enormous popularity, the number of Java development environments has gone from one to more than 40 in a matter of months. Many of these programs are in early beta release or are unreleased in any form, but several have reached the market.

This chapter focuses on some of the environments you are most likely to have heard about and seen available at retail outlets. As you evaluate whether each of these fits your own programming needs, you are better suited to choose from among the on-rushing march of IDEs that have become available.

Note
A frequently updated list of Java IDEs, with reviews and notes regarding these tools, is presented at the following URL: `http://www.cybercom.net/~frog/javaide.html`

The following development environments are described:

- Symantec Café
- SunSoft Java WorkShop
- SourceCraft NetCraft

You learn about the features of each environment, the systems they can run on, and some factors to consider when choosing which to use. Other Java development environments such as Microsoft Visual J++ are described briefly in order to provide a full picture of the choices available to you.

Selecting an IDE

When you are evaluating an integrated development environment for any language, you need to ask yourself some questions about what you need to have on hand when you're ready to develop a program. This holds true for any language and any environment but there are specific issues that relate to Java programming because of its unique features such as portability.

You need to ask yourself the following:

- How important is graphical interface design to your programs?
- Do your programs need to be completely portable across platforms?
- What is your comfort zone with IDE tools?
- Do you want to program in other languages at the same time?

GUI Development Tools

The popularity of windowing systems is well-established and users have begun to shy away from software that does not make use of these features. Though some of us grizzled veterans have a love of the old ways-DOS utilities, the command prompt, batch processing, '70s dance funk-most users today expect their software to have features such as mouse control, point-and-click, and resizeable program windows.

This makes graphical user interface design an important element of most Java programming, so it is important to select an IDE that is strong in this area.

Most Java IDEs are distinguishing themselves from each other by their approach to interface design and the functionality that is possible from within the interface development tool. Also, there is software such as RogueWave's JFactory that is primarily an interface builder rather than an entire IDE.

Note
Jfactory currently is available in beta form. You can find out more information on the product and download a free beta release of the software from the RogueWave Web site at the following URL: http://www.roguewave.com/products/jfactory/jfactory.html

Most Java interface builders work in largely the same way-like a painting program. You start with a blank form and a palette of user interface components. These components are usually abstract windowing toolkit (awt) objects, and the interface builder generates awt code that can be modified by a programmer who is fluent in dealing with the toolkit. Some IDEs such as SunSoft Java WorkShop supersede the awt

by introducing a layer between the awt and the programmer's source code. The goal of a layer such as this is to make it easier to deal with windowing and interface issues. The IDE handles things behind the scenes so that the programmer can concentrate on larger issues.

For some programmers, especially those who spent a lot of time learning the intricacies of the awt, this might not be an attractive feature. Others might be more interested in the power that is offered by these interface builders or might be ready to leave the complexity of the awt behind.

One of the nicest features of these builders is the ability to generate event-handling code at the same time that a user interface component is created. Anyone who has used Visual Basic is familiar with this. You plunk down a text field where you want it on a dialog box, double-click the text field, and then begin entering the source code to control how it operates.

Portability of Code

One of the features to watch for when choosing an IDE is whether it produces code that is fully compatible with the Java class library. Several of the development environments such as Café and Java WorkShop come with their own versions of the JDK instead of being used with an existing implementation of the JDK.

This usually does not matter because one of the goals of any Java IDE is to take advantage of the language's ability to be portable across any platform. Many development environments such as SourceCraft NetCraft work entirely within the JDK, and they create software that does not enhance those features with proprietary extensions.

Others, such as Microsoft Visual J++, add features that require new classes. J++, formerly known under the code name Jakarta, has extensions to the Java language that are specific to the Windows operating system. For an applet or application designed with Visual J++ to be fully portable, it must not make use of these extensions. Because this IDE is not yet available, it remains to be seen whether the advantage of extended features makes up for the significant disadvantage of platform specificity. This issue is hotly debated within the community of Java developers because many believe that the continued growth of the language depends on its ability to stay cohesive and fully cross-platform.

An interesting side issue to the portability question is that most Java development environments are not cross-platform themselves, even when the IDE is touted as being a Java program. All of the major IDEs that have been introduced to date are offered for specific platforms, primarily because of the use of native code.

IDE Experience

One element of IDE use that sometimes gets lost in the shuffle is the skill level required to use them. If the idea of an IDE is to improve your programming, this isn't going to happen if you can't figure out the IDE! An integrated development environment is a complex type of software. It often makes use of a multiple document interface where you can have several windows open at once and a dizzying array of options.

For experienced programmers, this functionality is a great boon. You want to have as much power at your control as possible. A new programmer easily can get lost in the IDEs that are available-especially

if the programmer is still learning the language. When you are busy clearing out space in your brain for a new programming language, you shouldn't have to find room for an IDE at the same time.

Several of the IDEs available for Java are more suited for the code warrior-the multilingual veteran who can throw around jargon like OOP, MUMPS, and male-female connector with the greatest of ease. However, a few of the development environments are more suited for the newcomer because their interface is more approachable and less complex.

The best example of this is Java WorkShop from SunSoft because it uses the familiar web browser interface. A person who is coming to Java from a limited programming background, such as an HTML developer looking to upgrade skills, can find this kind of IDE more suited to his or her taste.

There is a trade-off for this ease of use, of course. An environment like Java WorkShop might require more steps to get a task done, because the functions are not immediately available, or might not offer some of the functionality of a more complex IDE.

Multiple Language Development

Another factor regarding the use of an IDE for some programmers is its use with other languages. Several of these environments, including MetroWerks CodeWarrior and Borland 5.0 C++ IDE with Java Enhancements, are designed to handle more than one language, or are fully equivalent to the company's other development tools.

If the IDE is a complex one, as these are, you learn how to use it and don't have to learn another when you shift gears and program in a non-Java language.

Also, if you write native methods for use in your Java programs, you can use some multi-language IDEs to write that code. The Borland 5.0 Java environment comes with a C and C++ compiler, so native methods can be written alongside Java methods in an integrated manner.

The Bottom Line

The goal of an IDE is to make you a better programmer. As Java developer Chuck McManis wrote in his August 1996 *JavaWorld* column, "I rate the IDEs by my ability to get productive work done while using them."

As you go over the details of the software that are discussed in the following sections, you get a clearer picture of how each environment can help you. Given the number of IDEs that already are available for Java, you should be able to match one with your skill level, programming tasks, and personal taste.

If not, there's always the JDK. It can be matched with word processors, custom interface builders, and other single-feature design tools to create a personalized IDE.

Caution

> If you are downloading tryout or beta copies of several IDEs, as has been done in the course of preparing this material, be advised that these rival products might not co-exist peacefully. Many Java development tools make use of environment variables such as `CLASSPATH` and `JAVAHOME`, and they are not happy if another software tool has claimed these variables for its own purposes. As an example, Café and Java WorkShop are the Prince Charles and Lady Diana of software-they should be kept apart for the benefit of everyone involved. If possible, de-install one IDE before installing the next one on your system and make sure your bootup files are cleaned out as well. If you need more than one IDE active on your system, you can establish multiple configuration files to be executed when one of the environments is used.

Symantec Café

Symantec Café, released in March, is the first development environment that became widely available for Java programming after the JDK. Symantec calls it an integrated development and debugging environment, or IDDE, but the added D doesn't make it different than most IDEs. They usually include a debugger, too.

Café is based on Symantec's C++ environment, but Café is a stand-alone product that does not require that C++ platform in order to run.

Figure 4.1 shows an example of Café at work.

Figure 4.1 : *A screen capture of Symantec Cafe.*

System Requirements

Symantec has released versions of Café for the Microsoft Windows 95, Windows NT 3.5x, and Macintosh systems.

For Microsoft users, an Intel 386 processor and 8M memory are required, but a 486 or better and 16MB memory are recommended. A VGA monitor is needed, but Symantec recommends that an SVGA monitor be used if available. The software, and all of its sample files and help files require 60MB of disk space and a CD-ROM drive.

For Macintosh owners, a Power Macintosh, 68030, or 68040 Macintosh is required, and 16M memory is recommended. The full installation of the software requires 30M of disk space.

Café incorporates the JDK into its release with a full implementation of the Java class libraries and source code samples. You do not need to have the JDK before installing Café. In fact, it is prudent to de-install the JDK before implementing Café to avoid system conflicts between the two.

Overview

Café is a sophisticated IDE that offers an excellent source editor with color highlighting of syntax, an editor for class and hierarchy modification, a Studio tool for interface design, and numerous example applets.

Because it has been on the marketplace for a long time relative to its competitors, Café has the advantage of being more robust than some other IDEs. It also has been documented more fully in books such as *Teach Yourself Java in Café in 21 Days*, available now from Sams.net Publishing.

To aid in the design of a class hierarchy, Café has a class editor for navigating through classes and editing class methods and a hierarchy editor for viewing and modifying Java class relationships. Changes in the source code that affect the class hierarchy can be seen as the program is being written, instead of requiring that it be compiled before changes are reflected in the hierarchy. You also can change the source code from within the class editor-clicking the function or method within a class brings up its source code in a window that can be used to edit the code.

With appexpress, the process of creating a skeleton Java program is speeded up. This and several other Express Agents-Café's term for wizards-make it easier to begin projects and new programs.

In the source editor, Java syntax is highlighted, making it easier to spot typos and other errors immediately. The editor can be customized to behave like several popular programmers' editors such as Brief, Emacs, and Epsilon. It also uses standard Windows cut, copy, and paste commands.

With Café Studio, designing a graphical user interface for your Java programs can be done in a visual, drag-and-drop manner. Studio enables programmers to develop the dialog boxes and other visual elements visually, and it creates event handlers for these components automatically. There's also a menu editor with an active window where the menu can be tested. These resources are saved in separate `.rc` files that can be edited later just as source files are edited, and the `.rc` format is compatible with other design tools that generate `.rc` files.

One interesting aspect of Café Studio is the ability to design a form and dictate exactly how it looks. With the JDK and its abstract windowing toolkit, graphical user interface designers had to allow their work to be changed depending on the platform the applet or application was being run on. This is similar to the way that HTML can be modified to fit the large number of platforms that can be used on the World Wide Web, and it's an approach well-suited to cross-platform design.

With Café Studio, programmers can choose to use one of these variable layout managers or to dictate the position and size of all interface elements.

When you're ready to compile a program, Café provides the option to use Sun's JDK compiler or the Café compiler, which operates more quickly than the current JDK version.

The Café debugger provides several different ways to temporarily halt the execution of code, including a quick-breakpoint feature for a one-time run that stops at a specific line. The debugger also enables a large amount of control over threads in multi-threaded programs. During debugging, a watch view can be used to monitor the contents of variables.

The environment of Symantec Café is highly customizable; all toolbars and palettes can be resized and

placed where you want them onscreen. Several windows can be open at the same time, making it possible to view the object hierarchy while entering source code and using the form editor, for instance.

There are 54 example Java programs included with Windows versions of Café and more than 90 with the Macintosh version. Many of these are duplicates of the sample applets that Sun offers with the JDK or on its web site at `http://java.sun.com`.

Pricing and Additional Information

Pricing is subject to change, of course, but the most recent retail price for Café quoted on the company's web site is $299.95 for Windows users. An introductory price of $99.95 was being offered to Macintosh owners for 90 days, which was to be followed by a $299.95 price. It can be purchased on Symantec's web site in addition to retail and mail-order outlets.

Some folks might not have to buy Café at all-customers who bought Symantec C++ from December 1, 1995, to March 1, 1996, are entitled to a free upgrade to Café.

The home page for Symantec Café is the following URL:

`http://cafe.symantec.com/`

The customer service number for the company is (800) 441-7234, and its e-mail address for Java-related comments and questions is `javainfo@symantec.com`.

SunSoft Java WorkShop

SunSoft Java WorkShop, the development tool offered by the language's home team, is scheduled for fall release and might be on the market as you read this.

The IDE is written almost entirely in Java, according to its designers, and its development has been used to help improve the Java language. The mindset at Sun is that committing to such a large-scale undertaking in the company's own language gives them insight into the issues that other developers are facing, and reveals any kinks in Java that still need to be straightened out.

However, that doesn't benefit the developer looking for a tool to write software. Java WorkShop is evaluated here on the basis of its applicability to this task.

Figure 4.2 shows an example of Java WorkShop in use.

Figure 4.2 : *A screen capture of Java WorkShop.*

System Requirements

Versions of Java WorkShop are available for the following systems: Microsoft Windows 95, Windows NT 3.5.1, SPARC Solaris (2.4 or later), and Intel x86 Solaris systems.

Microsoft Windows 95 and NT systems must be running a 90-megahertz Pentium or better with 16M of memory and 45M of hard disk space. Solaris systems must have 32M of memory, 45M of disk space, and an OSF/Motif 1.2.3-compliant windowing system. The recommended display resolution to use with

Java WorkShop is 800 by 600 pixels.

Java WorkShop comes with its own modified version of the JDK, so it cannot be used in conjunction with an existing installation of the kit. Like Café, Java WorkShop needs to have any existing JDK copies deinstalled before WorkShop can be installed and run correctly.

Overview

Java WorkShop, one of the most approachable IDEs for a novice programmer, uses a web interface to offer the following features: a source editor, class browser, debugger, project management system, and Visual Java, a tool for the visual design of a graphical interface and an easier means to create windowing software.

Although still in beta release as of this writing, Java WorkShop has been available long enough to assess what kind of functionality it offers when the product hits the market. The most striking difference between it and other IDEs is the interface. Java WorkShop looks more like a web browser than a programming development environment. It is a web browser, in fact-users of Sun's HotJava browser will recognize elements from that software in the design of WorkShop and you can view any web page while in Java WorkShop.

Java WorkShop's browser interface is easier to use for programmers who are unfamiliar with IDEs and similar software and frustrating to some of those who are comfortable with these tools.

WorkShop has a source browser for viewing a class hierarchy, public methods, and variables. This creates HTML pages in the same format as HTML documentation generated by the JDK's `javadoc` utility.

The WorkShop source editor is still in an early stage of development and it lacks some of the cut-and-paste functionality of other more established editors. It works in conjunction with WorkShop's debugger-compile errors create links directly into the source editor for fixing and breakpoints can be used. The WorkShop debugger provides breakpoints and other methods of debugging.

The Visual Java feature provides a way to graphically design an interface, much like Café Studio. Visual Java enables programmers to develop the dialog boxes and other visual elements and it automatically creates event handlers for these components. There's also a menu editor and resources are saved in separate `.gui` files that can be edited later just as source files are edited.

Java WorkShop in its current release requires the use of run-time classes that make Visual Java work but developers have said that this will not be the case when the software hits the market.

The environment is not customizable in the manner of a product such as Café, but the web interface makes it easy to integrate other tools and programs into WorkShop. The program is a collection of web pages with Java programs embedded in them and around them. The user can go to a different page from within Java WorkShop as easily as a URL is entered in a web browser. This makes it possible for a user to create original pages of Java development tools that can be linked to WorkShop pages. This might be unusual for someone accustomed to a development environment that is written as a cohesive, single executable file that can't be changed, as most are. However, it suits the spirit of Java-independent programs linked together by HTML pages, which can be modified as individual elements without

affecting the other parts of the whole.

Pricing and Additional Information

When it is released, SunSoft Java WorkShop is expected to retail for $295. While it is still in beta release, the software can be downloaded freely for evaluation. For more details, and the opportunity to download a beta release, visit the following URL:

`http://www.sun.com/sunsoft/Developer-products/java/index.html`

The customer service number to use for the company is (800) 786-7638 (SUN-SOFT) in the United States, or (512) 434-1511 elsewhere; and its e-mail address for comments and questions is `sunsoft@selectnet.com`.

SourceCraft NetCraft

SourceCraft, the developer of the ObjectCraft development environment, is making its NetCraft Java IDE available as freeware. For those unfamiliar with the term, *freeware* is software that is available for no cost as long as you comply with the developer's terms and conditions for the use.

This makes NetCraft attractive if cost is a criterium, obviously, but the IDE still has to be well-designed or you pay in terms of lost time and efficiency.

Figure 4.3 shows a look at the NetCraft environment.

Figure 4.3 : *A screen capture of NetCraft*.

System Requirements

Versions of NetCraft are available for Microsoft Windows 95 and Windows NT 3.5.1 systems running a 486 or better with 8M of memory. NetCraft comes bundled with the current version of the JDK, and SourceCraft also makes it available for download without the JDK if you already have it installed.

Overview

SourceCraft NetCraft is somewhat less ambitious in its approach than other IDEs because it has a smaller set of features that are available. However, it is a fully featured replacement for the JDK, and it creates Java programs that are compatible across all Java implementations. It has an editor, a class inspector, a user interface designer, and a compiler.

NetCraft is an IDE that can be used for any type of Java applet or application. In its approach to the software, SourceCraft focuses on Java's applicability in Intranet environments.

The Package Inspector, part of NetCraft's system for organizing projects, includes a way to browse the methods used in a class. There also is a Class Inspector for looking at the following aspects of a class: its position in the hierarchy, its methods, and its variables. When you are looking at a method with this tool, you can view the source code of the method and how it is used in a program.

NetCraft, like Café and Java WorkShop, includes a way to visually develop a graphical user interface. The NetCraft UI Builder generates Java code that uses the Abstract Windowing Toolkit as its raw material, so the code does not rely on any new classes introduced with the development environment. When you create an interface component, NetCraft generates source code for that component, with a `TODO` comment line where the event-handling code for that component is placed. It's a simpler approach than some of the alternatives and a programmer comfortable with the awt should be comfortable with this approach.

The source editor uses Windows cut-and-paste commands and is similar to other small word processors that you probably are familiar with.

The NetCraft UI builder is not much more difficult to use than a word processor. Although the components in the release available at this writing have the odd habit of moving around a little when clicked, a nice feature of the builder is the ability to set the specific coordinates, height, and width of a component by entering numbers into text fields. This makes it easy to bring wandering components inline with each other.

The environment is simpler to use and master than other IDEs. However, this might be a problem in the development of sophisticated programs with numerous windows and interactions because some of the tools to manage this software are not available in NetCraft. It depends on where SourceCraft, the maker of other development tools, plans to go with this freeware product.

For basic tasks and applets, NetCraft appears to be a good substitute for JDK users seeking to migrate to a graphical interface.

Pricing and Additional Information

For more details, and the opportunity to download NetCraft at no cost, visit the home page for NetCraft at the following URL:

`http://www.sourcecraft.com:4800/about/netcraft/`

The customer service number for the company is (617) 221-5665, and its e-mail address for comments and questions is `edc@sourcecraft.com`.

Other Offerings

The following products for Java development cannot be described fully here, either because of availability or other considerations. However, they're profiled so that you can get a fuller picture of the IDEs that are available to you.

Borland C++ 5.0 with Java Enhancements

As described previously, Borland C++ 5.0 with Java Enhancements is a C++ development environment that has been extended to include Java programming tools. The advantages of this approach are multi-language development within the same environment for native method use, the ability to program in three languages (C, C++, and Java) without learning three IDEs, and software that has become robust

from several years of use by the C and C++ development community.

The home page for Borland's Internet development tools is

http://www.borland.com/internet/.

MetroWerks CodeWarrior

CodeWarrior is one of several IDEs that have been made available in pre-release or beta form for Macintosh Java development. CodeWarrior is a multi-language development environment with an introductory version called Discover Programming with Java that is intended for novices. CodeWarrior can be used to develop programs in C, C++, ObjectPascal, and Java.

The home page for Metrowerks CodeWarrior 9 is the following URL:

http://www.metrowerks.com/products/announce/cw9.html

Java WebIDE

This development environment is worth looking at to see what's being attempted-a fully web-based programming tool that doesn't require downloading. You run it off the web page over the Internet, which is how many software packages will be run as Java development matures, according to the language's adherents. WebIDE is an experiment that does not supplant a more traditional IDE at this point, offering only source creation, compilation, and syntax high-lighting in its present incarnation. However, as more tools are added it will become more interesting, and WebIDE is one of the only development environments for Java that attempts full cross-platform support. The home page is the following URL:

http://www.chamisplace.com/prog/javaide/.

Kalimantan

Before it was christened as Kalimantan, this IDE was one of several Java-related products that staked a claim to the name Espresso. The developers have been kind enough to offer links to the other Espressos, so the Kalimantan web page is a good place to sort out any Espresso confusion that you might have. Kalimantan is another cross-platform IDE and it has been tested for use with Solaris 2.4 and up, and Windows 95 systems. Kalimantan's current beta release includes only an inspector to look at the values of internal variables and a debugger, but it is bundled with the `teikade` suite of utilities from PFU Limited. This suite includes a class browser that is familiar to those who have used class browsers with the Smalltalk programming language. The home page for Kalimantan is the following URL:

http://www.real-time.com/java/kalimantan/index.html

Natural Intelligence Roaster

Roaster was made available to developers in January 1996, making it the first Java IDE for the Macintosh. The current version at this writing is Developer Release 2.1. The Roaster Professional Edition includes a visual interface builder, compilation that can be targeted for Macintosh or Microsoft Windows systems, and an extended class library. The Sams.net book *Teach Yourself Java for the Macintosh in 21 Days* was written for the Roaster environment. Details on Roaster are available from the

following URL:

http://www.natural.com/pages/products/roaster/index.html

Microsoft Visual J++

It is just becoming available in beta release as of this writing, but the company developing Visual J++ makes the product worth keeping an eye on. Visual J++ is Microsoft's machine-proprietary answer to Java development, and it features extensions to the Java class library that are specific to Microsoft's operating system. It integrates Java with the component object model (COM) integration through Microsoft ActiveX, and is integrated with the Internet Explorer browser that implemented Java with its 3.0 release.

There isn't a home page for this software yet on Microsoft's site, but the company's SiteBuilder Network, which provides an opportunity to download the beta release of Visual J++, is at the following URL:

http://www.microsoft.com/sitebuilder/

RogueWave JFactory

JFactory is an interface builder rather than a full IDE. It is used in conjunction with any editor and Java compiler. JFactory is a sophisticated interface builder with the capability to generate a lot of the support code that is associated with interface components, and it can import `.rc` and `.dlg` files created with other programming environments. The JFactory web site is at the following URL:

http://www.roguewave.com/products/jfactory/jfactory.html

Cosmo Code

Cosmo Code is an integrated development environment for Java that runs on Silicon Graphics IRIX operating system versions 5.3 and 6.2. There is a compiler, interpreter, debugger, and a class browser. The compiler can create machine-independent code, symmetric multiprocessing on SGI systems, and the creation of executable native code files. The Cosmo Code home page is at the following URL:

http://www.sgi.com/Products/cosmo/code/index.html

Summary

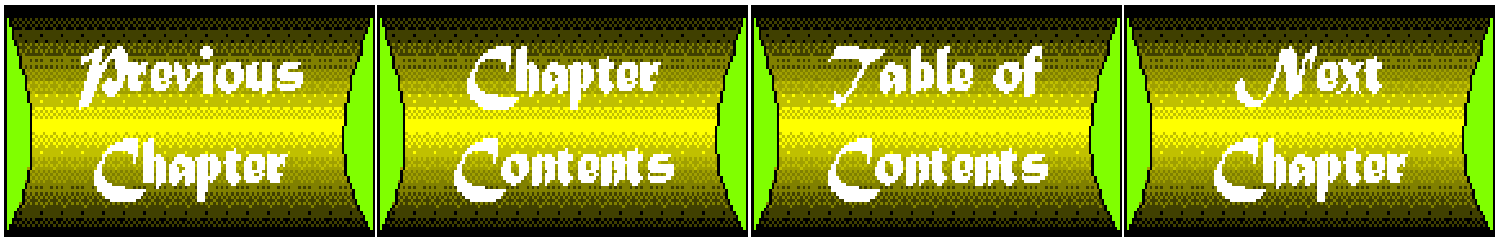
As you probably have discovered by this point, the decision about which IDE to use depends on your programming experience, personal taste, and the tasks you need to accomplish with your software creations.

Because developments occur at such a rapid pace in regard to Java, a trend that will continue for the foreseeable future, it is worthwhile to use the Internet to keep up with changes. As stated previously, one web site has been established to offer the latest news on IDEs, links to reviews, and a full listing of announced software. It's at the following URL:

http://www.cybercom.net/~frog/javaide.html#reviews

The Java newsgroups such as `comp.lang.java`, `comp.lang.java.misc` and `comp.lang.java.programmer` are another way to get a range of user opinions on the IDEs you are considering. Many IDE developers also participate in these forums, including the folks who created Café and Java WorkShop.

Most of the IDEs that have not reached the market can be downloaded as beta versions over the World Wide Web. You ought to make use of this before you choose a development environment. It's a hassle to install the software and deinstall its rivals, but even if the IDE doesn't suit your needs, it gives you a much better idea about what you do need in a development environment.



Chapter 5

Intranet Security

CONTENTS

- Introduction
- Why Security?
  - What Are the Security Features of an Intranet?
  - It's Your Call
- Security on Your Web Server
  - Controlling Access Globally and Locally
  - Username/Password Authentication
  - Authentication Based on Network Hostname or Address
  - Combined Authentication
- Secure/Encrypted Transactions
  - Secure HTTP (S-HTTP)
  - Secure Sockets Layer (SSL)
- The Common Gateway Interface (CGI) and Intranet Security
- Your Intranet and the Internet
  - Firewalls
  - Virtual Intranet
- Summary

---

security \si-'kyur- t-e-\ n: *the quality or state of being secure*

Introduction

You might think that there is little reason to be concerned about security in an intranet. After all, by definition an intranet is *internal* to your organization; outsiders can't access it. There are strong arguments for the position that an intranet should be completely open to its users, with little or no security. You might not have considered your intranet in any other light.

On the other hand, implementing some simple, built-in security measures in your intranet can allow you to provide resources you might not have considered possible in such a context. For example, you can give access to some Web pages to some people without making them available to your entire customer base, with several kinds of authentication. In this chapter, you'll learn how simple security measures can be used to widen the scope of your intranet.

To help you get oriented to the material to be presented, the following is a list of chapter objectives. You might want to refer to this list as you work your way through the chapter. In this chapter, you'll

- Consider the overall security aspects of your intranet.
- Learn how implementing security on your intranet can actually broaden the ways in which it can be useful in your organization.
- Learn how to set up username/password authentication to limit access to resources on your intranet.
- Learn how to provide secure access to intranet resources to groups of customers.
- Learn how to restrict access to sensitive resources based on customers' computer hostnames or network addresses.
- Learn about the security aspects of CGI-BIN scripting.
- Learn about using encrypted data transmission on your intranet to protect critical information.
- Learn important information about securing access to your intranet when your corporate network is attached to the Internet.
- Learn how to provide-and limit-secure access to your intranet from outside your immediate local network.

Intranet security is, then, a multifaceted issue, with both opportunities and dangers, especially if your network is part of the Internet. I'll walk through the major ones, with detailed information on using built-in intranet security features, in this chapter.

Warning
Except in the sections of this chapter that are specifically devoted to Internet security issues, it's assumed that your intranet is *not* accessible from outside your organization. If you are on the Internet, the intranet security measures discussed in this chapter may not be sufficient to secure your system. If you want to make the services and resources of your intranet accessible from the outside, you'll need to take significant additional steps to prevent abuse and unauthorized access. Some of these steps are described at the end of this chapter in the section titled "Your Intranet and the Internet."

Why Security?

Many people view computer and network security in a negative light, thinking of it only in terms of restricting access to services. One major view of network security is "that which is not expressly permitted is denied." Although this view is a good way of thinking about how to connect your organization to the Internet, you can, and possibly should, view intranet security from a more positive angle. Properly set up, intranet security can be an *enabler*, enriching your intranet with services and resources you would not otherwise be able to provide. Such an overall security policy might be described as "that which is not expressly denied is permitted."

This does not mean that you should throw caution to the wind and make everything available to your users on your intranet. There are many things to consider when placing sensitive business data out on your intranet. It may fall into the wrong hands, or worse, be used against your business.

This chapter takes the latter approach, presenting intranet security in terms of its opportunities for adding value to your intranet. For example, some of your users might have information they would like to make available, provided access to it can be limited to a specified group-for example, confidential management or financial information. Without the ability to ensure that only those who have the right to see such information will have access, the custodians of such data

will not be willing to put it on your intranet. Providing security increases your organization's ability to use the important collaborative aspects of an intranet.

For example, your company's accounting department wants to publish a weekly list of the top ten delinquent clients and the amounts they owe. They've hired a young stud programmer who has created a link to the company database that updates this information automatically. While this information is useful to upper management, and perhaps the sales staff, it shouldn't be viewed by other departments. The fact that a client is delinquent in payment can cause your employees to think less of those clients and perhaps cop a bad attitude toward them.

The more defensive approach, preventing abuse of your intranet, is also given play, however. Organizations' needs for security in an intranet can vary widely. Businesses in which con-fidentiality and discretion are the norm in handling proprietary information and corporate intellectual property have different needs than a college or university, for example. Academic institutions generally tilt toward making the free exchange of ideas a primary interest. At the same time, though, the curiosity (to use a polite word) of undergraduates requires strong needs for security. Keeping prying sophomores out of university administration computing resources is a high priority; for example, students have been known to try to access grade records (their own or those of others) for various reasons. Even simple adolescent high jinks take on new dimensions on a computer network.

What Are the Security Features of an Intranet?

Before going into a great deal of detail about how you can use security to enhance your intranet, take a high-level look at what security features are available to you. These break down into three main categories. First, you can take steps on your Web server to set up security. Second, you can take steps with the other TCP/IP network services you've set up on your intranet to enhance their security. Third, you can secure customers' Web browsers themselves to limit what they can do with them.

Web Server Security

There is a wide range of very flexible security features you can implement on your Web server. Here's a summary:

- Access to Web servers, individual Web pages, and entire directories containing Web pages can be set to require a username and password.
- Access to Web servers, individual Web pages, and entire directories containing Web pages can be limited to customers on specific computer systems. (In other words, access will be denied unless the user is at his or her usual computer or workstation.)
- You can organize individuals into groups and grant access to individual Web servers, Web pages, and entire directories containing Web pages based on group membership.
- You can organize computers into groups, and grant access to individual Web servers, Web pages, and entire directories containing Web pages based on group membership.
- CGI-BIN scripts on your Web server can use any of the above access restrictions, though you must take care in writing them to ensure you don't make security-related mistakes.
- Some httpd server software is capable of communicating with compatible Web browsers in a secure, encrypted fashion, defeating even network-level sniffers and ensuring confidential data transmission across your intranet.

You can combine these features in a number of ways, such as requiring a password and limiting access to a group of users who must access your Web server from a specific group of computer systems. You'll see a good deal of detail about Web server security setup in this chapter.

Security in Other Intranet Applications

In addition to the access controls you can set up on your Web servers, you can implement security in some of the other network services that may be offered on your intranet. Here are some of the steps you can take:

- Access to your anonymous FTP server can be limited in several important ways, much like with your HTTP server, while still enabling authorized customers to upload files to it.
- Access to your Usenet news server can be limited in much the same way.
- Access to searchable intranet indices and databases can be controlled through password-protected Web interfaces.
- Access to Gopher services can be controlled based on TCP/IP network address, and separate browse, read, and search permissions can be set on a per-directory basis.

This chapter, doesn't provide any additional information about these services. You'll want to refer to the documentation for these network packages to learn about how to handle access control and other security features in them.

Securing Users' Web Browsers

Some Web browsers can be set up in *kiosk mode*, which limits the features of the package that users can access. Available primarily in ncSA Windows Mosaic and Mosaic-based browsers, kiosk mode runs the browser with a limited set of features. Users cannot save, print, or view the HTML source of Web pages, and hotlist/bookmark editing is not allowed. The user cannot even exit from the browser and restart it in normal mode without exiting from Windows altogether. Even the overall Mosaic window cannot be minimized or maximized, and the normal pull-down control menu for Windows is missing. This can be quite effective in a controlled environment. However, end users tend to find their way out of paper bags quite often. Don't count on kiosk mode being your super security implementation.

Figure 5.1 shows ncSA Mosaic for Windows in kiosk mode, and, for your comparison, Figure 5.2 shows the same page in standard Mosaic. As you can see, many of the normal toolbar buttons are missing, as is the Options menu. The remaining pull-down menus are also limited in the available features. Kiosk mode is primarily for use in library or trade-show environments, where users need to be limited in what they can do, but you might find a use for it in your intranet if you need to limit what some customers can do with the package. The Netscape Navigator browser does not have a kiosk mode.

Figure 5.1 : *ncSA Mosaic for Windows in kiosk mode*.

Figure 5.2 : *ncSA Mosaic for Windows in normal mode.*

Tip
Resourceful users will quickly figure out they can manually edit their pc's `autoexec.bat` file or Web browser's `.ini` file to override kiosk mode, undoing the limitations you've placed on them. If you're concerned about such things, you'll need to place user startup and Windows and browser setup files on a file server to which users have read permission only. You'll also need to limit access to the Mosaic startup command itself, or else users would simply use the Windows Program Manager's Run command to start another Mosaic session. As a result, kiosk mode might not be worth your trouble except in limited situations, such as at a trade show.

It's Your Call

It's your responsibility to determine the level of security you need on your intranet, and, of course, to implement it. Putting most of the security measures mentioned into place, as you'll learn in the following sections, is not difficult. Your primary concern will be explaining to customers how intranet security works, not so much as a limiting factor but as an opportunity for increased use and collaboration using your intranet. Assuring decision-makers that they can make information available on your intranet in a secure fashion can go a long way toward making your intranet a success. At the same time, it's important to

make sure both information providers and their customers understand a number of critical aspects of intranet security, so they don't inadvertently defeat the purpose of it.

There are network security commonplaces, unrelated to intranet security specifically, that need your attention. All the security precautions in the world can't protect your intranet from overall poor security practices. Users making poor choices on passwords always leads the list of computer and network security risks. You can limit access to a sensitive Web resource based on the TCP/IP network address of the boss's pc, but if the boss walks away and leaves his pc unattended without an active screenlock, anyone who walks into the empty office can access the protected resources.

Caution
Password security is only as good as the passwords that are chosen. Be sure to impose some sort of password setting and changing policy. This will save you time in the end. Unique passwords that are a garble of letters and numbers are the best, but the hardest to remember. Encourage your users to be creative in their password selection. Sometimes a user uses his own name as his password, or his significant other's or pet's name; password-guessing is simple for anyone who knows him. Some people write their passwords down and tape them to their keyboards or monitors. These bad habits need to be avoided to completely secure your intranet.

In other words, the same good security practices that should be followed in any networked computing environment should also be made to apply in your intranet. Not doing so negates all the possible security steps you can take and reduces the value of your intranet. Even in the absence of malice, the failure to maintain any security on your intranet will inevitably result in an intranet with little real utility and value to its customers.

Security on Your Web Server

It's useful to break the overall subject of World Wide Web server security down into three pieces and discuss them separately. I'll do so in this section, covering user/password authentication, network address access limitations, and transaction encryption. Bear in mind throughout the discussion of these separate pieces that you can combine them in various ways to create flexible and fine-grained access control. In fact, combining two, or even all three, of these methods provides the best overall security.

Controlling Access Globally and Locally

Before I turn to the individual methods, I'll cover some high-level information about Web server security setup.

Whichever individual security mechanisms you implement on your Web server, the first thing you need to know is that you can implement them at either or both of two levels. First, you can specify high-level access control in a Global Access Configuration File (GACF), specifying overall access rules for your server. In the ncSA httpd server and those which are derived from it, such as the Windows httpd and Apache servers, the GACF is called `access.conf`. The CERN/W3 server doesn't have a separate GACF; rather, all access control information is in the main server configuration file, `httpd.conf`. The Netscape servers have a graphical interface (actually, Netscape Navigator itself) for overall server administration, including setting up access control. If you feel more comfortable editing configuration files, the Netscape server does allow them, calling them Dynamic Configuration Files. Although you can do both global and local configuration using the graphical tool, you can also manually create a top-level Netscape Dynamic Configuration File, which can then be hand-edited to function as a GACF.

Second, you can set up per-directory access control using local ACFs (LACFs) for each directory or subdirectory tree. Usually named `.htaccess` or `.www_acl` (note the leading periods in the filenames), LACFs lay out access control for an individual directory and its subdirectories, although subdirectories

can also have their own LACFs. The CERN/W3 server can even extend protection to the individual file level using LACFs. In the Netscape server, lower-level Dynamic Configuration Files serve as LACFs. You can change the names of LACFs in both the ncSA and Netscape servers, but you're stuck with `.www_acl` in CERN/W3.

With a few important exceptions, you can do everything with an LACF you can do with a GACF. Although you can control access to every directory in your Web server document tree from the GACF, you'll probably not want to do so, especially if your needs for access control are complex. It's easy to make mistakes in a lengthy configuration file like the GACF, and you'll get unexpected, unintended results when you do. These might be hard to track down and might not even show up without extensive testing. Overall, it's better to use your GACF to establish a high-level security policy and then set up lower-level, simpler controls using LACFs.

Note
The CERN/W3 server's LACF files have a completely different format than its GACF. Most of the examples in this chapter apply only to the format of the GACF.

What's the GACF for, then? Most Webmasters use the GACF to establish a general access policy for their Web server. For example, if your Web server is accessible to the Internet at large and you're not using a firewall system (see the "Firewalls" section later in this chapter) to limit access to your network from the outside, you may want to establish a policy in your GACF that only computers with TCP/IP network addresses that are inside your network can access your Web server's document tree. Similarly, you can use the GACF to segregate public and private areas on your Web server according to some criteria, and require usernames and passwords for access to the private areas.

After you've established your overall policies, you can implement LACFs to fine-tune your setup. In doing so, you can selectively apply different access controls to the directory or directories controlled by the LACF.

Earlier, exceptions to the statement that you can do everything with an LACF you can do with a GACF were mentioned. Here is a quick, incomplete list; you'll want to consult detailed server documentation for comprehensive explanations of these and others. The first one applies to all httpd servers, and the last three refer only to UNIX servers.

- If you want to control all access on your Web server with your GACF, you can use it to prohibit the use of LACFs altogether.
- You can deny use of a potentially dangerous and CPU-hogging feature called server-side includes, which actually cause the server to execute outside commands each time a page containing them is accessed, in user Web pages.
- You can limit access to CGI-BIN scripts in the server's main CGI-BIN directory, preventing users from creating potentially dangerous ones in their own Web direc-
  tories.
- You can prevent potential security problems that can come from following UNIX symbolic links.

With respect to symbolic links, confidential files on the system that are completely outside of your Web server tree could be compromised by a naive or malicious user. For example, if a user created a symbolic link in her home directory pointing to the UNIX `/etc/passwd` file, which contains usernames and encrypted passwords, outside users could obtain a copy of that file using their Web browser and then run a password-cracker on it offline. Of course, a malicious user can grab `/etc/passwd` himself and run the cracker directly, or e-mail the file to someone else for the same reason, but that's no reason to make it easy to do so via your intranet. (The UNIX System V `/etc/shadow` file is not readable by non-root users, nor is the IBM AIX `/etc/security/passwd` file.) See the section titled "The Common Gateway Interface (CGI) and Intranet Security" later in this chapter for discussion of CGI-BIN and server-side include security issues.

These generalities out of the way, let's turn our attention to the three major elements of Web server security.

Username/Password Authentication

The first major element of Web server security is username/password authentication. All the sample Web servers discussed in this book provide this basic kind of security. I kick off this discussion by looking at what the Web browser user sees when he encounters a Web page that requires username/password authentication for access. Figure 5.3 (part of ncSA's excellent access control tutorial, at `http://hoohoo.ncsa.uiuc.edu/docs/tutorials/`) shows a Prompt dialog box asking for a username. Once the username is entered, a new dialog box asks for a password, as shown in Figure 5.4.

Figure 5.3 : *The user is prompted to enter a username on a protected Web page.*

As you can infer from Figures 5.3 and 5.4, there are three aspects of username/password authentication: the *username*, the *password* that applies to that username, and what is permitted to that user when a correct username and password are supplied. Usernames and passwords are meaningless unless you specify a directory, directory tree, or filename to which your username/password access restrictions apply.

Figure 5.4 : *The user is also prompted for a password on a protected Web page.*

To make this more understandable, look at an example. Suppose your httpd server's `DocumentRoot` directory contains three main subdirectories, named `public`, `management`, and `personnel`. Using your GACF, you can specify that access to the `management` and `personnel` subdirectory trees requires username/password authentication, while `public` is left wide open for anyone to access without being prompted for a username and password. You can also set up LACFs within the protected subdirectories to further limit access to particularly sensitive documents by using usernames/passwords.

Setting Up Username/Password Authentication in a Netscape Server

Of the servers covered in this book, setting up username/password authentication is simplest in the Netscape servers. The Netscape servers actually use the Netscape browser itself as a graphical interface for administering the server, providing a set of private Web pages and configuration scripts to do so. Using the Server Manager page, you can easily enter new users and their passwords into what Netscape calls the user database. (Your empty user database must first be created before you can add any users and passwords to it.) Figure 5.5 shows the User Database Management screen. Notice there's an administrative password that must be given, as well as the username and password for the user being added to the database. After you've entered this information, click Make These Changes.

Figure 5.5 : *The Netscape Communications Server Add-User form.*

After you've set up one or more users, you can continue to use the Server Manager to apply access control rules to users. You can associate groups of users together for purposes of authentication, and define access control rules that apply to groups as well as to individuals. With group access controls, users must still provide their own usernames and passwords, but access to a specified area of the server file tree (Netscape calls this a *realm*) can be controlled by requiring that a user be a member of a group for access. Even if a user provides the correct username and password, he may be denied access based on group access control rules if he is not a member of that realm's group.

Setting Up Username/Password Authentication in the CERN/W3 httpd Server

The CERN/W3 httpd server uses a UNIX-like password file (but with only three colon-separated fields) containing usernames, encrypted passwords, and users' real names. The password file is controlled using the `htadm` program that comes with the httpd server software. This program enables you to create and delete user accounts, as well as change and verify existing passwords. Although you can provide all the information `htadm` needs on the command line, it's easier to let the program prompt you for it. For example, to add a new username and password:

```
# path/to/htadm /path/to/passwordfile
```

You must specify the name of the password file on the command line, but `htadm` will prompt you for the function you want to perform and the actual username, password, and user's real name, as appropriate. (You can use multiple words in the `realname` field to include a full name and other information.) If you're in a hurry or have a long list of users to add or delete, you can take advantage of specifying all the `htadm` command-line arguments at once, like in these examples:

```
# htadm -adduser passwordfile joeuser joespassword Joe User
# htadm -deluser passwordfile baduser
```

The first example creates the user `joeuser` with the password `joespassword`, and the second deletes the user `baduser`. This enables you to do mass account deletion using shell looping and to take username, password, and real name input from a file. You'll need something like the `expect` package to do automated, mass account creation. There are also `-passwd` and `-check` command-line arguments to `htadm`, which enable you to change and verify passwords, respectively.

As with the Netscape server, the CERN/W3 httpd server also enables you to associate individual users with groups. You can set up group authentication rules in LACFs that control access to portions of your Web server document tree. CERN/W3 uses a group file, the format of which is based on the standard UNIX `/etc/group` format, but it has an added feature for defining access control rules and for recursive inclusion of groups into metagroups. A simple group file, which I'll use for examples in this chapter, might be something like this:

```
management: tom, mary, joan
personnel: anne, joe, jerry
staff: management, personnel
public: All, Anybody
```

Here, four groups are defined. The first two each contain a list of several individual usernames, but the last two are groups of groups. (Two special groups, *All*, meaning all authenticated users and *Anybody* meaning anyone, authenticated or not, are predefined by the CERN/W3 httpd server software, and refer to anyone who might access the server; see the server documentation for details on the distinction between All and Anybody.)

After setting up your password and group files, you can add access control protection to your server. As noted, high-level rules go in `httpd.conf`, the GACF. Access control rules in `httpd.conf` use the `Protect` directive and associated protection rulesets. Here's a simple `Protect` directive, based on the group file shown above. It implements the example division of your Web server's document tree into `public`, `management`, and `personnel` subtrees:

```
Protect /personnel/* Personnel
```

This example indicates that all subdirectories and files in the `personnel` subtrees of your Web server `DocumentRoot` are subject to the rules in the protection ruleset named `Personnel`. (You can name protection rulesets with any name you want, but it makes sense to use meaningful names.) According to this `Protect` directive, the ruleset itself also appears in the GACF, under the label `Personnel`, and might look like this:

```
Protection Personnel {
     AuthType     Basic
     Passwordfile /usr/local/etc/httpd/passwd
     GroupFile    /usr/local/etc/httpd/group
     GetMask      personnel
}
```

Note

> The `Protect` directive can specify the protection ruleset be read from a file, rather than from another part of the GACF. In this case, the directive would look like this:
> `Protect /personnel/* /usr/local/etc/httpd/acls/Personnel`
> Here, the absolute pathname to the file named `Personnel` (not relative to the server `DocumentRoot`) is specified. This example assumes you've created a special subdirectory (`/usr/local/etc/httpd/acls`) in which to store all your access control information. If you use individual files like this to define your protection rulesets, you need not enter the curly braces that are required in the GACF.

This simple example applies username/password authentication access control to all files and subdirectories in the `personnel` directory, using the following criteria, all of which must be met before access is granted:

- Users must enter a username and password.
- Usernames and passwords are validated against the file `/usr/local/etc/httpd/passwd`.
- Authenticated usernames are checked for membership in the group named `personnel` in the groups file `/usr/local/etc/httpd/group`.

Going back to the sample group file, you can see that only users Anne, Joe, and Jerry will be granted access to files in this directory tree. Even if Tom provides his correct password, he will not be given access.

This has been a very cursory look at user authentication in the CERN/W3 httpd server. `Protect` directives and protection rulesets can be quite detailed, including other features not described here. In addition, you can set up both a default protection ruleset and progressively more limited protection rulesets according to your own criteria, adding access control all the way down to the individual file level. For details, see the documentation which comes with the CERN/W3 httpd server software (on the *Developing Intranet Applications with Java* CD-ROM). Or check out the World Wide Web Consortium's online CERN/W3 httpd documentation at `http://www.w3.org/pub/WWW/Daemon/User/Admin.html`.

Setting Up Username/Password Authentication in the ncSA httpd Server

The ncSA httpd server, along with those derived from it (WinHttpd for Windows and the Apache package for UNIX systems), provide similar username/password authentication mechanisms. Except where there are differences among these packages, I'll discuss them as a group.

Although these packages use authentication methods that are similar to the methods used in the CERN/W3 httpd package, there are differences. Let's first focus on the similarities. Most importantly, the ncSA packages support both GACFs and LACFs, enabling you to set high-level policy at the server level and then fine-tune it at the directory and subdirectory levels. In addition, both individual user and group authentication are provided for. Finally, some configuration commands, such as `AllowOverride`, affecting critical items listed earlier might only appear in a GACF. You can also disallow the use of server-side includes and the following of symbolic links, for example, as described earlier.

The GACF in the ncSA packages is the file named `access.conf` (`access.cnf` in WinHttpd) and is located in the `conf` subdirectory of your Web server's file tree. On UNIX systems, the server is usually installed in `/usr/local/etc/httpd`, and on Windows systems, the server is in `c:\httpd`. In both cases, there exists a `conf` subdirectory in the top-level httpd directory. However, the layout and syntax of the `access.conf` file is significantly different from the GACF in the CERN/W3 httpd server.

The ncSA file is divided into sections, one for each directory to be controlled. Each directory section in `access.conf` looks something like this:

```
<Directory /absolute/directory/path>
[ Various configuration commands ]
</Directory>
```

Like HTML markup, each `Directory` (the literal word `Directory` must appear) section is marked off by the `access.conf` tags `<Directory>` and `</Directory>`, surrounded with angle brackets. Case is not significant in the word `Directory`, although it might be in the actual directory name.

The directory path here is an absolute pathname and is not relative to either the Web server's `ServerRoot` or `DocumentRoot` directories. If you use `/usr/local/etc/htdocs/`, for example, you must specify it in full and not just simply use `/htdocs`. Within each `Directory` section of the file, you specify one or more options, or configuration commands, which will be applied by the server to the specified directory. There are a number of different options, but we're concerned here with username/password authentication.

Of course, before you can apply a username/password access control, you need to have established users and passwords on your server. Usernames and encrypted passwords are stored in a special httpd password file. ncSA provides a utility program, `htpasswd`, for creating this file; you'll find it in the `support` subdirectory of your ncSA httpd server file tree, and you might need to compile it. The syntax of the `htpasswd` command is substantially simpler than that of the CERN/W3 `htadm` command, as are its capabilities. To add a user to your password file or change his password, use this syntax:

```
# htpasswd /path/to/passwordfile username
```

If you don't already have a password file, you need to modify this command a bit:

```
# htpasswd -c /path/to/passwordfile username
```

The `-c` argument creates a new password file, so you use it only once. If you use it again, you'll erase your current password file. You can name your password file anything you like.

You can't remove a user from your password file with the `htpasswd` command. Instead, you'll have to hand-edit the password file with a text editor and delete the user's entry. The format of the file is quite simple, with just two fields in each record, separated by a colon:

```
tkevans:TyWhfX9/zYd7Y
```

The first field is the username. The second field is the encrypted password. Permissions on the password file must be set so as to be readable by the system user under whose userid the httpd server runs (usually, the no-privileges user `nobody`), so passwords are not stored in clear text.

Besides the httpd password file, the ncSA servers also respect a group file in which you can define groups of users. Groups can be treated like individual users with respect to access control, so the group file can add capabilities and save data-entry time. For the most part, syntax of the ncSA httpd group file is exactly the same as that shown earlier in this chapter for the CERN/W3 group file.

There is one significant difference in what the two group files may contain, however. As noted above, the CERN/W33 group file can include group entries which consist of other groups. The ncSA group file can include only individual users as members of groups. Thus, the recursive staff group, consisting of all the members of the personnel and management groups, is not possible in ncSA. To create such a group, you would need to re-enter each user's name in the group entry for `staff`.

Now that you've set up your password and group files, you're ready to add username/password authentication in your GACF or LACFs. Take a look at an example:

```
# Anybody in the personnel group can get to the top level
# of the personnel filetree
<Directory /usr/local/web-docs/personnel>
AuthType Basic
AuthName Personnel Only
AuthUserFile /usr/local/etc/httpd/userpw
AuthGroupFile /usr/local/etc/httpd/ourgroup
```

```
<Limit GET>
Require group personnel
</Limit>
</Directory>
```

Here, in the GACF file, you've limited access to the top level of the `personnel` tree of the Web server. Only members of the predefined group `personnel` (defined in the `ourgroup` file) are allowed to `GET` (access) files in the directory tree, and they must provide a valid username and password, verifiable against the encrypted password in the `userpw` file.

Most of the lines in the example are clear, but a couple need a little more explanation. `AuthName` is just an arbitrary label for your rule; you should put something there that'll make sense when you read the rule a year from now, and you can use a phrase here. The `<Limit GET>` subsection of the file is the critical section, in which you actually specify who has access. You can also include comments in the file, as indicated by the first two lines, where the # symbol is used.

As I've noted, you can use LACFs to refine the access rules in your GACF. Here's an example of an ncSA httpd LACF: a file named `.htacces` in the `personnel/executive` subdirectory. See if you can translate its meaning:

```
AuthType Basic
AuthName Anne Only
AuthUserFile /usr/local/etc/httpd/userpw
AuthGroupFile /usr/local/etc/httpd/ourgroup
<Limit GET>
Require user anne
</Limit>
```

You're right; this rule limits access to the `executive` subdirectory to a single user: `anne`. The heart of this rule is the matter between the `<Limit>` and `</Limit>` tags near the end of the file. Other users, including the other members of the `personnel` group, are denied access, even if they give a correct password for themselves. A dialog box will demand Anne's username and password. Notice that this LACF file, which controls access to a single directory (`personnel/executive`), does not require the opening and closing `<Directory>` and `</Directory>` tags required in the server's GACF because there are no subdirectories in this directory.

Important Warnings About Username/Password Authentication

Unless the access rules change (that is, new LACFs are encountered) as a user moves around on your intranet Web pages (as with the `personnel/executive` subdirectory in the previous example), he will be prompted only once in his browser session for a username and password. As long as he continues his browser session, he can access all of the files and directories available to him under the most recent access rule-without being prompted again for his password. This is for the sake of convenience; customers shouldn't have to repeatedly provide their usernames and passwords at each step of the way when the access rule hasn't changed.

However, this situation has important ramifications if you follow it logically. Suppose Anne, having authenticated herself to access the `executive` subdirectory, leaves her Netscape or Mosaic session running, as most of us do. Her privileged access remains open to all the files protected by that one-time, possibly days-old, authentication. If she leaves her workstation, pc, or terminal unattended when she goes to lunch or goes home for the day, without any sort of active screen or office door lock, anyone can sit down and browse the files and directories that are supposed to be limited to Anne's eyes only. This is a potential security breach, and one that the Webmaster can do little about. This is really no different from a user who leaves his workstation unattended without logging off. Although you can try to educate your customers about such everyday security matters, even though they have very little to do with your intranet, you'll agree a security breach like this can be potentially harmful to all your work.

User passwords are transmitted over your network by most Web browsers in a relatively insecure fashion. It is not terribly difficult for a user with a network snooper running to pick out the httpd network packets containing user passwords. Although the passwords are not transmitted in clear text, the encoding/encryption method is a very old and widely used one. Every UNIX system, for example, has a program (`uudecode`) that can decode the encrypted password in a captured httpd packet. If you believe this may be a problem on your intranet, you'll want to consider the secure Web servers and browsers that encrypt user-transmitted data, as discussed in the section titled "Secure/Encrypted Transactions," later in this chapter.

Authentication Based on Network Hostname or Address

All the Web servers discussed in this chapter provide an additional authentication method, using the TCP/IP hostname or numerical network address of customer workstations or pcs as access criteria. As you'll learn in later chapters, in the context of CGI-BIN programming, every Web browser request for a document or other intranet resource contains the numerical IP address of the requesting computer. Servers look up hostnames using these addresses and the Domain Name Service (DNS). You can set up rules in your GACFs and LACFs based on either of these, making a considerable amount of fine-tuning possible.

Hostname/Address Authentication in the ncSA Servers

Because the format of the ncSA `access.conf` file is still fresh in your mind from the last section, look at this one first in the context of hostname/network address authentication. You'll place your rules for this sort of authentication within the `<Limit>` and `</Limit>` tags of the server's GACFs or LACFs `<Directory>` sections. Do this with several new access control directives, including

- `Order`, which specifies the order in which the other directives in the file are to be evaluated.
- `Allow`, which permits access based on a hostname or IP address.
- `Deny`, which denies access based on a hostname or IP address.

Here's a simple example limiting access to the `personnel` subtrees of your Web server. (The opening and closing `<Directory>` tags have been left off so as to cut right to the chase.) For purposes of this example, I'll assume your company's TCP/IP network domain is subdivided along operational lines and that there is a `personnel` subdomain in which all of the computers have IP addresses beginning with `123.45.67`.

```
<Limit GET>
order deny,allow
deny from all
allow from personnel.mycompany.com
allow from 123.45.67
</Limit>
```

In plain English, this example rule says, "access is denied to all hostnames and IP addresses *except* those in the subdomain `personnel.mycompany.com` and those in the numerical IP address family `123.45.67`." Notice that both the subdomain name and IP address family are wildcards that might match many computers; you can also use individual hostnames or addresses for even finer-grained control.

As you can see, I've used each of the three directives listed. You might wonder why I used both `allow` and `deny` statements. The World Wide Web was built with openness in mind, not security. The server therefore assumes, without instructions to the contrary, all directories are accessible to all hostnames/addresses. (This is the same as the username/password authentication about which you learned earlier. In the absence of a username/password requirement, all directories and files are accessible to all users.) Without a `deny` directive, the rule might just as well not exist. The server assumes, in the absence of a `deny` directive, all hostnames/addresses are allowed access. Why have any rule at all, then, if all are allowed access? In other words, it makes no sense to have rules with `allow` directives that don't have `deny` directives.

Because you must have both `deny` and `allow` directives in order to have meaningful access rules, the order in which the rules are evaluated is important. One

way to evaluate your implementation is to follow the actual order in which the directives appear in the file, but it's easy to make mistakes with this approach. Instead, ncSA httpd uses the `order` directive so you can explicitly instruct that your directives be processed in the order you want. The example uses `order deny,access`, indicating all incoming requests are to be tested against the `deny` directives first and then tested against the `allow` directives. In the example, you set up a general `deny` rule and then make exceptions to it. The `order` directive can also be turned around, with `allow` rules processed first. Using this sequence, you can make your server generally available and then add selective denials. For example:

```
<Limit GET>
order allow,deny
allow from all
deny from .mycompetitor.com
</Limit>
```

Here, you're granting access to your server to everyone *except* your competitor. For more information about hostname/IP address authentication, see the ncSA httpd server documentation on the *Developing Intranet Applications with Java* CD-ROM, or the authentication tutorial at ncSA's Web site, http://hoohoo.ncsa.uiuc.edu/docs/tutorials/.

Hostname/Address Authentication in the CERN/W3 Server

You can also impose hostname/IP address access control with the CERN/W3 httpd server. Although you can accomplish the same ends as with the ncSA server, the method of doing so is different, and the access control file formats are different. As you'll recall from the earlier username/password authentication, the CERN/W3 httpd server uses protection rulesets in the GACF or LACF. I'll modify the earlier example in which you limited access to the `personnel` portion of your Web server by group name to illustrate hostname/IP address authentication. For purposes of this example, I'll assume that your company's TCP/IP network domain is subdivided along operational lines and that there is a `personnel` subdomain, all of the computers in which have IP addresses beginning with `123.45.67`.

```
Protection Personnel {
     AuthType     Basic
     Passwordfile /usr/local/etc/httpd/passwd
     GroupFile    /usr/local/etc/httpd/group
     GetMask      @*.personnel.mycompany.com,@123.45.67.*
}
```

As you can see, the only thing changed about this ruleset is the `GetMask` line. In the earlier example, I used `GetMask` to limit access based on membership in a defined group of usernames, `personnel`. Here, I've done access control limitation in two ways. First, I specified an sub-domain name (`personnel.mycompany.com`). Second, the rule contains a numerical IP address family. In both cases, I've used a special wildcard syntax; note the use of both the `@` symbol and the asterisk (`*`). You can think of the string `@*.personnel.mycompany.com` as meaning any user at any computer in the `personnel` subdomain. Similarly, `@123.45.67.*` refers to any user at any computer with an IP address beginning with `123.45.67`.

Actually it was not needed. You might be wondering why, since all computers in the `personnel` subdomain have IP addresses in the `123.45.67` family, I've included both rules. I did this for a couple of reasons. The first is to show that you can use either symbolic host/domain/subdomain names or numerical IP addresses.

The second reason is a more technical one. In some cases, your httpd server won't be able to resolve the hostname of a computer making a request for a document from the numerical IP address it receives in the browser request. The reasons for this inability vary, but they usually involve out-of-date or inaccurate DNS information. In growing networks, newly networked computers might not get added to the database promptly. Errors in DNS configuration, such as misspelled hostnames, can also result in unresolvable hostnames. To be safe, placing both symbolic host/domain name and numerical IP address information in

your `GetMask` is a good idea; there's nothing like having the boss's brand-new PowerMac being denied access to your intranet's Web server on his very first try because its DNS entry hasn't been made by the network operations staff yet.

Hostname/Address Authentication in the Netscape Server

As with most aspects of Netscape Communications Server administration, you can set up hostname/IP address access control using a graphical interface. Start up the Administration Manager and select Restrict Access From Certain Addresses. This opens a document with extensive instructions for setting up access restrictions. You'll find fill-in boxes in this document for hostname/IP address restrictions. Figures 5.6, 5.7, and 5.8 show the essential parts of this form. You have all the same choices here for restricting access that you saw in the ncSA and CERN/W3 httpd servers.

Figure 5.6 : *Netscape Communications Server Host restriction (Part 1).*

Figure 5.7 : *Netscape Communications Server Host restriction (Part 2)*

The first step is to select what Netscape calls a *resource* to which you'll apply hostname/IP address restriction. For this purpose, a resource can be the entire Web server tree, a particular part of it, or one or more individual files. Clickable buttons (as shown at the top of Figure 5.6) enable you to select the resource you want. In this example, your resource would be the `/usr/local/web-docs/personnel` subdirectory of your httpd server tree. After you've selected your resource, scroll down the form to the headline `What To Protect`. (See Figure 5.7.) Here, you'll find two important choices.

You can simply accept the default of protecting everything in the selected resource. Or you can specify a wildcard filename pattern to match the files you want to protect. Notice the hypertext link labeled `wildcard pattern`, which takes you to a detailed document describing how wildcard pattern-matching works in the Netscape servers. (Essentially, it's standard UNIX shell filename expansion, but has some additional features.)

For the purposes of the example, you need not enter anything, because you're going to accept the default restriction to all files and directories in the `personnel` resource. However, you could have entered the wildcard pattern for the files to which you wanted to apply your hostname/IP address restrictions in the boxed and labeled `Pattern of files to protect`. The `Addresses to allow` section, which starts in Figure 5.7 and ends in Figure 5.8, tells you how to enter hostnames and IP addresses.

Figure 5.8 : *Netscape Communications Server Host restriction (Part 3)*

As with filenames, you can enter either specific individual hostnames or IP addresses, or wildcard patterns that match multiple hosts. The `Hostnames to allow` and `IP addresses to allow` boxes are shown in Figure 5.8 with the `personnel` example filled in.

The bottom of Figure 5.8 shows how you can set up a custom message to users who try to access restricted resources, giving them a reason for the denial of their request. You need not use this, but it can be friendlier than the generic `Not Found` message most httpd servers return. Here, I've set things up so the contents of the file `/usr/local/web-docs/private.txt` will be returned. This file could explain politely, for example, that access to `personnel` resources on the Web server is limited to the Personnel Department. After you finish the form, scroll all the way to the bottom (not shown in Figure 5.8) and click Make These Changes to apply your restrictions.

An Important Warning About Hostname/IP Address Authentication

All of the Web server software described in this chapter trustingly accepts the word of a requesting computer when it sends its IP address. Verification of this information is not possible. It's relatively easy for a user to change the hostname/IP address of a UNIX system, and laughably easy to change that of a pc or Mac. A curious, mischievous, or malicious person can reconfigure his computer to impersonate someone else's simply by changing the IP address of his own. Although this is an overall network security issue, not specifically one for your intranet, it's important you know about it because it can affect the security of your access controlled documents. Security-minded network administrators can use special hardware and software to prevent this sort of IP spoofing, but for

your intranet, you'll probably want to combine hostname/IP address authentication with username/password authentication, as outlined in the following section.

Combined Authentication

Now that you understand how username/password and hostname/IP address authentication work separately, consider how you can combine the two to beef up your access control. Begin with the Netscape Communications Server.

Combined Authentication in the Netscape Server

Netscape's scanty $40 documentation for the Communications Server doesn't address this subject directly, but you can infer from it how to implement combined username/password and hostname/IP address authentication. As you learned earlier, the Netscape server uses one or more user databases to store usernames and passwords, and you can apply access control limits based on both individual usernames and on group membership. Also, the Netscape server can restrict access by hostname/IP address, as described in the previous section. Although the Netscape Communications Server manual and its essentially identical online help describe these two methods as an either/or choice, it would appear that applying both kinds of access control to a single resource would result in both methods being applied. In other words, you can

- Define a set of users, such as the sample `personnel` group I've used, in the Netscape user database.
- Apply username/password authentication, such as to the `personnel` resource, limiting access to the members of the `personnel` group in the user database.
- Apply hostname/IP address restrictions, such as to the same `personnel` resource, limiting access to those computers in the `personnel` subdomain (or even to the individual computers of the members of the `personnel` group).

Because the documentation doesn't say what happens in such a situation, including whether there is an order of precedence in the testing of the access control rules, you should very carefully check how things work when you set up intersecting access control rules of this sort.

For example, it isn't clear which rule would be applied first. If the username/password authentication rule goes first, the user will be prompted for a username and password. The hostname/IP address rule would then deny access to even authenticated users. Applying the hostname/IP address rule first, however, will correct this problem.

Fortunately for those who want to have their access control rules perform exactly as they want them to, Netscape provides another means of access control, using Dynamic Configuration Files (DCFs). You can think of Netscape's DCFs as what I've called LACFs in this chapter-access control files that apply to a single directory or subdirectory on your Web server. Normally named `.nsconfig` (note the leading period in the filename), DCFs are organized into discrete sections with HTML-like markup. Each section is marked off by the tags `<Files>` and `</Files>`, in between which are access control and other rules that apply to the files specified. You can do many things with Netscape DCFs; here's an example that replicates the combined username/password and hostname/IP address access control to the `personnel` section of the example Web server:

```
<Files *>
RequireAuth dbm=webusers userpat="anne|joe|jerry" userlist="anne,joe,jerry"
RestrictAccess method=HTTP method-type=allow ip=123.45.67.* dns=*.personnel.mycompany.com
</Files>
```

This DCF, which goes in the top level of the `/usr/local/web-docs/personnel` directory, applies to all files and subdirectories in that directory tree. It requires username/password authentication, limiting access to users `anne`, `joe`, and `jerry` listed in the Netscape user database named `webusers`. It further limits access by both numerical IP address and symbolic hostname, both using wildcards. Notice that it's not necessary to specify both `allow` and `deny` rules; Netscape's server takes a more conservative approach to access restrictions than do ncSA and CERN/W3.

Tip

Netscape DCFs in lower-level directories take precedence over the rules in a DCF in a higher-level directory. Thus, by creating a `.nsconfig` file in the `personnel/executive` subdirectory, you can limit access to files in that directory to the user `anne`, as you did earlier in this chapter. Such a DCF might look like this:

```
<Files *>
RequireAuth dbm=webusers userpat=anne userlist=anne
RestrictAccess method=HTTP method-type=allow ip=123.45.67.89
dns=annspc.personnel.mycompany.com
</Files>
```

You can enable Netscape DCFs using fill-in forms similar to those shown earlier for setting up hostname/IP address access control. For example, you can enable a DCF for a given server resource, and the graphical interface will create a skeleton `.nsconfig` file. However, you'll need to use a text editor to add your own detailed access control and other directives.

Combined Authentication in the ncSA Servers

Combining username/password and hostname/IP address authentication in the ncSA httpd servers is fairly simple. You'll extend the rules in the `<Limit>` sections of either the GACF or LACF. Here's the now-familiar `personnel` example, modified to combine the two access control methods:

```
AuthType Basic
AuthName Personnel Only
AuthUserFile /usr/local/etc/httpd/userpw
AuthGroupFile /usr/local/etc/httpd/ourgroup
<Limit GET>
order deny,allow
deny from all
allow from personnel.mycompany.com
allow from 123.45.67
Require group personnel
</Limit>
```

As you can see, all you needed to do was to pull in both of the two sample methods shown in the earlier ncSA examples. Notice that order counts in the `<Limit>` section. Here, the hostname/IP address access control rules are applied first (using the `deny` and then `allow` sequence). After those rules are satisfied, the user is prompted for a password as the username/password authentication is applied. Based on this example, it's easy to modify this rule for an LACF in the `personnel/executive` subdirectory, simply by replacing `Require group personnel` with `Require user anne`.

Combined Authentication in the CERN/W3 Server

The CERN/W3 Server is similarly capable of combining username/password and hostname/IP address authentication. Here, you'll modify the `GetMask` directive in your GACF. Again, here is the modified `personnel` example, this time limiting access using both methods:

```
Protection Personnel {
     AuthType     Basic
     Passwordfile /usr/local/etc/httpd/passwd
```

```
        GroupFile    /usr/local/etc/httpd/group
        GetMask      @*.personnel.mycompany.com, @123.45.67.*,
                     personnel
    }
```

As with the ncSA example, this one applies hostname/IP address access control first (since it appears on the `GetMask` line first) and then username/password authentication. Both rules must be satisfied before access is permitted. To further restrict access, you'll need to develop LACFs for individual directories and subdirectories. As noted earlier, the CERN/W3 LACF file's format is completely different from that of the server's GACF. Here's one (note the file must be named `.www_acl`) that can be placed in the `personnel/executive` directory to limit access to the subdirectory to user `anne`, and only from a specific hostname/IP address:

```
*  :  GET  : anne@annspc.personell.mycompany.com,
ann@123.45.67.89
```

This simple file has just one rule. (The rule is usually a single line, with colon-separated records, but it can be wrapped, as shown above, after a comma.) No one other than the user `anne` (who must give a password under the rule in the previous example) can access any files in the `personnel/executive` directory. Moreover, `anne` must be accessing the files from her normal pc to be granted access, even if she gives the correct password. For more information on CERN/W3 LACFs, check out the online documentation at http://www.w3.org/pub/WWW/Daemon/User/Admin.html.

Secure/Encrypted Transactions

You can further enhance security on your intranet by encrypting Web transactions. When you use an encryption facility, information submitted by customers using Web fill-in forms-including usernames, passwords, and other confidential information-can be transmitted securely to and from the Web server.

There are a wide range of proposed or partially implemented encryption solutions for the Web, but most are not ready for prime time. Of the several proposed methods, only two have emerged in anything like full-blown form. Let's look at the Secure HTTP (S-HTTP) and Secure Socket Layer (SSL) protocols in this chapter. Unfortunately, the two protocols are not compatible with each other. Worse, Web browsers and servers that support one method don't support the other, so you can reliably use one or the other only if you carefully match your Web server and customers' browsers.

Secure HTTP (S-HTTP)

S-HTTP was developed by Enterprise Integration Technologies and RSA Data Security, and the public S-HTTP standards are now managed by CommerceNet, a nonprofit consortium conducting the first large-scale market trial of technologies and business processes to support electronic commerce over the Internet. (For general information on CommerceNet, see http://www.commerce.net/.) S-HTTP is a modified version of the current httpd protocol. It supports

- User and Web server authentication using Digital Signatures and Signature Keys using both the RSA and MD5 algorithms.
- Privacy of transactions, using several different key-based encryption methods.
- Generation of key certificates for server authentication.

EIT has developed modified versions of the ncSA httpd server and ncSA Mosaic (for UNIX and Microsoft Windows), which both support S-HTTP transactions. Although the licensing terms allow for ncSA to fold EIT's work into its free httpd server and Mosaic browsers, there's been no public indication of ncSA's plans to do so. Meanwhile, the CommerceNet secure ncSA httpd server and Mosaic browser are available only to members of CommerceNet. You'll find information about both packages, including full-text user manuals, at the CommerceNet home page http://www.commerce.net/.

Secure Sockets Layer (SSL)

S-HTTP seems to have been engulfed in the 1995 Netscape tidal wave. Unwilling to wait for widely accepted httpd security standards to evolve (as it was with HTML as well), Netscape Communications Corporation developed its own SSL encryption mechanism. SSL occupies a spot on the ISO seven-layer network reference below that of the httpd protocol, which operates at the application layer. Rather than developing a completely new protocol to replace httpd, SSL sits between httpd and the underlying TCP/IP network protocols and can intervene to create secure transactions. Netscape makes the technical details of SSL publicly available. In addition, C-language source code for a reference implementation of SSL is freely available for noncommercial use.

The Netscape Navigator Web browser has built-in SSL support, as does the Netscape Commerce Server; the Netscape Communications Server does not support SSL. Given Netscape's share of the Web browser market, it's hard to see how S-HTTP has much of a chance at becoming widely available. With the exception of ncSA Mosaic, most other Web browsers have-or have promised-SSL support. Some of them are Spry's newer product, Internet in a Box, Mosaic in a Box for Windows 95, and Release 2 of Microsoft's Internet Explorer for Windows 95 and the Macintosh. By the time you read this book, all of these packages might have completed their SSL implementations.

Note
Even though a browser might support secure transactions using SSL or S-HTTP, no transactions are actually secure except those between the browser and a compatible Web server. Thus, using Netscape, for example, won't provide any security unless you're also using the Netscape Commerce Server. It's also important to note that simply using a proxy service (that is, passing Web services through network firewalls) does not imply secure transactions unless *both* the proxy server and the destination server do.

As noted in the preceding section, the Netscape Commerce Server supports the company's SSL security mechanism. Other packages that support SSL include the Secure WebServer package from Open Market, Inc., (`http://www.openmarket.com/`), which also supports S-HTTP, and IBM's Internet Connection Secure Server, which runs under IBM's UNIX, AIX Version 4, and OS/2 Warp. (Evaluation copies of Secure WebServer for several UNIX systems are available at the Open Market Web site.)

Both Secure WebServer and Internet Connection Secure Server are based on Terisa Systems, Inc.'s SecureWeb Client and Server Toolkit. This package provides source code for developers building secure Web servers and browsers. The Terisa Toolkit supports both SSL and S-HTTP. For more information about the package, visit Terisa's Web site at `http://www.terisa.com/`. Open Market's promotional announcements about Secure WebServer state that the package supports secure transactions through Internet firewalls, but no details on just how this works are provided.

The Common Gateway Interface (CGI) and Intranet Security

CGI is the mechanism that stands behind all the wonderful, interactive fill-in forms you'll want to put on your intranet. Your customers might demand these kinds of intranet resources. CGI-BIN scripting is susceptible to security problems, so do your scripting carefully to avoid such problems.

You can minimize much of your risk of security breaches in CGI-BIN scripting by focusing on one particular area: Include in your scripts explicit code for dealing with unexpected user input. The reason for this is simple: You should never trust any information a user enters in a fill-in form. Just because, for instance, a fill-in form asks for a user's name or e-mail address, there is no guarantee that the user filling in the form won't put in incorrect information. Customers make typographical errors, but probing crackers, even those inside your organization, might intentionally enter unexpected data in an attempt to break the script. Such efforts can include UNIX shell meta-characters and other shell constructs (such as the asterisk, the pipe, the back tick, the dollar sign, and the semicolon) in an effort to get the script to somehow give the user shell access. Others intentionally try to overflow fixed program text buffers to see if the program can be coaxed into overwriting the program's stack. To be secure, your CGI-BIN scripts have to anticipate and deal safely with unexpected input.

Other problems inherent with CGI-BIN scripts include

- Calling outside programs, opening potential security holes in the external program. The UNIX sendmail program is a favorite cracker target.
- Using server-side includes in scripts, which dynamically generate HTML code. Make sure user input doesn't include literal HTML markup that could call a server-side include when your script runs.
- Using SUID scripts are almost always dangerous, whether or not in a CGI-BIN context.

Paul Phillips maintains a short but powerful list of CGI-BIN security resources on the Web. Check out http://www.cerf.net/~paulp/cgi-security, where you'll find a number of documents spelling out these and other risks of CGI-BIN scripting. For an extensive list of general CGI-related resources, go to Yahoo!'s CGI page, at http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/CGI_Common_Gateway_Interface/index.html.

Your Intranet and the Internet

Is your intranet accessible from the Internet? If so, all of the security problems of the Internet are now your intranet's problems, too. Throughout this book, an implicit assumption has been made that your intranet is private to your organization. You can, however, connect safely to the Internet and still protect your intranet. You can even use the Internet as a means of letting remote sites in your company access your intranet.

First, look at some Internet security basics.

Firewalls

It's a fact of Internet life that there are people out there who want to break into other people's networks via the Internet. Reasons vary from innocent curiosity to malicious cracking to business and international espionage. At the same time, the value of the Internet to organizations and businesses is so great that vendors are rushing to fill the need for Internet security with Internet firewalls. An Internet firewall is a device that sits between your internal network and the outside Internet. Its purpose is to limit access into and out of your network based on your organization's access policy.

A firewall can be anything from a set of filtering rules set up on the router between you and the Internet to an elaborate application gateway consisting of one or more specially configured computers that control access. Firewalls permit desired services coming from the outside, such as Internet e-mail, to pass. In addition, most firewalls now allow access to the World Wide Web from inside the protected networks. The idea is to allow some services to pass but to deny others. For example, you might be able to use the Telnet utility to log into systems on the Internet, but users on remote systems cannot use it to log into your local system because of the firewall.

Here are a couple of good general Web resources about Internet firewalls:

- Marcus Ranum's Internet Firewalls Frequently Asked Questions document at http://www.greatcircle.com/firewalls/info/FAQ.html
- Kathy Fulmer's annotated list of commercial and freeware firewall packages (with many hyperlinks to firewall vendor Web pages) at http://www.greatcircle.com/firewalls/vendors.html

If your company is also connected to the Internet, you'll want to know how to make sure your intranet isn't generally accessible to the outside world. You learned earlier in this chapter about denying access to your Web server using hostname and IP address authentication, but the fact that IP addresses can be easily spoofed makes it essential that you not rely on this mechanism as your only protection. You'll still want to rely on an Internet firewall to protect your intranet, as well as all your other network assets. Moreover, unless your corporate network is not connected to the outside world at all, you'll want to ensure the security of your other intranet services, including not only your Web servers, but also your FTP, Gopher, Usenet news, WAIS, and other TCP/IP network services.

Virtual Intranet

More and more companies with widely distributed offices, manufacturing sites, and other facilities are turning to use the Internet to replace private corporate networks connecting the sites. Such a situation involves multiple connections to the Internet by the company, with the use of the Internet itself as the backbone network for the company. Although such an approach is fraught with security risks, many organizations are using it for non-sensitive information exchange within the company. Using a properly configured firewall, companies can provide access to services inside one site's network to users at another site. Still, however, the data that flows across the Internet backbones between the corporate sites is usually unencrypted, plain text data that Internet snoopers can easily read. Standard firewalls don't help with this situation.

A number of firewall companies have recently developed Virtual Private Network (VPN) capabilities. Essentially, VPN is an extension of standard firewall capabilities to permit authenticated, encrypted communications between sites over the Internet. That is, using a VPN, users at a remote site can access sensitive data at another site in a secure fashion over the Internet. All the data that flows on the public Internet backbones is encrypted before it leaves the local network and then decrypted when it arrives at the other end of the connection.

The VPN is similar to a wide area network, or WAN, in that you are connecting one or more smaller networks together. WANs, however, typically utilize dedicated phone lines to communicate with each other. Many T1 and T3 lines are used for this purpose daily all over the world. This WAN setup can be quite secure because the information flow is over private telephone networks, not a giant chaotic network like the Internet.

WANs are not as prone to outages as VPNs are. That is because the VPN is at the mercy of a) your Internet service provider, and possibly b) your Internet service provider's Internet service provider. If either of these two sites go down, you're in the dark. That's a bad thing when you need to get at your corporate database in Chicago for a meeting that started five minutes ago.

What VPNs do give you is secure communications over a relatively cheap medium. Instead of paying hundreds or possibly thousands of dollars per network drop site, plus the cost of WAN equipment (which can get expensive), you simply pay for an Internet connection. These can be had for as little as $125.00 per month. Granted you won't get much bandwidth for that price, but it's a start. If you are considering extending your corporate network, consider VPNs too.

The most mature VPN product comes from Raptor Systems (`http://www.raptor.com/`), part of the company's Eagle family of products, and others are available from Checkpoint (`http://www.checkpoint.com/`) and Telecommerce (`http://www.telecommerce.com/`).

Figure 5.9 shows a schematic drawing of a VPN, reprinted with the permission of Raptor Systems, Inc. The cloud represents the Internet, and the firewall system, local network, and remote site are shown as workstations. The broad line connecting the workstation at the remote site to the local workstation illustrates the VPN. Such products make it possible for you to extend the availability of your intranet to remote company sites without having to set up a private network.

Figure 5.9 : *Virtual Private Network.*

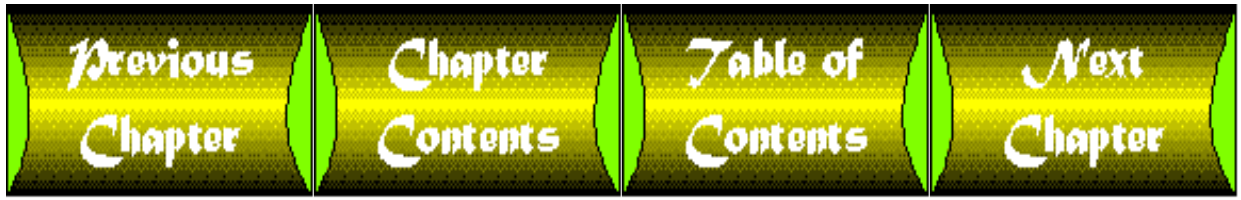
Summary

This chapter has dealt with implementing security on your intranet. Although an intranet is, by definition, internal to an organization, security is important not so much because it prevents things, but because it enables them. Judicious use of built-in security features of Web servers and other intranet resources can add value to your intranet by making new things possible. In this chapter, you have

- Considered the overall security aspects of your intranet.
- Learned how implementing security can actually broaden the ways in which your intranet can be useful in your organization.
- Learned how to use username/password authentication to limit access to resources on your intranet.

- Learned how to provide secure access to intranet resources to groups of customers.
- Learned how to restrict access to sensitive resources based on customers' computer hostnames or network addresses.
- Learned about the security aspects of CGI-BIN scripting.
- Learned about encrypted data transmission on your intranet to protect critical information.
- Learned important information about securing access to your intranet in the case where your corporate network is attached to the Internet.
- Learned how to provide-and limit-secure access to your intranet from outside your immediate local network.

In the next section, you will examine the various methods available to Java for connecting to databases.



Chapter 6

Database Connectivity Options

CONTENTS

- Introduction
- Database Overview
- Database Terminology
- Database Locations
  - Local and Remote
  - Tiering 1-2-3
- Database Access Methods
  - Native Drivers
  - ODBC
  - SQL
- Databases and Java
  - Access via Web Server
  - Access via Proprietary Server
  - Network Access
  - Direct Access
  - JDBC
  - JDBC Goals
  - JDBC Overview
  - JDBC Vendor Support
- Summary

---

connect \k-'nekt\ v: *to join or fasten together*

Introduction

A computer application is nothing more than a bundle of source code that manipulates a set of data. That data is key to the operation and functionality of the application. The data, however, can come from a variety

of places. Be it user entered, in-memory defaults, or offline stored databases, these databases and their relationship to Java are the subject of this chapter.

In this chapter I'll cover the following topics:

- A quick introduction to databases. This will cover some "databasics," and some terms that will be used in the remainder of this book
- How Java can interact with databases
- A sampling of several Java/Database connectivity solutions
- The Java Database Connectivity Package, or JDBC
- A working JDBC/Database package called mSQL

This chapter will set the stage for developing our database framework and the database applications later in the book.

Database Overview

Databases come in all different shapes and sizes. They can be flat files of ASCII data (like Q&A) or complex binary tree structures (Oracle or Sybase). In any form, a database is a *data store*, or a place that holds data. The type of data that is contained in the store is irrelevant.

If a database is simply a collection of data, then what keeps track of changes to this data? That is the job of the *database management system*, or DBMS. Some DBMSs are *relational*. Those are RDBMSs. The relational part refers to the fact that separate collections of data within the reaches of the RDBMS can be looked at together in unison. The RDBMS is responsible for ensuring the integrity of the database. Sometimes, things will get out of whack and the RDBMS will keep all that data in line.

Note
Very few DBMSs are not RDBMSs these days. We will refer to any database, be it DBMS or RDBMS as RDBMS for the remainder of this chapter and book.

There are so many different types of RDBMSs available today that it would probably take two full books to give a summary of each one. This overview is a quick start guide for those of you who are not familiar with some newer concepts in data storage.

Database Terminology

In the days of yore, programming database applications was pretty simple. There were mainframe databases and there were very few microcomputer databases available. The ones that were available cost an arm and a leg. The cheap ones were, well, you got what you paid for. But you always had database files, records, and fields.

The database terms of yesteryear, however, have been replaced by new ones. Some of the bigger database companies like Oracle and Sybase have redefined database terminology. The main thrust of this redefining is most likely in response to the larger customer base that is not "programming-literate."

A programmer can deal with files, records, and fields. But more and more non-technical people are creating database applications and queries these days. Their formal training is through the use of applications. As you'll see, some of the new terms are commonly found in spreadsheet and word processing programs.

The following is a list of current database terms that will be used in the rest of this chapter and throughout the book:

- *Client\Server*. Client\server is more of an architecture than a tangible entity. The client is a computer system that requests the services provided by an entirely different computer system. On a smaller scale, the client and server may be separate processes running on the same computer system. The distinction is that there is a service provider (the server), and a consumer of that service (the client). For your purposes here, the server would be the RDBMS, and the client is your application that is requesting data from the server.
- *Database* or *Instance*. The database or instance is the entity, or collection of data, that is created and stored for retrieval and modification. Depending on the RDBMS, several of these entities can exist on a single machine. For instance, multiple Oracle instances can exist on a UNIX server. Each has a distinct area for data, and unless properly configured, they have no knowledge of each other. A database or instance is comprised of *schemas*.
- *Schema*. A schema is a collection of database objects that belongs to a single user of the database. Databases have many users.
- *Table.* A table is a database object that contains a single set of data. Like things are stored in tables. For instance, at a company a normal table to have is an employee table. This would store all kinds of information about an employee. A table contains *rows* of data.
- *Row*. A row of data is a single record in a table. A row is divided into *columns*.
- *Column*. A column is the smallest unit of data in a table. It is a part of the row. When data is displayed in a spreadsheet-like fashion, a column would be the up/down slice of data.

Figure 6.1 is a visual representation of the preceding terminology.

Figure 6.1 : *A visual guide to the database terminology used in this book.*

Database Locations

Databases can exist in various places. Larger databases require the horsepower of a multiple CPU server. Smaller databases can get away with only a microcomputer serving data. But where the data is stored is important to the application programmer. There are only two options for database location: local and remote.

Local and Remote

A local database is one that resides on the machine on which client applications run. Local databases offer the fastest response time because there is no network traffic between the client (your application) and the server (the RDBMS engine). Some examples of local databases are Paradox from Borland, Access from Microsoft, and Personal Oracle from Oracle.

A remote database, on the other hand, is one that resides on a machine that the client software does not run on. This is an important distinction for two reasons:

- Response time will be slower. Even the fastest network connections will not get you the same response time as a local database. Bear in mind, the difference might be seconds or less.
- An additional software layer is needed to communicate with the database.

The first item is the general case. It is also only relevant for performance-critical applications. A well-tuned RDBMS server can out-perform a poorly tuned local server in some cases.

The second item, however, might cause grief and headaches that weren't expected. With some database server and client products, a second software layer is necessary to transparently interact with the remote database. This software might be an optional software package that is not included with the server software. I'll get to this layer in the next topic, "Database Access."

Tip
Here are some of the database software vendor Web sites, and some excellent sources of database information: • Data Access Corporation. http://www.daccess.com/ • Informix Software, Inc. http://www.informix.com • Free Database List. http://cuiwww.unig.e.ch/~scg/FreeDB/FreeDB.list.html • Microsoft Corporation. http://www.microsoft.com • Oracle Corporation. http://www.oracle.com • Progress Software Corporation. http://www.progress.com • Sybase, Inc. http://www.sybase.com

Tiering 1-2-3

There is one more topic I'd like to touch upon before I get into accessing databases: it is the client-server concept of multi-tiering. Unfortunately, I have heard about ten different explanations of this concept and not one of them ever is the same. The following is my take on the single-, two- and three-tiered architectures.

Single-Tiered

The application and the data reside together logically. These are not usually database programs. An example would be a calculator program. The logic and its data reside together. Figure 6.2 shows a model of a single-tier application.

Figure 6.2 : *A single-tier application.*

Two-Tiered

The application resides in a different logical location than the data. These are usually database applications. Most client/server applications fit into this category. Figure 6.3 shows a model of a two-tier application.

Figure 6.3 : *A two-tier application.*

Three-Tiered

In a three-tiered system, the application resides in a different logical location than the logic of the application *and* the data.

To put it another way, the client software makes a call to a remote service. That remote service is responsible for interacting with the data and responding to the client. The client has no knowledge of how and where the data is stored. All it knows about is the remote service. Conversely, the remote service has no knowledge of the clients that will be calling it. It only knows about the data.

This partitioning of logic allows for better data control and reuse of existing code. Three-tier architecture is becoming more widespread because more and more tools are being created that handle the tiering automatically.

Figure 6.4 shows a model of a three-tier application.

Figure 6.4 : *A three-tier application.*

Database Access Methods

In order to "talk" to your database you need to use some sort of software. Whether it comes with your server or you have to write the code yourself, this software is essential for database communications.

Although there are innumerable methods of retrieving and storing data, the following are the most common: native, ODBC, and SQL. SQL is probably the most common data access method, ODBC a close second, and, except for driver creators, native methods are rarely used. Figure 6.5 illustrates the software layers in the three methods outlined.

Figure 6.5 : *The software layers that can be used to access data.*

The following is a short discussion on each of the three access methods.

Native Drivers

Native drivers give you the raw power of talking directly to your database. When you make the connection and retrieve data, you are talking right to the file system. An example of a native driver is the Oracle Call Interface, or OCI from Oracle Corporation for Oracle databases.

Native drivers are usually statically or dynamically linked into your software at compile time.

Advantages

- Very fast - with the actual database access code linked together with your program, data access is lightning fast.

Disadvantages

- Not portable - applications created in this manner are usually not portable to other platforms without code modifications.

- Inflexible - because the driver is linked into your application, changes in the driver software require possible recompilation of your application.

ODBC

Open Database Connectivity, or ODBC, is a standard developed by Microsoft Corporation. ODBC is an application program interface for accessing data in a standard manner from an abundance of data sources regardless of their type. If the data source is ODBC compliant, your program can talk to it.

ODBC drivers are available for almost every major database vendor.

Tip
Check out the ODBC Homepage for some cool ODBC links! `http://ourworld.compuserve.com/homepages/VBrant/`

Advantages

- Fast - Not as fast as native drivers, but pretty fast. You have one additional layer of software to go through to get to your data.
- SQL Enabled - You can use SQL to query the database.

Disadvantages

- Not portable - Applications created in this manner are usually not portable to other platforms without code modifications.
- Inflexible - Because the driver stub is linked into your application, changes in the driver software API require possible recompilation of your application.

SQL

Although not a "layer" of access to databases like ODBC or native drivers, the Structured Query Language, or SQL, provides a standard method of querying data from different data sources.

SQL, usually pronounced like the word "sequel," was adopted as an industry standard in 1986. SQL was completely overhauled in 1992 and the new language was called SQL92, or SQL2. Work is currently in progress to produce the next generation, SQL3. The following is a short list of SQL commands and their meanings:

- `COMMIT` - Most RDBMS's work with units called *transactions.* A transaction can be made up of multiple actions. The `COMMIT` command instructs the database to record all the actions that you have performed up until this point, and to reset the transaction. When you `COMMIT`, the data is available to everyone who has access. Before the `COMMIT` occurs, however, only people with access to your schema can see the changes.
- `INSERT` - Instructs the database to insert rows into a table.
- `DELETE` - Instructs the database to delete rows from a table.
- `ROLLBACK` - The `ROLLBACK` command is used instead of a `COMMIT`. This instructs the database to remove any changes you've made all the way back to the last `COMMIT`. This is very useful for long,

multiple-table updates. For example, let's say you need to add 10 rows to a table. After inserting 9 rows, the 10th insert fails. The first 9 rows must be removed for the data to retain its integrity. Using the `ROLLBACK` command, the 9 inserted rows will not be recorded.

- `SELECT` - Instructs the database to return rows from a table.
- `UPDATE` - Instructs the database to modify rows in a table.

We'll go over the syntax of some of the more commonly SQL commands. Just a reminder, though, that this is by no means an exhaustive SQL syntax review. Dozens of books about SQL have been published. The command syntax that follows is general ANSI SQL and might not be correct for your RDBMS. Please check your documentation if there is any doubt.

Note
In the syntax examples that follow, any parameter that is enclosed in square brackets ([]) is an optional parameter and may be left out.

The `WHERE` Clause

Most SQL commands act on all the rows of a table at one time. These global actions can be restricted to a limited number of rows by the use of a `WHERE` clause. The `WHERE` clause allows you to specify criteria that is used to limit the number of rows that an action is performed.

The general syntax for a `WHERE` clause is as follows:

```
COMMAND arguments WHERE [[[schema.]table.]column OPERATOR
value] [AND|OR [[[schema.]table.]column OPERATOR value]]
```

where

`arguments` are the arguments specific to the `COMMAND`.

`schema` is the area where the table exists.

`table` is the table where the column lives.

`column` is the column name to compare with *`value`*.

`value` is a literal or column name to compare with *`column`*.

Multiple operations may be checked in the `WHERE` clause. These can be linked with either the `AND` or `OR` keyword.

The `OPERATOR` might be many things depending on the RDBMS in use. Table 6.1 shows the `OPERATOR`s that are available in most RDBMSs.

Table 6.1. Operators.

Operator	*Meaning*	*Example*
<	Less than	`emp_id < 10`
>	Greater than	`salary > 50000`
=	Equal to	`can_be_paged = 'Y'`
<=	Less than or equal to	`user_count <= 128`
>=	Greater than or equal to	`user_count >= 0`

<>	Not equal to	lost_shovels <> 5
is	For checking NULL values	name_suffix is NULL
not	For negating an operator	name_suffix is not NULL
like	Allows for the use of wildcards	first_name like '%MUNSTER%'

Please note that the WHERE clause cannot be used alone. It must be appended to a DELETE, SELECT, or UPDATE command.

INSERT

The INSERT statement allows you to create a new row in a table.

The syntax for an INSERT statement is as follows:

```
INSERT INTO [schema.]table [(column[,column…])] VALUES
(value[,value])
```

Where:

schema is where the table exists

table is the target table

column is the column name(s) of the data you wish to insert

value is the value(s) that you wish to insert

Examples

```
INSERT INTO EMPLOYEE ( EMP_ID, LAST_NAME ) values ( 1,
'Munster' )
INSERT INTO ADDRESS ( EMP_ID, STREET_ADDRESS ) VALUES
➧ ( 1, '1313 Mockingbird Lane' )
```

DELETE

The DELETE statement allows you to remove a row or rows from a table.

The syntax for a DELETE statement is as follows:

```
DELETE FROM [schema.]table [WHERE expression]
```

where

schema is where the table exists

table is the target table

expression is an expression as outlined in the preceding WHERE clause section

Caution
Without a WHERE clause, the DELETE command removes all rows from a table.

Examples

```
DELETE FROM EMPLOYEE WHERE EMP_ID = 1
DELETE FROM ADDRESS WHERE CITY LIKE 'chICAG%'
```

SELECT

The SELECT statement allows you to retrieve a row or rows from a table.

The syntax for a SELECT statement is as follows:

```
SELECT [[schema.]table.]column [,[[schema.]table.]column] FROM
[schema.]table [WHERE expression]
```

where

schema is where the table exists

table is the target table

column is column or columns to retrieve. You can use the asterisk ('*') to indicate that the SELECT statement should return all columns.

expression is an expression as outlined in the preceding WHERE clause section

Examples

```
SELECT EMP_ID FROM EMPLOYEE
SELECT LAST_NAME, FIRST_NAME, MID_NAME FROM EMPLOYEE WHERE
EMP_ID = 666
```

UPDATE

The UPDATE statement allows you to modify a column or columns in one or more rows in a table.

The syntax for an UPDATE statement is as follows:

```
UPDATE [schema.]table SET [[schema.]table.]column = value
[,[[schema.]table.]column = value] [WHERE expression]
```

where

schema is where the table exists

table is the target table

column is column or columns to modify

expression is an expression as outlined in the preceding WHERE clause section

value is the new value that the column should hold

Examples

```
UPDATE EMPLOYEE SET SALARY = SALARY + ( SALARY * .05 )
UPDATE ADDRESS SET ZIP_CODE = 60805 WHERE ZIP_CODE = 60642
```

Advantages

- Simple English Syntax - You do not need to know how to program to use SQL.
- Standardized - Usually, SQL accesses one RDBMS, which can be used to access another RDBMS.

Disadvantages

- Wordy - SQL queries can become quite complex and lengthy. Because SQL syntax uses regular English words, the results can get somewhat monotonous.

SQL uses simple English words to instruct the database to perform certain actions. SQL can be used with almost every major database product available today. In addition, you can even use SQL syntax to interact with a data source using ODBC!

Tip
Here are some useful SQL Web sites: • Ask the SQL Pro: http://www.inquiry.com/techtips/thesqlpro/ • SQL Reference Page: http://www.contrib.andrew.cmu.edu/~shadow/sql.html

Databases and Java

Java is a platform independent programming language. In order to access a database with Java you need to use a platform-independent method. This is easy to accomplish, but might be quite cumbersome. Usually, you'd have to create a server program that speaks to the database of your choice and then your Java programs would need to interact with your server.

If you want to abandon your platform independence, you can always write native code to access your data. This would involve C or C++ programming in UNIX or Windows 95/NT. Without the proper tools, this can be a real headache.

So what does a programmer do when he needs data access from Java? There are several ways to get data to your application. These database access services include:

- Access via Web Server. The client requests data and a CGI program talks to the database and returns the formatted data to the Java program.
- Proprietary Server. The proprietary server knows how to talk to databases. Your Java program makes requests of this server, which in turn fulfills your requests.
- Network Access. Network access, through the use of a class library, interacts with networked database servers. These database servers are capable of responding to direct queries from your Java program.
- Direct Access. A direct access is also a class library that allows you to manipulate local databases. Those are databases that are on the same machine your Java program is running on.

The latest arrival onto the database scene is Sun Microsystem's own Java Database Connectivity, or JDBC. JDBC is big news and I'll get into that in a bit. But for now, let's take a closer look at these other data access options.

Access via Web Server

To access data via CGI scripts, the applet or application requests data from an HTTP server just like any other Web document. However, encoded in the CGI parameters is the database query that is to be executed. Once the HTTP server receives the request, it passes the parameters to the proper CGI program. The CGI program then performs the database query on the program's behalf.

Because this is a three step process, response time is not fantastic. But if the HTTP server is on the same machine as the application and the database, response times are better. Figure 6.6 illustrates a typical database access method via a Web server.

Figure 6.6 : *Database requests through HTTP and CGI.*

This option is useful for specific types of databases that cannot be moved or can only live on certain types of environments.

Of note is Oracle's WebServer product, which can access Oracle databases directly without going through a CGI program. Using the WebServer, you can embed database requests right in your HTML pages. Once these pages are received by the Oracle WebServer, they are parsed, and recreated on-the-fly. The recreated pages include, for instance, data from an Oracle database. This seamless integration removes the need for a separate CGI program to access data. This is a lot faster and is a complete database solution.

Web Server Database Solutions

Here are some CGI/HTTP database access solutions available for Java:

- Bulletproof's JAGG - http://www.bulletproof.com
- Oracle WebServer 2.0 - http://www.oracle.com

Access via Proprietary Server

Another Java database option is going with a proprietary server. In this access mode, a non-Web server listens for service requests. Once one is received, it will perform a database query on the client's behalf and return the results to the client.

Proprietary Server Access Solutions

Here are some proprietary server database access solutions available for Java:

- dbKona - http://www.weblogic.com/
- javaSQL - http://www.patriot.net/users/anil/java/javaSQL/
- Jade - http://hktrade.com/clients/kwan/

Network Access

Network access database solutions for Java are the best. These provide platform independence because the actual network connection and requesting is done in Java.

Network Access Solutions

Here are some network database access solutions available for Java:

- Ask Joe - http://www.imsweb.com/AskJoe.html
- MsqlJava - http://mama.minmet.uq.oz.au/msqljava/
- OCI/Java Gateway - http://multiserver.kuai.se/

Direct Access

Direct access is probably the fastest method of database access, however it is the least portable. You could possibly lose any platform independence you've gained by using Java in the first place. However, if you don't care about independence, this method of database access is by far the best performance-wise.

Direct Access Solutions

Here are some direct database access solutions available for Java:

- JavaBase - http://sio.ucsd.edu/~gabe/
- OraJava - http://users.aimnet.com/~omd/OraJava.html
- SybJava - http://users.aimnet.com/~omd/SybJava.html

JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. As a result of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. *SQL Level API*
   The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to "generate" JDBC code and to hide many of JDBC's complexities from the end user.
2. *SQL Conformance*
   SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.
3. *JDBC must be implementable on top of common database interfaces*
   The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.
4. *Provide a Java interface that is consistent with the rest of the Java system*
   Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.
5. *Keep it simple*
   This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.
6. *Use strong, static typing wherever possible*
   Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.
7. *Keep the common cases simple*
   Because more often than not, the usual SQL calls used by the programmer are simple `SELECT`'s, `INSERT`'s, `DELETE`'s and `UPDATE`'s, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.
8. *Use multiple methods to express multiple functionality*
   There are two schools of thought on functionality. One is to provide a single entry point into an API. The programmer must then use a variety of control parameters to achieve the desired result. The second is to provide multiple points of entry into the API. This second school of thought was the goal of JDBC. This goal is similar to the way that the Java system was designed.

JDBC Overview

JDBC is divided into two parts: The JDBC API, and the JDBC Driver API. The JDBC API is the programmer's API. This half is where you will spend most of your time coding. The JDBC Driver API is for driver writers and database vendors to create connectivity modules for their database software. Figure 6.7 shows the complete JDBC API class hierarchy.

Figure 6.7 : *The JDBC class hierarchy.*

The JDBC API consists of many classes and interfaces. This structure makes the API a semi-abstract set of functionality. In order for JDBC to be of any use, a database vendor must fill in the blanks.

Four of these blanks will be the center of any database programming that you do with the JDBC API. These four classes are

- `DriverManager` - This class is responsible for managing all the available database drivers. The `DriverManager` class retrieves the list of available classes for drivers from the system property called `jdbc.drivers`. Each of the found drivers is loaded.
- `Connection` - This interface defines a session between an application and a database.
- `Statement` - This interface is used to issue a single SQL statement through a `Connection`. It owns only one `ResultSet`. Therefore, multiple concurrent SQL statements must be done through multiple `Statements`. Issuing a SQL statement while processing the `ResultSet` of a previous `Statement` will result in the overwriting of the results.
- `ResultSet` - This interface provides access to the data returned from the execution of a `Statement`.

This short overview is only a small portion of the JDBC API. There is support for other database features such as cursors and stored procedures.

JDBC Vendor Support

Many database vendors have already pledged support of JDBC. The following is a list of vendors who plan on supporting JDBC. This list is from the JDBC Web site as of June, 1996.

- Borland International Inc. - http://www.borland.com
- Bulletproof - http://www.bulletproof.com
- Cyber SQL Corporation - http://www.cybersql.com
- Dharma Systems Inc. - http://www.dharmas.com
- Gupta Corporation - http://www.gupta.com
- IBM's Database 2 (DB2) - http://www.software.ibm.com/data/db2/index.html
- Imaginary (mSQL) - http://www.imaginary.com/~borg/Java/
- Informix Software Inc. - http://www.informix.com
- Intersoft - http://www.inter-soft.com/eng/products/system/essentia/essentia.html
- Intersolv - http://www.intersolv.com
- Object Design Inc. - http://www.odi.com
- Open Horizon - http://www.openhorizon.com
- OpenLink Software - http://www.openlink.co.uk
- Oracle Corporation - http://www.oracle.com
- Persistence Software - http://www.persistence.com
- Presence Information Design - http://cloud9.presence.com/pbj/

- PRO-C Inc. - http://www.pro-c.com
- RogueWave Software Inc. - http://www.roguewave.com
- SAS Institute Inc. (tm) - http://www.sas.com
- SCO - http://www.vision.sco.com/brochure/sqlretriever.html
- Sybase Inc. - http://www.sybase.com
- Symantec - http://www.symantec.com/cafe
- Thunderstone - http://www.thundestone.com
- Visigenic Software Inc. - http://www.visigenic.com
- WebLogic Inc. - http://www.weblogic.com
- Working Set, Inc. - http://dataramp.com
- XDB Systems, Inc. - http://www.xdb.com

Most big database vendors are on this list. If your database vendor is not on this list, fear not. There will be a JDBC-ODBC bridge driver from Sun. If your database vendor has ODBC support, then you will be in the clear.

For more information on JDBC, please visit the JDBC Web site at JavaSoft: http://www.javasoft.com.

Summary

This chapter was an overview of databases and database connectivity options that you have at your disposal. I discussed the database terminology that you will be using in the book for the first time. You are now familiar with rows and columns of data. The more you use these terms, the more comfortable you will be using them (it took me nearly 6 months!).

After the terminology discussion, I talked about the differences between local and remote databases. This led you right to a discussion about the advantages and disadvantages of various database access methods.

Finally, I ended this chapter discussing Java Database Connectivity, or JDBC. JDBC is the hot, new, up-and-coming database connectivity tool for Java.

Chapter 7

A Model Intranet Application

CONTENTS

- Introduction
- A Quick Overview of Intranet Applications
  - Configuration File Processing
  - Logging to Disk or Screen
  - Database Connectivity
  - Look and Feel
- Coding Style Notes
- Code Layout
  - Parentheses and Code Blocking
  - Using Tabs Versus Spaces
  - Liberal Use of Spaces
  - Multiple Lines Per Statement
- Comments
- Code Order
- Summary

---

model \'mäd-l\ n: *a pattern of something to be made*

Introduction

Intranet applications are like a corporate application suite. These include word processors, project planners, spreadsheets, and many other useful and productive applications. They encompass all departments and touch many types of data. But unlike the business productivity application suites available today, your intranet applications should all share a common foundation.

Creating a suite of applications from scratch is tedious and boring. Cutting and pasting code is easy, but that goes against all object-oriented programming practices. What is needed is a basic structure from which to build your applications. This foundation should be flexible, stable, and extensible.

This chapter introduces you to four standards that all of our applications share. These standards provide a flexible, stable, and extensible foundation for developing intranet applications. These four standards together produce a prototype or a model application that can be used as a base when developing applications.

A Quick Overview of Intranet Applications

On an intranet, the applications that are built share much of the same functionality. Sure they all do different things but they also do a lot of the same things. These commonalties should become the foundation for your intranet application framework. They are the base and are truly your application standards.

Our primary design goal is to provide a set of standard application features. These features create a familiar atmosphere for all your intranet applications. Familiarity provides users with a sense of comfort because they don't have to learn an entirely new program. Apple capitalized on this idea years ago when it introduced the Macintosh computer. If you learned how to use the Mac, then you knew how to run almost every Macintosh application written. It was the consistency and adherence to set standards that made this possible. Microsoft Windows has since capitalized on the same concept. The design presented here is not quite a Macintosh but what it provides is something similar: consistency.

The following four standard features are what we strive to provide in the model intranet application design:

- Standard configuration file processing
- Standard logging to screen or disk
- Standard database connectivity
- Standard look and feel

Let's examine each goal individually, show an example, then point you in the right direction to find more information regarding each particular standard.

Configuration File Processing

Using configuration parameters in programming can be a real hassle unless a stable foundation is in place. Generally you end up coding a new configuration scheme with each application. What we can create is a class that provides a solid method of getting configuration parameters. This method encompasses configuration files on disk and overridden parameters passed in by way of the command line of the application.

The configuration file in Listing 7.1 is an example of the kind of configuration files the applications have.

Listing 7.1. A sample configuration file.

```
# Configuration file for Employee Maintenance
WindowTitle=Employee Maintenance
server=tcp-loopback.world
user=munster
password=
```

At the start of the application, we read the configuration file into memory. These parameters are merged with any configuration parameters that are passed in by way of the command line. The applications then need a consistent method of retrieving these parameters from the configuration parameter storage area. We should model the retrieval method after the method used in regular Java applets.

Note
Chapter 8, "Utility Classes," discusses the `ConfigProperties` class which implements these design goals.

Logging to Disk or Screen

For tracking problems during the development cycle and for error logging after your application has been deployed, a log file is just the ticket. A common log file is even better for all of your applications and users. A common log file is easily searched and filtered for errors. This standard logging mechanism is the first standard feature that we need to supply.

To facilitate such a log file, we need to create it on disk. The log file is appended to each time; it is never overwritten. If we overwrite the file, then information from a previous session can be lost. You know how annoying that can be.

But what if the disk log fails to open, or if it can't be written to? Well, we need a backup log. These failed log entries should go to the screen.

This screen logging facility produces the same log information to a window, or using the `System.out` facility. When an application fails to create a disk log, all log output goes to the screen. Also, this screen log is used automatically if no disk log is specified or if there is an error.

Note
`System.out` is a Java system object. It is an alias for the console in Windows. In UNIX it is an alias for `stdout`.

Common Log Entries

The entries in the log file should follow a standard, one that is easy to view and search. This format should be used across all of your intranet applications, and possibly your non-intranet applications as well. This log file format should be simple enough so that other programs might even use it.

You might find in the future that you use a third-party network management tool that can monitor your intranet applications and their log files. These tools can be a lifesaver in a pinch, so why not think about their needs as well?

Therefore, the following log file format is what to go with as the design. It includes all the information needed in an easy-to-use format:

application|*user*|*date*|*level*|*entry*

where

application is the name of the application

user is the user who is running the application

date is the date of the log entry

level indicates the severity of the entry.

Six predefined levels are available: debug, informational, notice, warning, error, and fatal. Each is denoted in the log by the first letter in its name. D for debug, I for information, and so on.

Listing 7.2 shows some sample log entries.

Listing 7.2. Sample log entries.

```
Employee|960609|I|Application [Employee] started at Sun Jun 09 22:27:33
1996
Employee|960609|I|Port = 4333
Employee|960609|I|Server = mars.mcs.net
Employee|960609|I|Title = Employee Maintenance
Employee|960609|I|Application [Employee] ended at Sun Jun 09 22:28:38
1996
```

At startup, all of the intranet applications write several things to the log:

- A startup message showing the application and time/date of startup
- Optionally, the contents of the configuration file
- Optionally, any arguments passed in on the command line

At shutdown, the application writes a corresponding entry to its startup message. This is shown in Listing 7.2, also.

Note
Chapter 9, "Logging Classes," discusses the `DiskLog` and `ScreenLog` classes which implement these design goals.

Database Connectivity

Connecting to databases with Java is a key point in your intranet application design stage. Various methods are currently available. Some are HTTP server extensions that return data in HTML format. Others are non-portable system-dependent solutions. A more Java-like option is JDBC.

Note
JDBC, along with other database connectivity options are covered in depth in Chapter 6, "Database Connectivity Options."

It appears today that JDBC is the strongest supported database standard for Java. For this reason alone, JDBC is chosen as the database connectivity package for the intranet applications in this book. Using JDBC allows us to choose from almost any database and provides the flexibility to change databases when coding is complete.

To simplify database connectivity just a bit, we need to create a class that encapsulates the more monotonous aspects of connecting and disconnecting from a database server. This new class provides a connection strategy that is simple to use and easily extensible. This class also encapsulates much of the rudimentary JDBC initialization and cleanup. All we need to do is extend this class for each database we need to connect with.

Listing 7.3 shows how easy the class is to use.

Listing 7.3. How you should use your Database Connector Class.

```
//  Make the connection...
if ( myConnector.connect( "munster", "hermy", "tcp-loopback.world" ) )
{
    //  Connection successful...
}
else
{
    //  Connection failed...
}
```

As you can see, the `connect()` method is called with the connection parameters necessary to make the connection. The exception handling is handled for you in the class, returning nothing but a simple `true` or `false`. This indicates the connection state as well.

Note
Chapter 10, "Database Classes," discusses many database classes. One of them is the `DBConnector` class. This class implements many of these design goals.

Look and Feel

The final standard about the intranet applications is that they should have a consistent look and feel. This is achieved through the use of standard font, and a consistent layout of components.

Figure 7.1 illustrates the standard look and feel of the model intranet application. This application is the "Hello World" example presented in Chapter 11, "User Interface Classes."

Figure 7.1 : *The standard intranet application look and feel.*

Referring to Figure 7.1, you see the following standard application attributes:

- A standard Java font. We choose Dialog, size 12. In Windows 95, this is 8-point MS Sans Serif.
- A status bar along the bottom to display messages to the user.

You might notice that there is a menu shown in Figure 7.1; however, no menu is shown as a standard look and feel. This menu varies from application to application. It is not fair to impose a rigid menu structure on applications that might not even need a menu. Therefore, menus are not part of the application design.

Note
Chapter 11 discusses many classes which implement these design goals.

Coding Style Notes

There are several things about the source code in this book that might be different from other Java books that you have seen. A good percentage of the code styles here are simply done for readability. Here is a list of what I think is different and superior:

- Code layout is cleaner.
- Comments are used.
- Source code order is different.

Let's go over each individually and show some examples.

Code Layout

The code in this book is laid out, or rather formatted, in a manner that might be different from other Java books that have been published. The format used in this book is my personal coding style for C and C++. I feel that this format is more readable and easier on the eyes and brain. My style differs in the ways described in the next few sections.

Parentheses and Code Blocking

The original Java samples and source code that come with the JDK use what is known as the K&R style of code blocking. This style places the open parenthesis at the end of the statement that begins the block. Listing 7.4 is an example of K&R style.

Listing 7.4. The `BadKitty` class-K&R style.

```
public class BadKitty extends Kitty {
    public BadKitty( int clawCount ) {
    }
}
```

K&R stands for Kernigham (Brian) and Ritchie (Dennis), the creators of the C programming language. In their (in)famous book, *The C Programming Language*, this was the manner of code layout, hence the name.

In this book however, the parenthetical style used is that the braces match up in the same column. This style is easier to read and more clearly marks the blocks of code. Listing 7.5 illustrates this point.

Listing 7.5. The `BadKitty` class-Non-K&R style.

```
public class BadKitty extends Kitty
```

```
{
    public BadKitty( int clawCount )
    {
    //    Code goes here!
    }
}
```

Using Tabs Versus Spaces

Whenever code is indented, tabs are used. Tabs keep code aligned and are portable between operating systems. The tab stops for the code in this book are every four spaces.

Liberal Use of Spaces

As you may or may not know, the compiler really doesn't care about spaces in your code. All white space and comments are stripped out before the actual compilation is done. Adding more spaces than necessary is another readability issue. Wherever possible, spaces are used. This might seem a bit lengthy. However, you will see the improvement. The following line of code uses no spacing:

```
blackBoxToUse=(myBlackBox==null)?defaultBlackBox:myBlackBox;
```

And this one spaces things out nicely:

```
blackBoxToUse = ( myBlackBox == null ) ? defaultBlackBox : myBlackBox;
```

Look how much more readable the second line is! You can quickly parse the line and understand what is going on. You don't spend time parsing.

Multiple Lines Per Statement

In an effort to make the code more readable, all class and function declarations are split into two, three, or four lines. The number of lines depends on the number of elements that make up the statement. The rules are different between class and function declarations.

Class Declarations

Class declarations in this book can appear on up to four lines and are in the following order:

```
[access] class
name
[extends super_class]
[implements interface]
```

Usually this declaration is made on a single line. However, by splitting it up into multiple lines you can immediately learn information about the class without even reading the declaration. Listing 7.6 illustrates the class declaration.

Listing 7.6. A beautiful class declaration.

```
public class
FileDate
extends Date
{
    ...
}
```

Function Declarations

Functions are declared in much the same manner as classes. However, the function name is always on the second line, and all other information about the function is on the first line. Function declarations are always two lines long. The declarations are in the following format:

```
[access] return_type
function_name( arguments )
```

Listing 7.7 shows a complete function declaration using our described style.

Listing 7.7. A well-styled function declaration.

```
public String
toFileString()
{
    String    retString = "";
    int       m = 1 + getMonth();
    int       d = getDate();
    int       y = getYear() % 100;

    if ( y < 10 )
        retString += "0";

    retString += y;

    if ( m < 10 )
        retString += "0";

    retString += m;

    if ( d < 10 )
        retString += "0";

    retString += d;

    return( retString );
}
```

Comments

Comments are a sore issue with many programmers. A friend of mine (Hey Bill!) used to tell me that the only purpose comments serve are to amuse the compiler. In some cases, he might be right. However, to better document the code in this book, comments are everywhere. Generally the comments are restricted to a single line, right above the line of code to which it pertains. This is common practice.

In addition to the single line comments, I like to distinguish chunks of code from others with the use of a block header. It serves no purpose other than to segregate blocks of code visually. Listing 7.8 shows an example of such segregation.

Listing 7.8. Comments really do improve the readability of your code!

```
//*********************************************************************
//* Package                                              &nb
sp;                        *
//*********************************************************************
```

```
package                         jif.util;

//*************************************************************************
//* Imports                                                 &nb
sp;                         *
//*************************************************************************

import                          java.util.Date;

//*************************************************************************
//* Date
                            *
//*************************************************************************

/**
 * An extension of the default Java Date.
 *
 * @see java.util.Date
 *
 * @version        1.00, 1 May 1996
 * @author        Jerry Ablan, munster@mcs.net
 */

public class FileDate
extends Date
{

//*************************************************************************
//* toFileString                                              &nbs
p;                      *
//*************************************************************************

    /**
     * This method returns the date in a format suitable for using in
     * file names. This format is YYMMDD, where YY is the year, MM is the
     * month, and DD is the day.
     *
     * @see java.util.Date#toString
     */

    public String
    toFileString()
    {
        String  retString = "";
        int     m = 1 + getMonth();
        int     d = getDate();
        int     y = getYear() % 100;

        if ( y < 10 )
            retString += "0";

        retString += y;
```

```
            if ( m < 10 )
                retString += "0";

            retString += m;

            if ( d < 10 )
                retString += "0";

            retString += d;

            return( retString );
        }
    }
```

Code Order

Finally, there is code order. Again, some of the Java books published have different orders. The oddest order I've seen is that all class variable declarations are below all of the function declarations. This is quite odd, and not at all like any other programming language. I've tried to keep the coding style similar to C and C++ for clarity.

All the source code in this book lays out in the following order:

1. Package name
2. Imports
3. Class declaration
4. Class members declaration
5. Class functions

All class functions lay out in a similar manner:

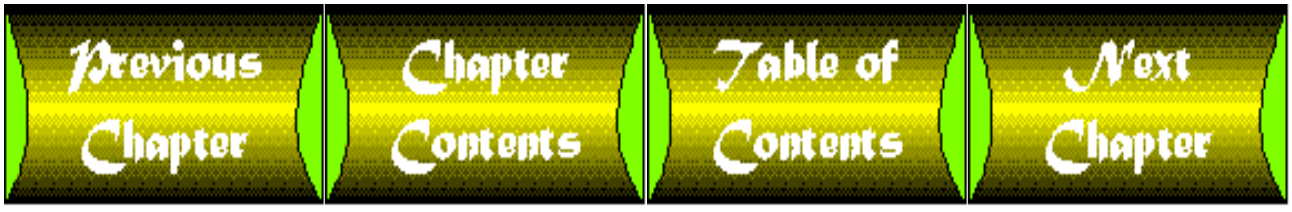
1. Function declaration
2. Function variables
3. Function code

Maintaining consistency throughout the code considerably improves readability.

Summary

This chapter introduces the four application design standards that are implemented in the next four chapters. These standards produce configuration processing, common logging, database connectivity, and consistent user interface. Through the use of these standards, the intranet application development will just fly by!

In the following chapters, the standards set forth in this chapter are implemented.



Chapter 8

Utility Classes

CONTENTS

- Introduction
- Timers
  - Timer Operations
  - Callbacks
  - Event Timers
  - Why Have Two Timers?
- Java Extensions
  - Extending Java's Date Class
  - Application Configuration Parameters
- Summary

---

utility \yü-'ti-l -te-\ n: *something useful or designed for use*

Introduction

During the process of creating any type of software, one usually ends up with many useful utilities. These utilities can range from simple functions to complex objects or data structures. Centralizing them where they are accessible to the entire application and can easily be moved to other projects is essential. Even Java has a utilities package called `java.util`; no doubt you are familiar with it.

Java intranet applications are no different. You'll develop many utilities during your development cycle. The utilities you create will be Java classes, however. This chapter will serve as a starting point for your utility class creation. In it you will find several classes that are quite useful for constructing intranet applications. You'll then learn how to keep all your classes together to easily use them again.

The types of utility classes that we'll explore in this chapter follow:

- Timers-These are like software alarm clocks. You set the alarm time, and they alert you later.
- Dates-Java's standard `Date` class is an excellent object. You'll look at how to tweak it just a bit to do some cool tricks.
- Properties-Java's `Properties` class is also an excellent object. You'll see how you can mold it to do your bidding.

Timers

A timer is a software mechanism designed to alert the program and programmer when a specified interval of time has elapsed. This mechanism is commonly used in all sorts of applications. A specific example of timer usage is in intranet database applications. You can use a timer to notify you whether certain events have or have not occurred.

For example, consider connecting to a database server. When your application attempts to connect with the server, your program and user must wait until the connection has been established. But how long do you wait? What if the server is down? What if the connection between your computer and the server is down? Many problems can occur, but if your program can be notified when a certain amount of time has elapsed, it can take action.

Another example of timer usage is for application monitoring. Perhaps you'd like to close any open server connections after a period of inactivity. When you set and reset a timer after certain events occur, your program can be notified of any inactivity.

Timer Operations

Timers work just like an alarm clock, which goes off when the alarm time has been reached. If you don't change the alarm time, the alarm will go off again in 24 hours.

Alarm clocks are far less versatile than timers, however. You cannot set the interval at which the alarm will sound; you're stuck with a 24-hour interval. If you want to set another time, you must constantly reset the alarm time. On the other hand, timers allow you to specify the interval of time between alarms, allowing you to generate constant alerts or signals to your program. You can then use these alerts however you'd like.

And how does your program receive these timer alerts? Your program can receive timer alerts in one of two ways: from a callback or from a Java event.

Callbacks

Java does not support the notion of pointers. With pointers, a conventional 3GL feature, you can pass the address of a method around in your program or from process to process so that other parts of your program, or other processes even, can call your method. This practice is commonly called a callback.

A callback is a way in which a program can register for notification when certain events occur. A real-world example of this is when you go to your book store and order a book that is not in stock. The clerk takes your name and phone number. Two weeks later, the store calls you on the telephone to notify you that your book has arrived. You've been called back.

A similar process takes place in programming. When certain programmatic or system events occur, you want to be notified of them. In other development environments, the callback is achieved by using pointers to methods. When you register for your notification, you inform the registrar of the method to call. The system then actually calls your method when the event occurs.

Look Mom, No Pointers!

As you've learned, Java has no pointers and, therefore, no method pointers. So how can you possibly create a callback mechanism? Easily-by using an interface. An interface in Java is like a template or hollow shell for classes. When a class implements an interface, it must "fill in" the methods defined in the interface.

When you define an interface for callbacks, any class that wants to receive them can implement this interface. The method that the receiving class implements is actually called by the timer object. Listing 8.1 shows an interface called `TimeOut`.

Listing 8.1. The `TimeOut` interface.

```
//*************************************************************************
//* TimeOut                                              &nb
sp;                        *
//*************************************************************************

public interface
TimeOut
{
```

```
//************************************************************************
//* timeOut                                                  &nb
sp;                        *
//************************************************************************

    public void
    timeOut( CallbackTimer timer );

}
```

The `TimeOut` interface defines a single method: `timeOut()`. A timer class can use this method to notify the user that a timer event has occurred. The next section explores a class called `CallbackTimer`, which uses this interface to implement a callback timer.

`CallbackTimer`

The `CallbackTimer` class uses the `TimeOut` interface to facilitate notification of timer events. When you create a `CallbackTimer`, you simply pass in the object that is the recipient of the notification as an argument. This passed-in object must be an implementor of the `TimeOut` interface. In addition, you must pass in the amount of time between notifications. The time interval is in milliseconds.

Tip
Milliseconds are 1/1000ths of a second; there are 1,000 milliseconds per second. Trying to keep seconds and milliseconds straight can be annoying, especially if your program uses both types at once. To help you remember, name your argument something that reminds you of the type. In our class, the argument is called `msInterval`.

Listing 8.2 shows the constructor of the `CallbackTimer`. The instance variables `myTarget` and `myInterval` hold the values for the rest of the object to use.

Listing 8.2. The `CallbackTimer` constructor.

```
//************************************************************************
//* CallbackTimer                                              &nb
sp;                    *
//************************************************************************

    public
    CallbackTimer( TimeOut target, int msInterval )
    {
        //    Save my values...
        myTarget = target;
        myInterval = msInterval;

        //    Indicate that this is not a user thread...
        setDaemon( true );
    }
```

The `CallbackTimer` class extends Java's `Thread` class. The reason for this is twofold. First and foremost, it is simple to implement (the best reason!). The `Thread` class provides much of the functionality needed for a notification process.

Second, with the `CallbackTimer`, you can create several timers and turn them off and on at will. After you create them, you can (and must) `start()` your timer, and whenever you want, you can `stop()` it. These methods are inherited from

the `Thread` base class.

Note the last line of the constructor:

```
setDaemon( true );
```

This `Thread` method tells the Java system that this thread is not a user thread. That means that the Java engine will not stop running until all user threads have died. If you have timers on in the background, your program will not end until you've stopped them all. The `setDaemon()` method informs the Java system that this thread is not a user thread. Therefore, if only nonuser threads are left running (that is, only timers), the system will stop your program immediately.

Implementing the callback mechanism is simple. Listing 8.3 shows the `run()` method from our `CallbackTimer` class.

Listing 8.3. The `run()` method from the `CallbackTimer`.

```
public void
run()
{
    //    Do this forever!
    while ( true )
    {
        try
        {
            //    Wait until next interval...
            sleep( myInterval );
        }
        catch ( InterruptedException ie ) {}

        //    Notify the target...
        myTarget.timeOut( this );
    }
```

Because the target of the callback implements the `TimeOut` interface, you are guaranteed that it will have a method called `timeOut()`. This way, you can call it when the timer goes off.

Let's create a Java program to illustrate the callback timer concept. The following fragments are part of the `TimerTester` application. The full source code, which can be found on the CD-ROM, creates 16 timers and displays the effect of them going off in 16 different windows. Let's walk through each part individually.

First, the declaration:

```
//*************************************************************************
//* TimerTester
;                       *
//*************************************************************************

public class
TimerTester
extends Frame
implements TimeOut
```

As you can see, you've implemented the `TimeOut` interface so that you can receive `timeOut()` callbacks.

You need a few instance variables for your program. The declarations follow:

```
//*************************************************************************
//* Members                                                 &nb
sp;                         *
```

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

        CallbackTimer[]        timers = new CallbackTimer[ 16 ];
        TextArea[]                showers = new TextArea[ 16 ];
```

The first variable, `timers`, is an array of 16 `CallbackTimer`s. The second variable is `showers` (not "showers"; "show-ers"). This array of 16 `TextArea` components will show information about the 16 timers you will create.

Your next constructor follows:

```
    //*************************************************************************
    //* Constructor
    ;                     *
    //*************************************************************************

        public
        TimerTester()
        {
            super( "Timer Tester!" );

            //    Create a panel for our grid display
            Panel p = new Panel();
            p.setLayout( new GridLayout( 4, 4 ) );

            //    Create some showers & timers...
            for ( int i = 0; i < 16; i++ )
            {
                showers[ i ] = new TextArea( 5, 10 );
                timers[ i ] = new CallbackTimer( this, 1000 * ( i + 1 ) );
                p.add( showers[ i ] );
                timers[ i ].start();
            }

            //    Add the timer panel to the frame...
            add( "Center", p );

            //    Pack and show the panels...
            pack();
            show();
        }
```

You've done several things in this constructor. Let's take a look at them step by step:

1. Call your base class constructor. First you must call your base class's constructor to initialize your base class's instance variable.
2. Create a `Panel` with a `GridLayout`. The grid is set at 4 rows high by 4 columns wide, which will lay out your 16 `TextArea`s in a nice grid format.
3. Next, create the array elements for your instance variables. The best way to do this is with a loop. You use a `for` loop to create each component. The interval for the timer is set to the iteration count times 1000. This gives you 16 timers ranging in intervals from 1 to 16.
4. Also within the loop, you add the newly created `TextArea` to your `GridLayout` and start up each timer.
5. Then add the panel to the center of the `Frame`.
6. Finally, `pack()` the layout and `show()` the window.

The final important part of the `TimerTester` example is the `timeOut` method that implements the `TimeOut` interface,

as shown in the following code.

```
//*************************************************************************
//* timeOut                                                  &nb
sp;                          *
//*************************************************************************

    public void
    timeOut( CallbackTimer whichTimer )
    {
        for ( int i = 0; i < TIMER_COUNT; i++ )
        {
            if ( timers[ i ] == whichTimer )
            {
                showers[ i ].appendText( "I ticked!\n" );
                return;
            }
        }
    }

}
```

The `timeOut` method is called when a timer goes off. In the example, however, you have 16 different timers. How do you know which one you're dealing with?

You use a `for` loop to compare each element in your array with the argument passed. The `CallbackTimer` class sends itself as an argument to the `timeOut` method. This way you can find out which timer it is.

Note
The alternative and much more object-oriented method of displaying the timer results in a window creates a new class that extends the `TextArea` component. This class could have a `CallbackTimer` as an instance variable set at construction. That way you are dealing with only a single timer and its display is itself.

Figure 8.1 shows the `TimerTester` program in action.

Figure 8.1 : *The* TimerTester *program output.*

Event Timers

Another method of receiving timer notifications is by using the Java event system, provided to you by the `EventTimer` class.

The `EventTimer` Class

The `EventTimer` class is similar to the `CallbackTimer` class in that it is also a timer. The notification method is completely different, however. The `EventTimer` class sends an `ACTION_EVENT` event to its owner when the timer goes off. What follows is the same example given earlier, except it uses the `EventTimer`.

The `EventTimer` declaration is similar to the `CallbackTimer` `TimerTester` example program except that classes using the `EventTimer` class do not need to implement the `TimeOut` interface. This is because the class that owns the `EventTimer` object will receive notifications through the Java event system. The following code is the declaration for the `EventTimerTester` program.

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

```
//* EventTimerTester
                  *
//*************************************************************************

public class
EventTimerTester
extends Frame
```

Notice that the preceding code does not implement any interfaces.

The constructor method of the `EventTimerTester` program is identical to that of `TimerTester`. In fact, the entire program is pretty much identical to the `TimerTester` program except that instead of receiving notifications via the `timeOut()` method, the `EventTimerTester` program receives its notifications via the `action()` event handler. The source code follows:

```
//*************************************************************************
//* action                                          &nbs
p;                          *
//*************************************************************************

    public boolean
    action( Event event, Object arg )
    {
        //    Locate the timer that has ticked...
        for ( int i = 0; i < TIMER_COUNT; i++ )
        {
            //    Is this the one?
            if ( timers[ i ] == arg )
            {
                //    Show it in the window...
                showers[ i ].appendText( "I ticked!\n" );
                return( true );
            }
        }

        return( false );
    }
```

Why Have Two Timers?

You might wonder why you're using two timer classes if one is enough. One is enough, but sometimes you might want the flexibility of two.

For example, the `CallbackTimer` is excellent for use within other classes. If you have a class that requires some sort of timer functionality, a `CallbackTimer` as an instance variable is a cool thing. The timer is fully integrated into your object and the consumer or user of your object does not need to worry about using it.

The `EventTimer` is excellent for on-the-fly timers. It's also good for adding timers to existing code when you don't want to modify the implementations of some classes. It can be used as a plug-and-play object.

And, because the bytecode for the two timers together is about 2KB, I'm not too worried about insufficient disk space.

Java Extensions

The second half of this chapter is devoted to Java extension classes. These classes extend, or augment, the functionality that the core Java classes provide. Although this base functionality is useful, you might like these objects to perform other tricks for your intranet applications. Two classes that you extend here are the `Date` and `Properties` classes.

Extending Java's `Date` Class

The first class you'll extend is Java's `Date` class. Although this class has many features, let's add a few more that will help in building intranet applications. The big change here is a string formatting method. You can configure this method in terms of how it is constructed and which delimiters to use. You'll call your new class `FileDate` because the augmentations will provide nice features when you create text files on disk.

To make this class robust and able to handle a variety of date formats, you need to create input and output methods. These methods will be the central point for date parsing and date string construction, giving you a single point to make changes if you ever need to implement new date formats or remove an existing format.

Parsing the Date Within a String

Java's `Date` class comes complete with a method called `parse()` to strip apart a string and convert it into a date. Although this method can handle several formats, it is very specific about which formats will work. If you look at the actual source code to the `Date.parse()` method in the JDK, you'll see that it is very difficult to follow. You'll take a much simpler approach. For your applications, you want to parse dates that are in the format of `MM<d>DD<d>YY[YY]`, where the `<d>` is a delimiter of the user's or programmer's choosing.

To quickly and easily parse the date from a string, you can use Java's `StringTokenizer` class. This breaks up the date string into tokens and you can then easily digest and regurgitate these tokens into anything you'd like. The following method, `valueOf()`, takes a string representing a date and parses it into its three components: month, day, and year. It looks like this:

```
//**************************************************************************
//* valueOf                                                &nb
sp;                        *
//**************************************************************************

    public static FileDate
    valueOf( String s )
    throws IllegalArgumentException
    {
        StringTokenizer     st = new StringTokenizer( s, "/.-," );
        String                 ms, ds, ys;

        try
        {
            ms = st.nextToken();
            ds = st.nextToken();
            ys = st.nextToken();
        }
        catch ( java.util.NoSuchElementException e )
        {
            throw new IllegalArgumentException();
        }

        int m = Integer.parseInt( ms );
```

```
            int d = Integer.parseInt( ds );
            int y = Integer.parseInt( ys );

            //    Convert four digit year to Java year...
            if ( y > 1900 )
                y -= 1900;

            return( new FileDate( y, m - 1, d ) );
        }
```

The method is declared `static` so that you can use it as a conversion method. With conversion methods, you can use features of a class without actually instantiating it. A good example of this is Java's `Math` class. All of its methods are `static` and all are called by prepending the class name (`Math`) to the method. Another example of a `static` method is used in the preceding parsing routine. When you use the `parseInt()` method of the `Integer` class, you are calling a `static` method.

Because this method creates and returns a new object, you'll need to throw the `IllegalArgumentException` if you are given bad data. These are commonly called factory methods because they build objects.

What Does `static` Really Mean?

When you declare a method or instance variable `static` in a Java class, you tell the compiler that that method or variable is the only copy for all instantiations of that class. This allows some special privileges and revokes some as well. For example, while gaining the ability to perform operations without actual instantiation, you can't reference any non-static member of your class. This can be annoying at times.

The `static` class methods are well suited for conversion or factory functionality and `static` instance variables are excellent for counters. You could create an instance counter that increments each time your class is created and decrements when the class is destroyed. If you declare your counter `static`, only a single copy will exist for all instances of your class.

The next step is to create a `StringTokenizer` for the string that was passed into your method. This tokenizer should recognize the following delimiters for dates: "`/`", "`-`", "`.`", or "`,`". You specify this as the second argument in the `StringTokenizer`'s constructor. You then attempt to pull out three tokens from the tokenizer. If this fails, you know you don't have a good date, and you throw your exception.

After you've received three good tokens from your main string, you simply convert them to integers. These integers are used to create a new instance of `FileDate`, which is returned.

Constructing a String with a Date

The second important method in this class is the output method, which will be responsible for creating a `String` from the date and returning it to the caller. This method must be able to handle a variety of formats easily and without too much effort. The more code you write, the more likely you are to make errors.

First, you need to establish the date formats that you'll support. Use three major formats: MDY, DMY, and YMD. MDY is the standard U.S. date format. DMY is common in Europe and favored by some in the U.S. The last format, YMD, can be used to format dates in a manner that will sort alphabetically. In addition to these formats, the option of returning the year in four digits must be available.

You also want your date formatter to return the month name instead of the month number. This can be used when talking to Oracle databases, for instance. The default date format accepted by Oracle is `DD-MMM-YY[YY]`.

You'll handle these formatting options with constants. Constants in Java are declared `final` and `static`. Your declarations follow:

```
        public final static int         MDY = 0;
        public final static int         DMY = 1;
        public final static int         YMD = 2;

        public final static int         MMMDY = 3;
        public final static int         DMMMY = 4;
        public final static int         YMMMD = 5;

        public final static int         MDYYYY = 6;
        public final static int         DMYYYY = 7;
        public final static int         YYYYMD = 8;

        public final static int         MMMDYYYY = 9;
        public final static int         DMMMYYYY = 10;
        public final static int         YYYYMMMD = 11;
```

These constants mimic the order of the dates they represent. This is easy to remember from a coding standpoint and easy to debug if there is a problem. The constants with `YYYY` in them represent four-digit years. The constants with `MMM` in them represent the non-numeric months.

In one instance variable, you need to format the month into a string. This is an array to hold the names of the months, which is declared as follows:

```
    static String               monthNamess[] = { "JAN", "FEB", "MAR",
    "APR",
        "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC" };
```

This array is easily accessed with a month variable. It is declared static because, as you know, month names never change.

Implementing this formatter is simple. You'll take the year, month, and day and create strings of them, padding them as necessary. When you have the three elements as strings, the rest is a cake walk. Depending on the format, you concatenate the three parts together, separated by the delimiter selected. The final result is returned to the caller.

The source code for your formatter method follows:

```
        public static String
        formatDateToString( int y, int m, int d, String delimiter, int fmtOpt
    )
        {
            String    ms, ys, ds;

            //    Build the month and day strings...
            ms = ( m < 10 ) ? "0" + Integer.toString( m ) : Integer.toString(
    m );
            ds = ( d < 10 ) ? "0" + Integer.toString( d ) : Integer.toString(
    d );

            //    Convert the year to four digit if requested...
            if ( fmtOpt == MDYYYY || fmtOpt == DMYYYY || fmtOpt == YYYYMD ||
                fmtOpt == MMMDYYYY || fmtOpt == DMMMYYYY || fmtOpt ==
    YYYYMMMD )
            {
                if ( y < 1900 )
                    y += 1900;
```

```
            }
            else
                //    Force it to two digits...
                y %= 100;

            //    Finally build the year string...
            ys = ( y < 10 ) ? "0" + Integer.toString( y ) : Integer.toString(
    y );

            //    Build the return string...
            switch ( fmtOpt )
            {
                case MDY:
                case MDYYYY:
                    return( ms + delimiter + ds + delimiter + ys );

                case MMMDY:
                case MMMDYYYY:
                    return( monthNames[ m - 1 ] + delimiter + ds +
                        delimiter + ys );

                case DMY:
                case DMYYYY:
                    return( ds + delimiter + ms + delimiter + ys );

                case DMMMY:
                case DMMMYYYY:
                    return( ds + delimiter + monthNames[ m - 1 ] +
                        delimiter + ys );

                case YMD:
                case YYYYMD:
                    return( ys + delimiter + ms + delimiter + ds );

                case YMMMD:
                case YYYYMMMD:
                    return( ys + delimiter + monthNames[ m - 1 ] +
                        delimiter + ds );
            }

            //    Unknown format option? Return MDY...
            return( ms + delimiter + ds + delimiter + ys );
        }
```

This method has been declared `static` so that it can be used as a conversion function or by an instantiation of the class.

One feature available in other languages but not in Java is a string formatting function. For you C and C++ programmers out there, I'm talking about `printf` functionality. With these functions, you usually can create a string of placeholders for various elements. The placeholders define the size of the substring in addition to padding. A set of arguments to put into those placeholders follows all of this. Unfortunately, Java does not have the capability to accept an unknown number of arguments to a function, which is required to provide general string formatting.

Java's `String` class is an awesome beast, however. It is well implemented and supports the `+` operator for concatenation. Although it's not the perfect solution, it will fit your needs just fine. By checking to see whether the month or day is less

than 10, you know to add a leading 0.

Tip
Notice that when the requested year is not in four digits, you modulus the value with 100. This is a neat and necessary little trick to always keep your year in the range of 0 to 99. When you reach the year 2000, the Java `Date` will return 100 or more because the Java `Date` class represents the year as an offset from 1900. Any year over 1999 will result in a number greater than 100. By using the modulus operator, you can ensure that your software will work well into the 21st century.

You also need to provide some alternative access methods to the `formatDateToString()` method. I have created some, but you might want to create more. Those that I have created follow.

This default date formatter will create a date of `MM/DD/YY` from the current value of the `FileDate` object:

```
public String
formatDateToString()
{
    return( formatDateToString( "/", MDY ) );
}
```

You might select the delimiter and formatting option, however, which is similar to the default:

```
public String
formatDateToString( String delimiter, int fmtOpt )
{
    return( formatDateToString( getYear(), getMonth() + 1, getDate(),
        delimiter, fmtOpt ) );
}
```

This default date formatter will create a date of `MM/DD/YY` from the value parsed from string `s`:

```
public static String
formatDateToString( String s )
{
    return( formatDateToString( s, "/", MDY ) );
}
```

This method parses string `s` and selects a delimiter. It defaults to `MDY` order:

```
public static String
formatDateToString( String s, String delimiter )
{
    return( formatDateToString( s, delimiter, MDY ) );
}
```

This method parses string `s` and selects a delimiter and an order:

```
public static String
formatDateToString( String s, String delimiter, int fmtOpt )
{
    FileDate date;

    try
    {
        date = FileDate.valueOf( s );
    }
```

```
            catch ( IllegalArgumentException e )
            {
                return( "" );
            }

            //    Send a blank delimiter...
            return( formatDateToString( date.getYear(), date.getMonth() + 1,
                date.getDate(), delimiter, fmtOpt ) );
        }
```

This default date formatter will create a date of `MM/DD/YY` from the values passed in:

```
        public static String
        formatDateToString( int y, int m, int d )
        {
            return( formatDateToString( y, m, d, "/", MDY ) );
        }
```

This method uses the passed in values and allows the selection of a delimiter. It defaults to `MDY` order:

```
    public static String
        formatDateToString( int y, int m, int d, String delimiter )
        {
            return( formatDateToString( y, m, d, delimiter, MDY ) );
        }
```

Application Configuration Parameters

The second Java class that you will create for your purposes is the `ConfigProperties` class. It extends the Java `Properties` class to add some unique functionality.

The intranet applications that you build can become quite complex, which only results in more testing when you implement changes. One excellent way to add functionality to complex programs without programming is by using configuration files. The more you are able to configure your program, the more useful it is.

Configuration files are not a new idea; they have been around for years. Many programmers avoid using them because of the parsing involved-parsing each line and figuring out what the configuration line means. Java makes this really easy for you to do, so you should take advantage of it.

Application configuration parameters, or properties, can come from two places: the command line and a configuration file. You want your class to provide a unified, or merged, set of properties. To do this, your class should accept an array of properties stored in strings as input. This array will merge with the array read from the configuration file stored on disk. The resulting combination represents all the properties.

The `ConfigProperties` Class

The declaration for your extension is as follows:

```
    //*************************************************************************
    //* ConfigProperties
                    *
    //*************************************************************************

    public class
    ConfigProperties
    extends Properties
```

`ConfigProperties` simply extends the Java `Properties` object. The real functionality is in the constructor, which

for this object accepts two parameters: an array of property strings and a filename.

Property strings are strings in the format:

```
key = value
```

Here, `key` is the name associated with the value. `key` and `value` are standard Java property strings that can be passed in on the command line to any Java application or applet with the `-D` option. For example, to set the property `head.size` to `large` in the application `InflateHead`, the command line would be as follows:

```
java -Dhead.size=large InflateHead
```

This and any other arguments are passed into your application's `main` method as an argument. If you pass this array of strings onto the `ConfigProperties` object, it will be merged in with the configuration parameters in the configuration file.

The property string array passed in is parsed into a temporary `Properties` object, as follows:

```
//*************************************************************************
//* parseArguments                                                        &n
bsp;                   *
//*************************************************************************

    public int
    parseArguments( String args[] )
    {
        int        i = 0;

        if ( args != null )
        {
            for ( i = 0; i < args.length; i++ )
            {
                StringTokenizer st = new StringTokenizer( args[ i ], "="
);

                if ( st.countTokens() == 2 )
                    argProperties.put( st.nextToken(), st.nextToken() );
            }
        }

        return( i );
    }
```

You use the `StringTokenizer` to break up the strings into their key and value components. The components are then added to the `argProperties` object, which is an instance variable of the class. After the argument properties are parsed and stored, the configuration file is read in and the argument properties are merged in.

An important design decision was made for this class. The configuration file properties are replaced by any matching argument properties. This is standard operating procedure. For the most part, command line parameters should always override any stored functionality, including configuration file parameters. This class implements that philosophy by adding the argument properties to the configuration properties after they have been read from disk.

The Configuration File

The configuration file is nothing more than a file containing lines of the key and value pairs. Listing 8.4 shows a sample configuration file.

Listing 8.4. A sample configuration file.

```
#
# Employee.cfg
# Employee maintenance configuration file
#
Title=Employee Files
user=dia_user
password=dia
server=dia_database
```

Use this file to store user names, titles, servers, or just about any parameter that will help the user to better control your programs. You'll use these files in your sample intranet applications later on in the book.

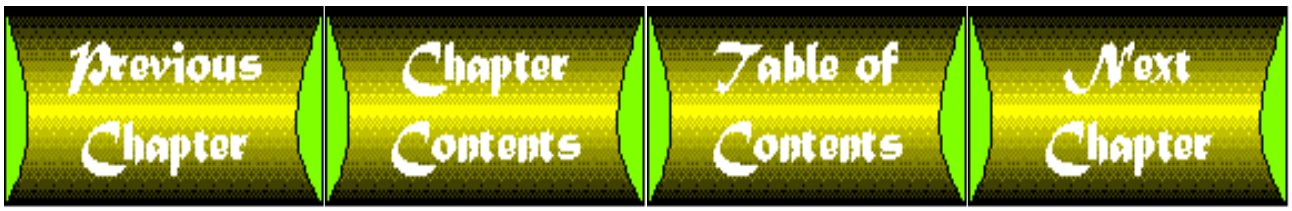
In Chapter 12, "Putting Them All Together," you'll see how you can use this class to provide a unique Java applet function to your intranet applications.

Summary

You've come to the end of your first foray into extending Java for intranets. Hopefully your interest is piqued. This chapter introduced you to some utility classes that will be useful when you develop intranet applications. These classes included timers, date formatters, and a class to help configure your applications.

You now should have a decent understanding of these classes and how you can use them. I showed example programs along with their output, and you can peruse and run the source code on the CD-ROM.

In Chapter 9, "Logging Classes," you will learn all about writing text to a window and to disk files. That chapter is all about logging-and not the kind that lumberjacks do!



Chapter 9

Logging Classes

CONTENTS

- Introduction
- The Log
  - The Log Entry
  - The Log Interface
- The Logging Classes
  - DiskLog
  - ScreenLog
- A Sample Logging Program
- Summary

---

log \'log\ n: *any record of performance*

Introduction

The output of your programs is important to you, especially if there are errors. The standard Java outputting mechanism is just fine for UNIX; it can be captured and redirected to files easily. But for Windows and Macintosh, it doesn't cut the mustard.

The biggest problem with output on the Windows and Macintosh platforms is that once your application has finished running, the output disappears. This is very annoying and does not help you in any way. Your intranet applications require much better log facilities than are available. You'll need nice logs that are easy to search. These logs are important for tracking errors and finding problems with your intranet and the applications that run on it.

This chapter will present two classes that implement a log file strategy that can be used in all of your applications. This strategy includes a common log file format and centralized log location. Finally, you'll take this log strategy even farther and apply it to Java's own output mechanism. This can be useful for Java applets or for running your applications on diskless workstations.

The Log

Logs and logging have been around since prehistoric times. The drawings left on cave walls by early man are logs of his day. After that, logs were used to record daily life in many ancient civilizations. Then, logs were used to record the voyages of sea captains. The versatility of the log is unbelievable, so why not utilize some of its charm in your own software?

You should, and you can. But writing log files with Java can be a hassle. If every time you create an application you have to perform the same initialization for logging, you'll soon not use your log for quick-and-dirty programs. But by making it simple, and providing an interface to the logging mechanism that is easy to use, logging becomes automatic and painless.

But what should you put into your log? How can you make it easy to read and search? Read on.

The Log Entry

A log entry is nothing more than a line of text documenting an event. The event might be a debug message or perhaps an error. It really doesn't matter what you're documenting because it is probably important to you. But your only real log choice with Java is writing information to the screen.

But writing information to the screen in Java is not the best solution for logs. The main reason for this is that if you are running from a window, the output disappears upon completion of your program. This is where the log classes come in. If you can use these classes to write your output, it will not disappear or be lost. It will be stored forever in a disk file.

The classes write log entries in a standard format. The format is as follows:

```
appName|date|user|level|text
```

where

appName is the name of the application that created the entry.

date is the date of the entry.

user is the user running the application.

level is an indicator of the severity of the log entry.

text is the text of the log entry.

Listing 9.1 lists a sample log file output.

Listing 9.1. Sample log file entries.

```
Employee|960609|munster|I|Application [Employee] started at Sun Jun 9
11:22:07
Employee|960609|munster|I|Application [Employee] ended at Sun Jun 9
11:22:18
Employee|960609|munster|I|Application [Employee] started at Sun Jun 9
12:24:57
Employee|960609|munster|I|Application [Employee] ended at Sun Jun 9
12:25:15
```

Each portion of the log entry is separated by the pipe character (`|`). This is a standard UNIX delimiter. It doesn't appear often in normal text and is easily spotted.

Nothing else in the entry needs explaining except the level indicator. This indicator allows you to place an importance level on your log entries. Afterwards, you can filter out the levels that you are interested in.

The levels and the interface to the logging mechanism are defined by the `Log` interface.

The Log Interface

The `Log` interface provides the template or pattern for all logging objects. There are two important parts to the `Log` interface. The first is the level indicators.

Log Level Indicators

There are five defined log level indicators. They are as follows:

- `D`: Debug
- `I`: Informational
- `W`: Warning

- E: Error
- F: Fatal

Each indicates the level of importance, or severity, of the log entry it is applied to. Debug is, of course, the least severe of all entries. Fatal is the most severe. The rest are all in between. As you code, choose the severity level that fits the situation.

Note
The log levels mean absolutely nothing to the program. They are simply a way for you, the programmer, to identify the severity of an event.

The severity levels are defined as `char`s and are declared `static final`. Here is the definition of the log level indicators as found in the `Log` interface:

```
public static final char     DEBUG = 'D';
public static final char     INFO = 'I';
public static final char     WARNING = 'W';
public static final char     ERROR = 'E';
public static final char     FATAL = 'F';
```

The uppercased word is the constant value that you use when indicating log levels. These levels are used by passing it along to the logging method. The logging method will then store it into the log file. The logging method is the second part of the `Log` interface.

The `log` Method

The `Log` interface defines a single method called `log`. It is as follows:

```
//*************************************************************************
//* log                                                   &
nbsp;                                *
//*************************************************************************

    public void
    log( char logLevel, String logEntry )
    throws IOException;
```

This method must be implemented by users of this interface. The `log` method accepts, as arguments, a log level and a log entry. The rest of the log implementation is up to the implementor of this interface. I've chosen to implement this method to produce log files with entries such as the ones shown in Listing 9.1.

The Logging Classes

There are two classes that provide standard logging, as described earlier in this chapter. They are `DiskLog` and `ScreenLog`. `DiskLog` writes all log entries to a disk file. The `ScreenLog` writes all the log entries to the screen or a window. The `ScreenLog` is useful in cases where perhaps a `DiskLog` cannot be created, or is not needed.

When a `DiskLog` object is created, it opens the log file and moves to the end of the file, never overwriting what is currently stored inside. This gives you a persistent log file. It also allows several processes to share the same log file. Instead, the `DiskLog` object by default creates a new log file each day.

Take a closer look at the two classes.

DiskLog

The DiskLog class extends Java's RandomAccessFile class. The RandomAccessFile class is the only Java file class capable of not deleting the contents of a file when opened. This allows you to create a persistent disk log file. The declaration of the DiskLog class is as follows:

```
//************************************************************************
//* DiskLog                                                  &nb
sp;                        *
//************************************************************************

public class
DiskLog
extends RandomAccessFile
implements Log
```

As you can see, you extend RandomAccessFile and implement the Log interface.

Log File Names

Log file names can be anything you want. However, because you want your applications to share a common log file, a single log file should be used. To achieve this, the DiskLog class names its log files syslog.YYMMDD where YYMMDD is the date of the log file. This is the default name. If you do not choose a name, this is the one that will be used. You can always override the default and select your own log file name.

By default, this log file is stored in the directory where your program is located. There are ways to override these mechanisms in the constructors.

DiskLog has three constructors. All of them require the name of the application that is going to be using it. This is because the application name is the first thing the log entry contains. The other two constructors accept a filename, overriding the default name, and a path name, overriding the default location of the log file. I'll discuss each one in detail.

The first constructor creates a default log filename in the current directory. Here is the call:

```
public
DiskLog( String name )
throws IOException
```

The second constructor accepts, as input, the log filename and the application name. This is one way to override the default name given to a log file. The file, however, still resides in the current directory. Here is the constructor call:

```
public
DiskLog( String logName, String name )
throws IOException
```

The last constructor accepts the path to the log file, the name of the log file, and the application name. It combines the path- and filenames together to make a complete path to the file. Here is the constructor code:

```
public
DiskLog( String logDir, String logName, String name )
throws IOException
```

Tip
Note that all three constructors throw the IOException. This is because you've extended the RandomAccessFile class. In your constructor, you call the base class's constructor. Whenever you override a constructor of a base class, you must throw the same exceptions that the base class throws.

Making a Log

Opening a log file is as simple as constructing an object in Java. The following code example from the LogTester sample program illustrates the construction technique for the `DiskLog` class:

```
//    Create a disk log...
try
{
    dLog = new DiskLog( "Log Tester" );
}
catch ( IOException e )
{
    sLog.log( Log.ERROR, "Could not create the DiskLog object [" +
        e.toString() + "]" );
}
```

First you `try` to create a new `DiskLog` and assign it to the variable `dLog`. If it fails and `IOException` is thrown, the `sLog` (an instance of `ScreenLog`) shows the error.

Now all you have to do is write to your new log file.

Writing to the Log

To write an entry to the log, `DiskLog` provides a single method: `log`. This method is the implementation of the `Log` interface's abstract method `log`. It accepts two parameters as input: a
logging level and the text of the log entry. The `log` method constructs a log entry from the information passed and what it has, and writes it out to the log file.

There is a method available for you to specify what to do with log entries should the primary log facility fail, such as a log entry on another disk drive, or a log on the screen. Whatever the case, the `setBackupLog()` method accepts any class that implements the `Log` interface as input and stores this when writing to the primary log fails. You'll see this implemented below. The following is the source code for the `log` method:

```
//************************************************************************
//* log                                                  &
nbsp;                          *
//************************************************************************

    public void
    log( char logLevel, String logEntry )
    {
        String    logLine = appName + "|";

        //    Use the jif.util.FileDate for this new method...
        logLine += ( new FileDate() ).toFileString() + "|";
        logLine += System.getProperty( "user.name" ) + "|";
        logLine += logLevel + "|";
        logLine += logEntry;

        //    Let the system define how lines are terminated...
        logLine += System.getProperty( "line.separator" );

        //    Write it out to disk...
        try
        {
            writeBytes( logLine );
```

```
            }
            catch ( IOException e )
            {
                if ( backupLog != null )
                {
                    try
                    {
                        //    Write to backup...
                        backupLog.log( Log.ERROR, "Error [" + e.toString() +
                            "] writing the following entry to the log file:"
    );
                        backupLog.log( logLevel, logEntry );
                    }
                    catch ( IOException e2 )
                    {
                        //    Write to screen
                        ScreenLog sl = new ScreenLog();
                        sl.log( Log.ERROR, "Backup log failed [" +
                            e2.toString() + "] writing the following entry:"
    );
                        sl.log( logLevel, logEntry );
                    }
                }
                else
                {
                    //    Write to screen!
                    ScreenLog sl = new ScreenLog();
                    sl.log( Log.ERROR, "Disk log failed [" + e.toString() +
                        "] writing the following entry:" );
                    sl.log( logLevel, logEntry );
                }
            }
        }
```

Basically, you construct a string called `logLine` using the input received plus some additional system information. The `log` method then throws a linefeed or carriage return/linefeed on the end and writes it out to the file. Don't forget that the `DiskLog` extends the `RandomAccessFile` class; therefore, the `writeBytes()` method is inherited.

Around the `writeBytes()` method you wrap a try/catch clause. You do this so the user of this class doesn't have to. If an error occurs while writing a log entry to disk, try to write the entry to the backup log. If writing to the backup fails, write it to the screen. If no backup log is defined, write the error to the screen. No matter what, the log entry will be shown somewhere.

Closing the Log

The log file closes by itself at shutdown. However, you might want to close the log a little sooner. There is no specialized code in the `DiskLog` object to close a disk log. However, you might call the `close` method of the `RandomAccessFile` class to close a disk log.

ScreenLog

The `ScreenLog` class is identical to the `DiskLog` class in output. However, there are a few slight differences. The first variation of the `ScreenLog` writes standard log lines to the screen. Here is the constructor:

```
    public
```

```
ScreenLog( String name )
```

This takes the name of the application for the log entries as input. Because this log type displays to the screen, there is no need for a disk filename or path. Figure 9.1 shows the output of a `ScreenLog`.

Figure 9.1 : *The output of the* ScreenLog *class to the screen*

The second type of `ScreenLog` creates a `Frame` window and displays all of the log entries in it. The constructor for this type takes two input arguments. The first argument is the title to show on the log's window. The second argument is a Boolean value that denotes whether or not to show the window.

Here is that constructor:

```
public
ScreenLog( String windowTitle, boolean popup )
```

The output is written to a `Frame` window. The output is shown in Figure 9.2.

Figure 9.2 : *The output of the* ScreenLog *class to a window.*

A Sample Logging Program

The following is the LogTester sample program. It is also included on the CD-ROM.

```
//*************************************************************************
//* Imports                                                 &nb
sp;                          *
//*************************************************************************

//    JIF Imports...
import                             jif.util.*;
import                             jif.awt.*;
import                            jif.log.*;

//    Java Imports...
import                            java.io.*;
import                            java.awt.*;

//*************************************************************************
//* LogTester                                                 &
nbsp;                        *
//*************************************************************************

public class
LogTester
extends Frame
{

//*************************************************************************
//* Members                                                 &nb
sp;                          *
//*************************************************************************

    ScreenLog                     sLog;
    ScreenLog                     wLog;
    DiskLog                         dLog;
```

```
    //************************************************************************
    //* main
                                *
    //************************************************************************

        public static void
        main( String args[] )
        {
            new LogTester( args );
        }

    //************************************************************************
    //* Constructor
    ;                         *
    //************************************************************************

        public
        LogTester( String args[] )
        {
            super( "Log Tester!" );

            //    Create a screen log...
            sLog = new ScreenLog( "Log Tester", true );
            wLog = new ScreenLog( "Log Tester", false );

            //    Create a disk log...
            try
            {
                dLog = new DiskLog( "Log Tester" );
            }
            catch ( IOException e )
            {
                sLog.log( Log.ERROR, "Could not create the DiskLog object ["
    +
                        e.toString() + "]" );
            }

            //    Tell our disk log that the screen is the backup...
            dLog.setBackupLog( sLog );

            sLog.log( Log.INFO, "Log object created" );
            dLog.log( Log.INFO, "Log object created" );
            wLog.log( Log.INFO, "Log object created" );

            //    Pack the panels...
            pack();

            sLog.log( Log.INFO, "Layout packed" );
            dLog.log( Log.INFO, "Layout packed" );
            wLog.log( Log.INFO, "Layout packed" );

            show();
```

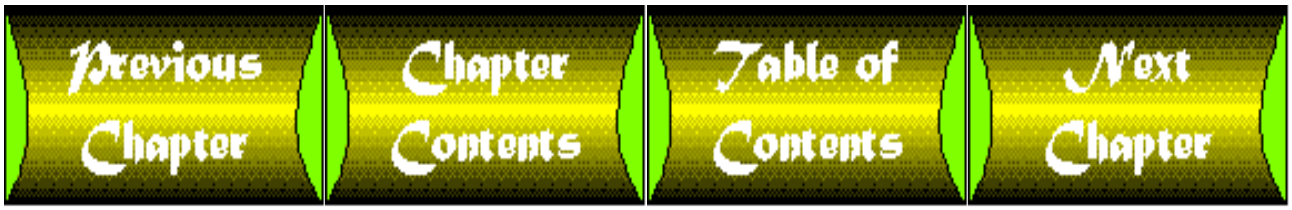
```
            resize( 100, 100 );

            sLog.log( Log.INFO, "Running!" );
            dLog.log( Log.INFO, "Running!" );
            wLog.log( Log.INFO, "Running!" );
        }

    //************************************************************************
    //* handleEvent
    ;                     *
    //************************************************************************

        public boolean
        handleEvent( Event anEvent )
        {
            if ( anEvent.id == Event.WINDOW_DESTROY )
            {
                hide();
                dispose();
                sLog.log( Log.INFO, "Exiting!" );
                dLog.log( Log.INFO, "Exiting!" );
                wLog.log( Log.INFO, "Exiting!" );
                System.exit( 0 );
            }

            //    I didn't handle it, pass it up...
            return( super.handleEvent( anEvent ) );
        }

    }
```

The program is quite simple. You create a log of each possible type: a screen log in a window, a screen log that goes to the screen (`stdout`), and a disk log. Various things are written to each of the logs. The output of this program is shown in Figures 9.1 and 9.2.

Summary

You've come to the end of your second extension discussion. By now you should have a good understanding of what a log is and how it can help you in your programming. You've been introduced to the `DiskLog` and the `ScreenLog` classes, and given a little background on how they work.

In Chapter 10, "Database Classes," you will find information about extending Java to communicate with database servers, as well as reading and writing data.



Chapter 10

Database Classes

CONTENTS

- Introduction
- JDBC in Depth
  - The DriverManager Class
  - The Driver Class
  - The Connection Class
  - The Statement Class
  - The ResultSet Class
  - A JDBC Sample Program
- Making JDBC Easy to Use
  - The Connector Interface
  - The SQLFactory Interface
- The Classes
  - The DBConnector Class
  - OracleSequence
- Summary

---

da ta base \da'te-bas' \ n: *a store of data*

Introduction

Now that you have some of the easier classes under your belt, it is time to start digging into some more advanced issues. The main thrust of this chapter is to introduce you to some interfaces and classes that will help your intranet applications communicate with databases.

This chapter builds on the classes in Chapter 8, "Utility Classes," and Chapter 9, "Logging Classes." You'll also build on Sun's own JDBC classes for Java to provide enhanced functionality and a higher level of access for your applications.

This chapter will take two forks or paths. The first is the database connection path. You will design and create a connection class that you can use to connect to almost any JDBC data source. You will use this connection class throughout the rest of the book and in the applications developed in later chapters.

The second path is that of database maintenance. Writing SQL statements to update your database is a dull and dreary task, but as you'll see, you can automate the task just a little, making this bogus task almost fun.

But before you even start talking about new classes, let's take a look deep inside JDBC.

JDBC in Depth

If you're not familiar with JDBC, please go back and read the JDBC section of Chapter 6, "Database Connectivity Options," where you will find a thorough overview of JDBC. In this chapter, you will investigate the classes that make up the JDBC class hierarchy and how to use them. For your convenience, the JDBC class hierarchy is shown again in Figure 10.1.

Figure 10.1 : *The JDBC class hierarchy.*

JDBC is a rich set of classes that give you transparent access to a database with a single application programming interface, or API. This access is done with plug-in platform-specific modules, or drivers. Various database manufacturers provide these drivers. Using these drivers and the JDBC classes, your programs will be able to access consistently any database that supports JDBC, giving you total freedom to concentrate on your applications and not to worry about the underlying database.

The JDBC class hierarchy lives in the `java.sql` package. As you can see in Figure 10.1, many of the classes in the JDBC hierarchy are `abstract`. It is up to the database vendor to provide implementations of these classes for its customers.

All access to JDBC data sources is done through SQL. Sun has concentrated on JDBC issuing SQL commands and retrieving their results in a consistent manner. Though you gain so much ease by using this SQL interface, you do not have the "raw" database access that you might be used to. With the classes you can open a connection to a database, execute SQL statements, and do what you will with the results, however. Don't forget that SQL statements can create database objects such as tables, views, synonyms, triggers, stored procedures, and so on. Don't think that JDBC is limited.

The `DriverManager` Class

The cornerstone of the JDBC package is the `DriverManager` class. This class keeps track of all the different available database drivers. For you Bozo fans out there, think of it as Ringmaster Ned. The `DriverManager` manages loading and unloading drivers. Just as Ringmaster Ned is essential for introducing new circus acts, the `DriverManager` is instrumental in creating new database connections.

You the programmer won't usually see the `DriverManager`'s work, though. This class mostly works behind the scenes to ensure that everything is cool for your connections.

The `DriverManager` maintains a `Vector` that holds information about all the drivers that it knows about. The elements in the `Vector` contain information about the driver such as the class name of the `Driver` object, a copy of the actual `Driver` object, and the `Driver` security context.

Note
The security context is an implementation-dependent object that defines the environment on which the application is running. This information is usually enough for Java to perform security checks. When the `DriverManager` opens or queries a `Driver`, the security context of the `Driver` will be checked against the security context of your application. If they don't jive, the `Driver` will not be loaded.

The `DriverManager`, while not a `static` class, maintains all `static` instance variables with `static` access methods for registering and unregistering drivers. This allows the `DriverManager` never to need instantiation. Its data always exists as part of the Java runtime.

The drivers managed by the `DriverManager` class are represented by the `Driver` class.

The `Driver` Class

If the cornerstone of JDBC is the `DriverManager`, then the `Driver` class is most certainly the bricks that build the JDBC. The `Driver` is the software wedge that communicates with the platform-dependent database, either directly or using another piece of software. How it communicates really depends on the database, the platform, and the implementation. Figure 10.2 illustrates the software layers in a Java JDBC application connection to a database.

Figure 10.2 : *The Java-JDBC database.*

You must create each `Driver` in your program or you can have them pre-loaded for you by specifying the class names of the driver you wish to have pre-loaded. To do this, you need to specify a value for the system property called `jdbc.drivers`. This property needs to contain the fully qualified class name of each `Driver` separated by a colon (:), which allows the `DriverManager` to instantiate the class because it knows how to find classes in your `CLASSPATH`.

As an example, a sample property setting follows:

```
jdbc.drivers=weblogic.jdbc.oci.Driver:jdbc.odbc.JdbcOdbcDriver
```

This example will load WebLogic's Oracle driver and the JDBC-ODBC bridge drivers. If you were using Sun's HotJava browser, this line would go in your `.hotjava/properties` file. You can also start your programs with the following code as a command-line argument:

```
java -Djdbc.drivers=weblogic.jdbc.oci.Driver:jdbc.odbc.JdbcOdbcDriver
    MyApplication
```

For your purposes, you will place them in your application configuration files and let your `ConfigProperties` class move it for us.

Note
The `ConfigProperties` class checks each property string that it contains for the `jdbc.drivers` and copies it to the system properties list for you.

After a `Driver` is constructed, you really never need to worry about it again. In fact, if you use the `jdbc.drivers` property as outlined above, you never even have to create driver objects.

It is the `Driver`'s responsibility to register with the `DriverManager` and connect with the database. Database connections are represented by the `Connection` class.

The `Connection` Class

The `Connection` class encapsulates the actual database connection into an easy-to-use package. Sticking with your foundation building analogy here, the `Connection` class is the mortar that binds the JDBC together. It is created by the `DriverManager` when its `getConnection()` method is called. This method accepts a database connection URL and returns a database `Connection` to the caller.

The Database Connection URL

To connect with a JDBC data source, a uniform resource locator, or URL, is used. The format follows:

`jdbc:<sub-protocol>:<sub-name>`

where

`sub-protocol` is the name of the driver set that defines a particular database connectivity method. This can be represented by several drivers.

`sub-name` is the additional information necessary to complete the URL. This information is different depending on the `sub-protocol`.

You are undoubtedly familiar with HTTP URLs. The JDBC URL simply prepends a `jdbc`.

Let's look at an example JDBC URL. Let's say you want to connect with an mSQL data source on host `hermy.munster.com` on port `4333`. The instance name is simply called data. The connection URL would be

`jdbc:msql://hermy.munster.com:4333/data`

`msql` is the sub-protocol, and `//hermy.munster.com:4333/data` is the sub-name.

To connect with an Oracle data source with the `jdbcKona` drivers from WebLogic, the following URL:

`jdbc:weblogic:oracle`

is sufficient. This is because the `jdbcKona` drivers use Oracle's SQL*Net software, which maintains its own set of network addresses for database instances. All it needs to know is the type of database with which it is connecting.

Remember that `Vector` that holds driver information in the `DriverManager` class? Well, here's where you'll use it.

When you call the `getConnection()` method, the `DriverManager` asks each driver that has registered with it whether the database connection URL is valid. If one driver responds positively, the `DriverManager` assumes a match. If no driver responds positively, an `SQLException` is thrown. The `DriverManager` returns the error "no suitable driver," which means that of all the drivers that the `DriverManager` knows about, not one of them could figure out the URL you passed to it.

Assuming that the URL was good and a `Driver` stepped up to the plate and said, "I can handle this!", then the `DriverManager` will return a `Connection` object to you. What can you do with a `Connection` object? Not much. This class is nothing more than an encapsulation of your database connection. It is a factory and manager object, and is responsible for creating and managing `Statement` objects.

The `Statement` Class

Picture your `Connection` as an open pipeline to your database. Database transactions travel back and forth between your program and the database through this pipeline. The `Statement` class represents these transactions.

The `Statement` class encapsulates SQL queries to your database. Using several methods, these calls return objects that contain the results of your SQL query. When you execute an SQL query, the data that is returned to you is commonly called the result set. You can choose from several result sets, depending on your needs:

- `ResultSet executeQuery( String sqlStatement )`
  This method sends the SQL query contained in `sqlStatement` and returns a single set of results. This method is best used in sending `SELECT` statements. These statements typically return a result set.

- `int executeUpdate( String sqlStatement )`
  This method sends the SQL query contained in `sqlStatement` and returns an integer. This method is useful when you send SQL `INSERT`s, `DELETE`s, and `UPDATE`s. These commands return a count of rows that were affected by your query. This statement should not be used for queries that return result sets.
- `boolean execute( String sqlStatement )`
  This method sends the `sqlStatement` to the database and returns `true` if the statement returns a result set or `false` if the statement returns an integer. This method is best used when multiple result sets can be returned.

Use the following methods to easily navigate the results a query returns:

- `boolean getMoreResults()`
  This moves to the next result set in the `Statement`. This, like the `execute()` method, returns `true` if the next result is a result set or `false` if it is an integer. If you have already retrieved a `ResultSet` from the `Statement`, this method will close it before returning.
- `ResultSet getResultSet()`
  This method returns to you a result set in a `ResultSet` object. This result set is the current result set.
- `int getUpdateCount()`
  This method returns to you the integer result that an `execute()` method returned.

By now you are probably wondering what this `ResultSet` class is all about, and how it can possibly hold all that data. Well, it's quite a class, as you are about to see.

The `ResultSet` Class

As you've probably guessed, the `ResultSet` class encapsulates the results returned from an SQL query. Normally, those results are in the form of rows of data. Each row contains one or more columns. The `ResultSet` class acts as a cursor, pointing to one record at a time, enabling you to pick out the data you need.

Gaining Access to a Column's Data

You can gain access to the data within the `ResultSet` using many different methods. These methods are in the form of `get`*`type`*`()`, where *`type`* is the data type of the column. These functions return a new instance of the type that contains the data from the result set. If the column value is `NULL`, the value returned from these methods is `NULL`.

Note
The `NULL` value in database lingo is not the same as the `NULL` value in programming languages. Programming `NULL`s usually indicate zero or nothing. In database storage, however, the `NULL` value indicates the lack of a value, enabling zeroes to be stored, giving you clear indication that a column has not been set or modified.

You can access the column's data either by column number or name. Sometimes you might know the column number, sometimes you might not. These methods give you the flexibility to access the columns either way.

Tip
Accessing the columns by name does present more overhead than accessing them by column. In performance-critical applications, consider accessing your data by column number.

The following list presents the `get`*`type`*`()` methods provided by the `ResultSet` class and their return types:

- `String getString()`
- `boolean getBoolean()`
- `byte getByte()`

- short getShort()
- int getInt()
- long getLong()
- float getFloat()
- double getDouble()
- Numeric getNumeric()
- byte[] getBytes()
- java.sql.Date getDate()
- java.sql.Time getTime()
- java.sql.Timestamp getTimestamp()
- java.io.InputStream getAsciiStream()
- java.io.InputStream getUnicodeStream()
- java.io.InputStream getBinaryStream()

Getting to the Next Row

When you've retrieved all the data you can from a column, it is time to move on to the next row. Moving to the next row is done with the `next()` method. This method returns a boolean value indicating the status of the row. Internally, it moves the `ResultSet`'s cursor to the next row, thereby giving access to the data stored in the columns of that row.

When a `ResultSet` object is created, its position is always before the first row of data contained within. This makes it necessary to call the `next()` method before you can access any column data. The first call to `next()` makes the first row available to your program. Subsequent calls to `next()` make the next rows available. If no more rows are available, `next()` returns `false`.

A JDBC Sample Program

Programming concepts aren't always clear when you read about them, therefore a small example of JDBC program is definitely in order. The following program in Listing 10.1 is a sample program that illustrates the concepts presented so far. These concepts follow:

- Connecting to a JDBC data source
- Executing a query
- Parsing the results of the query

Listing 10.1. A sample JDBC program.

```
//*************************************************************************
//* Imports                                              &nb
sp;                        *
//*************************************************************************

import                               java.sql.*;

//*************************************************************************
//* JDBCExample
;                        *
//*************************************************************************

public class
JDBCExample
```

```
    {

    //************************************************************************
    //* main
                                *
    //************************************************************************

        public static void
        main( String args[] )
        throws Exception
        {
            //    Find the class...
            Class.forName( "weblogic.jdbc.oci.Driver" );

            //    Open a connection...
            Connection myConnection = DriverManager.getConnection(
                "jdbc:weblogic:oracle:tcp-loopback.world",
                "scott",
                "tiger" );

            //    Create a statement...
            Statement myStatement = myConnection.createStatement();

            //    Execute a query...
            try
            {
                //    Execute the query...
                myStatement.execute( "select * from emp" );

                //    Get the result set...
                ResultSet mySet = myStatement.getResultSet();

                //    Advance to the next row...
                while ( mySet.next() )
                {
                    //    Get the data...
                    int empno = mySet.getInt( "empno" );
                    String ename = mySet.getString( "ename" );
                    long salary = mySet.getLong( "sal" );

                    //    Print it all out...
                    System.out.println( Integer.toString( empno ) + " - " +
                        ename + " - " + Integer.toString( empno ) );
                }
            }
            catch ( SQLException e )
            {
                System.out.println( "SQL Error: " + e.toString() );
            }

            //    Close everything up...
            myStatement.close();
            myConnection.close();
```

```
    }

  }
```

This program uses `jdbcKona` drivers to communicate with a Personal Oracle7 database on a local machine. When Personal Oracle7 is installed, it creates a database alias called `tcp-loopback.world`. You will connect with this alias in this example. Also included with Personal Oracle7 is a set of sample tables with data. This program will retrieve and print three of the columns from the `employee` table that comes with Personal Oracle7.

What is Personal Oracle7?

Personal Oracle7 is a single-user Oracle product. It runs in either Windows 95 or Windows NT, and provides all of the functionality of the full-blown Oracle server product at a fraction of the cost. It is an excellent tool with which database developers can tune their craft; it is also good for application development.
However, Personal Oracle7 is not like many desktop databases on the market today. There is really no comparison between Personal Oracle7 and Microsoft Access. Access provides you with a complete development environment rife with tools. Personal Oracle7 is simply a database and provides no user-friendly development tools such as wizards. Personal Oracle7 does come with a very nice interface program that lets you edit many database objects with dialog boxes instead of with SQL statements.
Personal Oracle7 can be downloaded for a free trial from the Oracle Web site at
`http://www.oracle.com`
Information about Microsoft Access can be found at
`http://www.microsoft.com`

Making JDBC Easy to Use

To make JDBC even easier to use in your intranet applications, you're going to create a set of interfaces and classes. These objects will greatly simplify your use of JBDC and database programming in general.

As you have been doing, you'll look at the interfaces first then move on to the classes. Two interfaces in particular will be useful to us: `Connector` and `SQLFactory`. They both provide very different functions, so let's cover them individually.

The `Connector` Interface

The `Connector` interface defines the pattern or template that a class must follow to comply with your database connection standard, JDBC. The standard is defined by this interface, however, for the most part. By extending this interface, you are extending the standard. The `Connector` is the place to implement the functionality your applications required.

The `Connector` interface defines four standard methods, which follow:

- `connect`: Connects with a data source
- `disconnect`: Disconnects from a data source
- `connected`: Indicates the connection status
- `getConnectionURL`: Returns an URL used for connecting to a JDBC data source

Let's take a look at each of these interface methods.

The `connect` Method

The `connect` method is defined as follows:

```
    //************************************************************************
    //* connect                                                  &nb
    sp;                         *
    //************************************************************************

        public boolean
        connect( String user, String password, String server );
```

The user's name and password and the server where the data exists are accepted as input arguments. The third parameter, the server where the data exists, might be optional with some `Connector` implementations, but I have included it here for completeness.

The `getConnectionURL` method used this information to connect with a JDBC data source. The implementor must connect to the database in this method.

The `disconnect` Method

The `disconnect` method is defined as follows:

```
    //************************************************************************
    //* disconnect
                           *
    //************************************************************************

        public boolean
        disconnect();
```

Nothing spectacular here. The implementor must disconnect from the database in this method.

The `connected` Method

The `connected` method simply returns `true` or `false` if the object is currently connected to a database. The method is defined as follows:

```
    //************************************************************************
    //* connected                                              &
    nbsp;                       *
    //************************************************************************

        public boolean
        connected();
```

The implementor must keep track of the connection status and report on it through this method.

The `getConnectionURL` Method

The final method in the `Connector` interface is the `getConnectionURL` method. It is defined as follows:

```
    //************************************************************************
    //* constructConnectionURL
                *
    //************************************************************************
```

```
public String
getConnectionURL()
```

The connection URL is used to connect with a JDBC data source. It is up to the implementor of this interface to return the correct connection URL for the database with which it is working.

The SQLFactory Interface

The second interface you need is one with which you can create classes that generate SQL statements for the objects that they contain. The implementor of this interface decides the method of generation.

Why would you want to generate SQL on-the-fly? Because with SQL, you can build smart objects that will know and keep track of whether they have been modified. If you then place these smart objects into a smart container, the container can produce an SQL statement to update the database. But there needs to be a template or pattern for these smart objects to follow, and that's where the SQLFactory class comes in.

The SQLFactory interface exists to provide a common set of methods for objects to use to generate SQL. The SQL generated is used to send information back to the database when values change. Chapter 11, "User Interface Classes," contains some classes that implement this interface.

Note
Chapter 6 discussed the INSERT and UPDATE SQL commands. These two commands are used to ins ert and update records, or rows, in a database and are the subject of these interface methods.

The SQLFactory interface defines several methods. Two methods are very important: generateUpdateSQL and generateInsertSQL.

The generateUpdateSQL Method

This method is defined as follows:

```
//************************************************************************
//* generateUpdateSQl
;                 *
//************************************************************************

    public String
    generateUpdateSQL( boolean addSet );
```

The generateUpdateSQL method takes a single argument: addSet. In an SQL UPDATE statement, you must place the keyword SET in front of the very first column. By setting this argument to true, a SET is placed in front of the UPDATE SQL returned. Sample return values follow:

```
emp_salary = 75000 if addSet is set to false
```

or

```
SET emp_salary = 750000 if addSet is set to true
```

These statements can be concatenated and used in an SQL UPDATE statement.

The generateInsertSQL Method

This method is defined as follows:

```
//************************************************************************
//* generateInsertSQL
```

```
;                *
//*************************************************************************

    public String
    generateInsertSQL( boolean addParen );
```

The `generateInsertSQL` method takes a single argument: `addParen`. In an SQL `INSERT` statement, you must place an opening parenthesis (`(`) in front of the very first column. By setting this argument to `true`, the parenthesis is placed in front of the `INSERT` SQL returned. Sample return values follow:

```
emp_salary = 75000 if addParen is set to false
```

or

```
( emp_salary = 750000 if addParen is set to true
```

These statements can be concatenated and used in an SQL `INSERT` statement.

The Classes

Let's create some classes to access these JDBC data sources. The first class you'll create should encapsulate the connection to a particular database. This class also should implement the `Connector` interface you've just defined. You might want to build several of these specialized connection classes however, so you'll place all of your base functionality into an abstract class, then create subclasses of it that implement the database specifics.

The first class, `DBConnector`, is the abstract base class. You will use it as a basis for specific database connection objects. Several classes will be derived from this base class:

- `MSQLConnector`: Connects to mSQL data sources
- `MSSQLServerConnector`: Connects to Microsoft SQL Server data sources
- `ODBcconnector`: Connects to ODBC data sources
- `OracleConnector`: Connects to Oracle data sources
- `SybaseConnector`: Connects to Sybase data sources

These classes really do nothing more than provide the base class with a database URL with which to connect. The `DriverManager` does the real work.

The last two classes that you've built for your intranet are the `OracleSequence` and the `SequenceGenerator` classes.

`OracleSequence` uses Oracle's sequence object to create the unique number. This object is Oracle's internal counter. When you query it for its next value, it increments its counter and stores the result. It also returns the result to you. Because not everyone will have access to Oracle sequences, however, the `SequenceGenerator` class is provided.

The `SequenceGenerator` class provides the exact same functionality as the `OracleSequence` class but in a generic manner that can be used with any database.

Let's look at some classes in depth.

The `DBConnector` Class

The `DBConnector` class encapsulates a lot of the repetitive tasks that are needed to connect with a JDBC data source. It is an abstract class; therefore, it is incomplete. You cannot instantiate the `DBConnector` class directly; you can only instantiate its derivatives.

The declaration of the `DBConnector` class follows:

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

```
//* DBConnector
;                     *
//*********************************************************************

public abstract class
DBConnector
implements Connector
```

The `DBConnector` class implements the `Connector` interface but not the entire interface. Extensions of this class must implement one method, `getConnectionURL`; therefore it is declared abstract.

Instance Variables

The `DBConnector` class has several important instance variables:

- `protected Connection myConnection`: Holds a JDBC `Connection` object for talking to the database
- `protected Statement myStatement`: Holds a JDBC `Statement` object for executing SQL queries
- `protected boolean isConnected`: Holds an indicator of the connection status
- `protected String lastError`: Holds the last error that occurred, if any

All of these instance variables are protected. Only `myStatement` is available outside of the `DBConnector` hierarchy through the `getStatement()` method. This class is mainly used to execute database queries.

`OracleSequence`

A sequence is a database object that provides unique numbers for storing in the database. For example, you could set up a sequence to provide new employee identification numbers for your employee table. This sequence would then generate the numbers for you.

This class encapsulates using an Oracle sequence. The sequence must exist for this class to work. If you don't have any sequences, you can create one with the following command:

```
CREATE SEQUEnCE sequence name InCREMENT BY increment START
    WITH starting value
```

where

`sequence name` is the name you wish to call the sequence.

`increment` is the amount to increment each time.

`starting value` is the value at which to start the sequence.

Note
The preceding command line is only enough to get by. Many more command options are available. Consult your Oracle manual for more information.

This class provided two methods: `getCurrentValue()` and `getNextValue()`. Each returns `int`s and provides the current and next values of the Oracle sequence. Listing 10.2 shows the source code for the `OracleSequence` class.

Listing 10.2. The `OracleSequence` class.

```
//*********************************************************************
//* OracleSequence                                          &n
bsp;                  *
//*********************************************************************

public class
```

```
OracleSequence
{
//*************************************************************************
//* Members                                              &nb
sp;                        *
//*************************************************************************

    /**
    * The name of the database sequence
    */
    String                          mySequenceName;

    /**
    * The name of the database sequence
    */
    OracleConnector             myConnection;
//*************************************************************************
//* Constructors                                            &nbs
p;                      *
//*************************************************************************

    public
    OracleSequence( OracleConnector connector, String sequenceName )
    {
        myConnection = connector;
        mySequenceName = sequenceName;
    }
//*************************************************************************
//* getNextValue                                            &nbs
p;                      *
//*************************************************************************

    public int
    getNextValue()
    {
        return( getDBSequenceValue( false ) );
    }

//*************************************************************************
//* getCurrentValue                                           &
nbsp;                   *
//*************************************************************************

    public int
    getCurrentValue()
    {
        return( getDBSequenceValue( true ) );
    }

//*************************************************************************
//* getDBSequenceValue                                          &nbs
p;                *
//*************************************************************************
```

```
    protected int
    getDBSequenceValue( boolean currentVal )
    {
        //    Placeholder for the value...
        int                currentValue = 0;

        //    This statement retrieves the current
value from the database
        String sql;

        if ( currentVal )
            sql = "select " + mySequenceName + ".currval from dual";
        else
            sql = "select " + mySequenceName + ".nextval from dual";

        try
        {
            if ( myConnection.getStatement().execute( sql ) )
            {
                //    The results are received here...
                ResultSet rs =
myConnection.getStatement().getResultSet();
                rs.next();
                currentValue = rs.getInt( 1 );
            }
        }
        catch ( SQLException e )
        {
            //    No records? Set to 1...
            currentValue = 1;
        }

        return( currentValue );
    }

}
```

The `getDBSequenceValue()` method constructs an SQL statement dynamically with the construction values that were passed. This statement is then sent to the database and the results are returned.

The `SequenceGenerator` Class

The last class of note is the `SequenceGenerator` class. This class generates a unique sequence number that identifies the rows in your database tables. For example, you can use the `SequenceGenerator` class to generate employee identification numbers. You'll see that use illustrated in Chapter 13, "Employee Files."

The `SequenceGenerator` works on a simple principle. Given a table and column, it retrieves the data in that column of the table in reverse sorted order. Therefore, the first row returned is the highest. It takes this highest value and increments it, creating your unique number.

The constructor accepts as input the necessary components to create the sequence retrieving SQL:

```
    public
```

```
    SequenceGenerator( DBConnector connector, String table,
    String column )
```

The constructor records the value of these input parameters and calls the following code to get the current highest value:

```
    //*************************************************************************
    //* getDBSequenceValue                                                &nbs
    p;                *
    //*************************************************************************

        protected void
        getDBSequenceValue()
        {
            //    The first row returned by this SQL
    statement is the largest number...
            String sql = "select " + myColumn + " from " + myTable +
                " order by " + myColumn + " desc";

            try
            {
                if ( myConnection.getStatement().execute( sql ) )
                {
                    //    Check the first row!
                    ResultSet rs =
    myConnection.getStatement().getResultSet();
                    rs.next();
                    currentValue = rs.getInt( 1 );
                }
            }
            catch ( SQLException e )
            {
                //    No records? Set to 1...
                currentValue = 1;
            }
      }
```

Summary

Well, this has been quite a chapter. You learned about many interfaces and classes that will help you construct intranet applications. These components also provide a consistent framework from which to build connections to JDBC data sources. In addition, you saw some examples of how to use these classes in real applications.

In Chapter 11, you will learn a few neat tricks for creating visually stunning user interfaces and you'll implement the `SQLFactory` interface in some of your classes.

Chapter 11

User Interface Classes

CONTENTS

- Introduction
- 3-D Effects
  - The Effects Interface
- The JifPanel Class
  - JifPanel Design
  - Constructing a JifPanel
  - Smoke and Mirrors
  - Drawing 3-D Borders
  - Tabbing Between Components
- SQL Generation
- The JifPanel Descendants
  - The CalendarPanel Class
  - The ImagePanel Class
  - The JifLabel Class
  - The JifTabPanel Class
  - The StatusBar Class
- The JifDialog Class
  - The MessageBox Class
  - The PickList Class
- Java TextComponent Extensions
  - Change Detection
- Summary

---

tool \'tül \ n: *an instrument or apparatus necessary in the practice of a vocation or profession*

Introduction

Java's Abstract Windowing Toolkit (awt) package contains classes with which you can interact with the user of your programs. Though the awt classes are nice, there are no bells and whistles. This chapter is dedicated to providing you with those bells and whistles to extend the visual beauty of your intranet applications.

The main thrust of this chapter is to introduce you to the interfaces and classes with which you can enhance your user

interface. You will use these classes in your programs mainly to construct visual elements to enhance the look and feel of your Java programs.

Before you dig into the classes, however, let's look at a little 3-D theory.

3-D Effects

Two features that I like to use in my GUI applications are 3-D panels and group boxes. They give a nice touch to any input screen and can transform a boring screen into one bubbling with personality.

Core Java cannot create these hip containers, however. Enter the `JifPanel` class. With this class, you can create an array of interesting effects with the greatest of ease. Figures 11.1 and 11.2 show the different things a `JifPanel` can do.

Figure 11.1 : *The versatile JifPanel.*

Figure 11.2 : *The JifPanel behaving like a group box.*

Note
JIF stands for Java Intranet Framework. As you'll see in Chapter 12, "Putting Them All Together," all of the classes you've talked about, including those in this chapter, are packaged together into the Java Intranet Framework, or JIF. You will see other objects that start with JIF.

The `JifPanel` class is covered in greater detail later in this chapter. For now, let's discuss the interface behind the `JifPanel` class: `Effects`.

Note
Figure 11.1 represents the output of one example program on the CD-ROM. The program is called `PanelTester`.

Note
Figure 11.2 represents the output of one example program on the CD-ROM. The program is called `GroupBoxTester`.

The `Effects` Interface

The `Effects` interface extends the components' look and feel. The `Effects` interface only consists of constants defining the styles available. Only the `JifPanel` class implements this interface.

The `Effects` styles are divided into three areas:

- Border Styles: Govern the look of the border surrounding the component
- Text Styles: Govern the look of any text that is contained within the component
- Text Placement: Governs the placement of any text that is contained within the component

Let's examine each set of styles individually.

Border Styles

The first set of styles is the border styles. These dictate how a component will render its border. The available styles are

- `Effects.NONE`: No cool 3-D look at all.
- `Effects.FLAT`: Provides a 2-D border around the component. Very boring.
- `Effects.GROOVED`: Provides a 3-D border that looks like a groove. The opposite of `Effects.RIDGED`.

- `Effects.LOWERED`: Provides a 3-D border that looks sunken into the component. The opposite of `Effects.RAISED`.
- `Effects.ROUNDED`: Rounds the corners of the border.
- `Effects.RAISED`: Provides a 3-D border that looks popped up. The opposite of `Effects.LOWERED`.
- `Effects.RIDGED`: This 3-D effect looks like a little wall around your component. The opposite of `Effects.GROOVED`.
- `Effects.CAPTION`: This style makes any text draw at the upper left-hand corner of the component.

Each of these styles is represented in Figure 11.1. Listing 11.1 shows the actual definitions of these styles.

Listing 11.1. The border styles available in the `Effects` interface.

```
public final static int     NONE = 0;
public final static int     FLAT = 1;
public final static int     GROOVED = 2;
public final static int     LOWERED = 4;
public final static int     ROUNDED = 8;
public final static int     RAISED = 16;
public final static int     RIDGED = 32;
public final static int     CAPTION = 64;
```

Notice that the styles are offsets in a bit-mapped field. That means that they can be added together to combine the effects. Some effects do not warrant combining and are pointless. For instance, you won't want to combine `GROOVED` with `RIDGED`, or `LOWERED` with `RAISED`. Sometimes you might want to combine `RAISED` and `ROUNDED`, however.

Text Styles

Text effects, the second set of styles defined by the `Effects` interface, dictate the way a component will draw the text it contains. Four styles are available:

- `Effects.TEXT_NORMAL`: Normal, boring text.
- `Effects.TEXT_LOWERED`: Text appears to be sunken into the component.
- `Effects.TEXT_RAISED`: Text appears to jump right off the screen.
- `Effects.TEXT_SHADOWED`: Text appears to have a drop shadow.

Each of these styles is represented in Figure 11.1. Listing 11.2 shows the actual definitions of these styles.

Listing 11.2. The text styles available in the `Effects` interface.

```
public final static int    TEXT_NORMAL = 0;
public final static int    TEXT_LOWERED = 1;
public final static int    TEXT_RAISED = 2;
public final static int    TEXT_SHADOW = 3;
```

Unlike the border styles, these styles increment sequentially. They cannot be combined.

Text Placement

The third and final style set is the text placement styles. While not styles so much as locations, they indicate one aspect of the display of the text. Three styles are available:

- `Effects.CENTER`: Text is centered within a component.
- `Effects.LEFT`: Text is left justified.
- `Effects.RIGHT`: Text is right justified.

Again, please refer to Figure 11.1. Each of the preceding styles is demonstrated in the example. Listing 11.3 shows the actual definitions of these styles.

Listing 11.3. The text placements available in the `Effects` interface.

```
public final static int        CENTER = 0;
public final static int        LEFT = 1;
public final static int        RIGHT = 2;
```

Like the text styles, these placement styles increment sequentially. They also cannot be combined.

The `JifPanel` Class

The most important user interface class you'll develop is the `JifPanel` class, which implements the `Effects` interface and provides many 3-D effects along with some special features. To fully explain the `JifPanel`, this section discusses its design and then covers some of the constructors and the source code.

`JifPanel` Design

The `JifPanel` extends Java's `Panel` class. As you know, the `Panel` class extends the `Container` class, which allows the `Panel` to hold or contain other components. You can assemble this panel of components, along with other panels of components, to construct a complete user interface for an application.

The following is the declaration of the `JifPanel` class:

```
//************************************************************************
//* JifPanel                                              &n
bsp;                        *
//************************************************************************

public class JifPanel
extends Panel
implements Effects
```

The `JifPanel` implements the `Effects` interface. As the `Effects` interface only contains constants, it imposes no structure on the `JifPanel` class. All of the constants available in `Effects` are available to the `JifPanel`, however.

Constructing a `JifPanel`

You can construct a `JifPanel` in several ways. You can use it as a regular Java `Panel` by creating it with no arguments:

```
JifPanel myPanel = new JifPanel();
```

You can create one that has a specific type of border:

```
JifPanel myPanel = new JifPanel( Effects.RAISED );
```

You can specify a fixed size for your panel:

```
JifPanel myPanel = new JifPanel( Effects.RAISED, 100, 100 );
```

You can create one that contains a text string:

```
JifPanel myPanel = new JifPanel( "JIF Rules!!" );
```

Finally, you can create one with a cool border that is of a fixed size with text inside:

```
JifPanel myPanel = new JifPanel( Effects.RAISED, 100, 100, "JIF Rules!!"
);
```

Hopefully there are enough options from which you can choose. If there aren't, you can always subclass the `JifPanel` and create your own extensions.

Once again, refer to Figure 11.1 for a gander at the different border, text, and text placement styles available for a `JifPanel`.

Smoke and Mirrors

If you're at all like me, you are probably ready to look at some source code. I'm sure you're wondering how to create those cool 3-D border and text effects. It doesn't take smoke and mirrors or a degree from a clown college. It's quite simple, and I'll show you how.

Drawing 3-D Borders

A 3-D border around a component is one of the easiest and most professional looking effects to program. Drawing a 3-D border around any component requires two details: border style and border thickness.

The first step is to decide on the style of the 3-D border you want to draw. The two basic styles are lowered and raised.

Lowered Borders

A lowered border is a border that looks sunken or lower than the area that contains it. Figure 11.3 illustrates the concept of a lowered border.

Figure 11.3 : *A lowered border.*

To make an object appear as if it is sunken, you need to create what appears to be a shadow. To do this, the upper and left side of your component should be darker than the lower and right side. In Figure 11.3, a 10∞10 grid represents a 10∞10 grid of pixels. The dark pixels represent the area in the shadow. The clear, or white, pixels represent the area that is lit up. If you squint your eyes and look at the figure, it will look lowered into the book.

Raised Borders

A raised border is a border that looks as if it is coming off of the screen. This is a common look for buttons and other clickable objects. Figure 11.4 illustrates the concept of a raised border.

Figure 11.4 : *A raised border.*

To make the object appear as if it is raised, you need to create a shadow at the bottom of the object. To do this, the upper and left side of your component should be lighter than the lower and right side. In Figure 11.4, a 10∞10 grid represents a 10∞10 grid of pixels. The dark pixels represent the area in the shadow. The clear, or white, pixels represent the area that is lit up. Again, if you squint your eyes and look at the figure, it will look raised from the book.

Grooved and Lowered Borders

The grooved and lowered border types are worth mentioning. You can create group boxes with these two borders. Figure 11.5 illustrates the method for creating a grooved and lowered border.

Figure 11.5 : *A grooved border*.

Figure 11.6 illustrates the method for creating a lowered border.

Figure 11.6 : *A lowered border*.

Border Thickness

The last detail you need before you dig into the code is the thickness of the border. The thickness of the border is the width of the border in pixels. The default thickness for a `JifPanel` is two pixels.

Thicker borders look odd, and borders thinner than two pixels almost defeat the purpose of the 3-D effect.

Drawing the Border

The `drawFrame()` method in the `JifPanel` class is responsible for drawing the 3-D border. The source code for that method follows:

```
//************************************************************************
//* drawFrame                                                    &
nbsp;                       *
//************************************************************************

    public void
    drawFrame( Graphics g )
    {
        int         i, j, offset = 0, rounded = 0;
        Dimension    bounds = size();

        if ( isStyleSet( CAPTION ) )
        {
            //    Fix up for font...
            FontMetrics fm = g.getFontMetrics();
            offset = fm.getHeight() / 2;
        }

        for ( i = 0, j = 1; i < thickness; i++, j += 2 )
        {
            if ( i != 0 )
                rounded = 0;
            else
                if ( isStyleSet( ROUNDED ) )
                    rounded = 1;

            if ( isStyleSet( RAISED ) ||
                ( isStyleSet( RIDGED ) && ( j <
thickness ) ) ||
                ( isStyleSet( GROOVED ) && ( j >
thickness ) ) )
                g.setColor( Color.white );
            else
                g.setColor( Color.gray );
            //    Draw top line...
            g.fillRect( i + rounded,
                i + offset,
                bounds.width - ( 2 * ( i + rounded ) ),
                1 );

            //    Draw the left side...
            g.fillRect( i,
                i + rounded + offset,
                1,
                bounds.height - ( 2 * i + 1 + rounded ) - offset );
```

```
            if ( isStyleSet( RAISED ) || isStyleSet(
FLAT ) ||
                ( isStyleSet( RIDGED ) && ( j < thickness ) ) ||
                ( isStyleSet( GROOVED ) && ( j >
thickness ) ) )
                g.setColor( Color.gray );
            else
                g.setColor( Color.white );

            //    Draw the bottom line...
            g.fillRect( i + rounded,
                bounds.height - ( i + 1 ),
                bounds.width - ( 2 * ( i + rounded ) ),
                1 );

            //    Draw the right side...
            g.fillRect( bounds.width - ( i + 1 ),
                i + rounded + offset,
                1,
                bounds.height - ( 2 * i + 1 + rounded )
- offset );
        }
    }
```

The `drawFrame()` method draws the border in a loop fashion. Each run through represents one pixel from the thickness of the border. If the border thickness was set to two, the default, this loop would be performed twice.

This method first draws the upper and left lines of the border. It then swaps colors and draws the lower and right lines of the border. Each time through the loop, it thickens the border by one pixel.

Ridged and grooved borders are a little bit more complex. They swap colors halfway through the loop, giving them a walled or grooved look.

Drawing 3-D Text

Drawing 3-D text isn't nearly as complicated as drawing the 3-D borders. Making the text look lowered or raised is very simple. The following source code is for the `JifPanel` method `drawtext()`:

```
//************************************************************************
//* drawtext                                              &n
bsp;                    *
//************************************************************************

    public void
    drawtext( Graphics g )
    {
        Color               oldColor = g.getColor();
        int                   xx = 0, yy = 0;
        FontMetrics           fm = g.getFontMetrics();
        Dimension           bounds = size();

        //    Decide where to place the text...
        if ( !isStyleSet( CAPTION ) )
        {
```

```
                switch ( textPlacement )
                {
                    case CENTER:
                        xx = ( bounds.width / 2 ) - ( fm.stringWidth( text )
  / 2 );
                        yy = ( bounds.height / 2 ) - ( fm.getHeight() / 2 );
                        break;

                    case LEFT:
                        xx = thickness + TEXT_OFFSET;
                        yy = ( bounds.height / 2 ) - ( fm.getHeight() / 2 );
                        break;

                    case RIGHT:
                        xx = bounds.width - thickness - TEXT_OFFSET -
                            fm.stringWidth( text );

                        yy = ( bounds.height / 2 ) - ( fm.getHeight() / 2 );
                        break;
                }
            }
            else
            {
                int spacer = fm.charWidth( 'i' );

                xx = thickness + TEXT_OFFSET + spacer;
                yy = 0;

                //    Fill a rectangle in the bounding space of the string...
                g.setColor( getBackground() );
                g.fillRect( xx, yy,
                    fm.stringWidth( text ) + ( spacer * 2 ),
                    fm.getHeight() );

                xx += spacer;
            }

            //    Adjust for drawString weirdness...
            yy += fm.getHeight() - fm.getDescent() - 1;

            if ( textStyle == TEXT_LOWERED )
            {
                //    Draw highlight to right and below text
                g.setColor( Color.white );
                g.drawString( text, xx + 1, yy + 1 );
            }

            if ( textStyle == TEXT_RAISED )
            {
                //    Draw highlight to left and above text
                g.setColor( Color.white );
                g.drawString( text, xx - 1, yy - 1 );
            }
```

```
        if ( textStyle == TEXT_SHADOW )
        {
            //    Draw shadow to right and below text
            g.setColor( Color.gray );
            g.drawString( text, xx + 1, yy + 1 );
        }

        g.setColor( Color.black );
        g.drawString( text, xx, yy );

        //    Restore the old color...
        g.setColor( oldColor );
    }
```

The first half of this method sets the starting point for the text to be drawn. Based on the text placement, the x and y coordinates for drawing the text are calculated. For `CENTER`, the text is centered in the width of the component. For `LEFT`, the text is offset by five pixels. For `RIGHT`, the text is offset by five pixels from the right side of the component.

Now, if the `CAPTION` style is set, you need special processing because the `CAPTION` style requires a background to be drawn before the text is drawn. The method calculates the size of the rectangle that will be drawn. This rectangle will be a little wider than the text itself but exactly the same height.

The second half of this method draws the specialty text, which is the lowered, raised, and shadowed text styles. These effects are achieved simply by drawing a copy of the text in white and then drawing the text again in black, offset by one pixel. This offset is important. If you offset up and over one, you get that lowered look. If you offset down and to the right, you get a raised look. The shadowed text is just like the lowered text, but the background color is gray instead of white.

Finally, you restore the old color to the `Graphics` canvas. Others may depend on this color being there. If you don't do this, subsequent drawing of other components may be colored incorrectly.

Tabbing Between Components

Another feature of the `JifPanel` class is that you can tab between the components contained in the panel. This behavior of tabbing from one component to another is a GUI standard that many have come to feel comfortable with. But creating the effect is quite simple as you'll see.

Note
As of the JDK v1.0.2 for Windows NT and Windows 95, this tabbing feature was not available and has been reported as a bug. I have added it here for this reason. If later versions of the JDK support tabbing, this feature of the `JifPanel` will no longer be necessary.

You took over the event handler in our `JifPanel` class and looked for the pressed Tab key. The Tab key's value is 9, and you created a constant called `TAB_KEY` to hold that value:

```
    final static public int        TAB_KEY = 9;
```

The instance variable called `currentPos` keeps track of the component that currently has the focus.

When you press a key on the keyboard, a `KEY_PRESS` event is generated. Capturing keystrokes is as easy as overriding the `handleEvent()` method.

When you press the Tab key, you check to see if the target of the key press is a component. If it is, you jump the focus to the component directly following the target in the creation order. If you hold down the Shift key when you press the Tab key, you jump backward one component. This handles the jumping from the Tab key.

One last thing you need to do is keep track of mouse clicks. When the user clicks the mouse on a component, that component receives the focus and then generates a `GOT_FOCUS` event. You capture this event as well and set your internal pointer, `currentPos`, to point at this component.

Caution
As of the JDK v1.0.2 specifically on the Windows NT and Windows 95 platform, components did not properly generate the `GOT_FOCUS` and `LOST_FOCUS` events when they received and lost the focus. This appears to be a bug in the implementation, but it might turn out that some components never will generate these events. In any case, the behavior of tabbing can be affected by the user clicking on a component out of the tab order because of this bug. You might see the Tab key cause the cursor to jump around.

The following is the source code for the `handleEvent()` method of `JifPanel`:

```
    //************************************************************************
    //* handleEvent
    ;                       *
    //************************************************************************

        public boolean
        handleEvent( Event event )
        {
            //    Look for certain events to move the focus...
            if ( ( event.id == Event.GOT_FOCUS || event.id ==
    Event.ACTION_EVENT )
                && event.target instanceof Component )
            {
                //    Set the focus to this comoponent...
                setFocus( ( Component )event.target );
            }

            //    Handle tabs nicely...
            if ( event.id == Event.KEY_PRESS && event.key == TAB_KEY )
            {
                if ( event.target instanceof Component )
                {
                    if ( !event.shiftDown() )
                        focusForward( ( Component )event.target );
                    else
                        focusBackward( ( Component )event.target );
                }

                //    I handled it...
                return( true );
            }

            //    I don't want this...
            return( super.handleEvent( event ) );
        }
```

The two methods, `focusForward()` and `focusBackward()`, move the focus to the next or previous component, respectively. These two methods skip `Label`s and disabled components. You don't want an inert object to get the focus;

you'll never get the focus back.

SQL Generation

The last feature of the wonderful `JifPanel` is that it can generate a SQL statement suitable for sending to a database. This feature only works when the panel itself contains other components that implement the `SQLFactory` interface, however. (See Chapter 10, "Database Classes," for more information on `SQLFactory`.)

When you request a `JifPanel` to generate a SQL statement for the components contained within, the `JifPanel` queries each component for some information. First and foremost, the panel needs to know if the component implements the `SQLFactory` interface. If it does not, there is really no way to generate SQL for that component. To see whether a class implements an interface, the operator `instanceof` returns `true`. The following code shows this in action:

```
//    Is this a non-mundane component?
    if ( cList[ i ] instanceof SQLFactory )
    {
        //    Get the SQL statement...
        String sql = ( ( SQLFactory )cList[ i ] ).generateUpdateSQL(
false );
    }
```

The `cList` is an array of the components contained within the panel. You check each element in the array with the `instanceof` operator. If the component implements the `SQLFactory` interface, you then can call its `generateUpdateSQL()` method by casting the component array variable to that of a `SQLFactory` object.

In any case, the `JifPanel` will return complete `INSERT` or `UPDATE` SQL statements if filled with components that implement the `SQLFactory` interface. These strings are suitable for sending directly to your database.

The `JifPanel` Descendants

Five classes descend from the `JifPanel` class: `CalendarPanel`, `ImagePanel`, `JifLabel`, `JifTabPanel`, and `StatusBar`. Let's look at each one in detail.

The `CalendarPanel` Class

As the name implies, the `CalendarPanel` represents a calendar. (See Figure 11.7.) The calendar starts at a certain month and can be moved forward or backward one month at a time. Each day is implemented as a button that the user can press.

Figure 11.7 : *The* CalendarPanel *as used in the* CalenderTester *program.*

When the user presses one of the days, an `ACTION_EVENT` is sent to the parent of the object. This event contains the date selected in a Java `Date` object as the argument.

Listing 11.4 is from the `CalendarTester` program. It creates and displays a `CalendarPanel` object for you to play with. The complete source code is available on the CD-ROM. The output of the `CalendarTester` program is shown in Figure 11.8.

Figure 11.8 : *The output of the* CalenderTester *program.*

Listing 11.4. A code snippet from the `CalendarTssester` program.

```
//************************************************************************
//* CalendarTester                                              &n
bsp;                  *
//************************************************************************
```

```
    public
    CalendarTester()
    {
        super( "Calendar Tester" );

        //    Let's use a nicer font...
        setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );

        //    Create the calendar...
        myPanel = new CalendarPanel();
        add( "Center", myPanel );

        //    Pack it up!
        pack();
        show();
    }

//************************************************************************
//* action                                              &nbs
p;                          *
//************************************************************************

    public boolean
    action( Event e, Object a )
    {
        //    Was this my boy?
        if ( e.target == myPanel )
        {
            //    Tell the user!
            MessageBox mb = new MessageBox( this, "Hey!",
                    "You picked " + ( ( Date )a ).toString() );

            mb.show();
            return( true );
        }

        return( false );
    }

}
```

This program simply creates a `CalendarPanel` object and adds it to the center of its `BorderLayout`. The `ACTION_EVENT` event tells you that someone clicked a button. So, you've added an `action()` method to receive these `ACTION_EVENT` events. When the events are received, you verify that they were generated from your `CalendarPanel`. If all checks out, you notify the user that a day has been picked with another cool class-the `MessageBox`. This class is discussed later in this chapter.

The `ImagePanel` Class

The `ImagePanel` class is an extension of the `JifPanel` class that loads an image and resizes itself to the size of the image. This is useful for many applications, including in the `JifDialog` descendant class `MessageBox`. Later in this chapter you'll see more about the `MessageBox` class.

`ImagePanel`s are constructed in the following manner:

```
ImagePanel ip = new ImagePanel( "c:\\image.gif", 15 );
```

or

```
ImagePanel ip = new ImagePanel( "c:\\image.gif );
```

Each constructor takes as the first argument the name of the graphic file to load. Whatever image types Java supports, the `ImagePanel` also supports. Currently, only GIF and JPEG image formats are supported.

The second, optional, argument allows to you specify a pixel border between the edge of the panel and where the image is drawn. The default is five pixels.

The `JifLabel` Class

One shortcoming of Java's `Label` class is that it can only nicely display a single line of text. If you put a second line in there, separated by a line feed perhaps, you get an ugly display on the screen. The `JifLabel` fills this niche. It creates a multiline label in an easy manner.

Another Java book that includes an excellent multiline label class is on the market, so I tried to do one better. The other multiline label class depends on when you created the components and manages the size of the text. Ugh! While elegant and visually pleasing, it was too much work. I like the KISS principle of software construction: Keep It Simple, Stupid. On that note, I present the entire source code for the `JifLabel`:

```
//*************************************************************************
//* JifLabel                                                  &n
bsp;                        *
//*************************************************************************

public class
JifLabel
extends JifPanel
{

//*************************************************************************
//* Constructors                                                 &nbs
p;                     *
//*************************************************************************

    public
    JifLabel( String s )
    {
        //    Stuff the string in a tokenizer to get at each line...
        StringTokenizer st = new StringTokenizer( s, "\n" );
        int tCount = st.countTokens();

        //    Make a grid...
        setLayout( new GridLayout( tCount + 2, 1, 0, 0 ) );

        //    Leave some space...
        add( new Label( " " ) );

        //    Make each line a new label and add to layout...
        for ( int i = 0; i < tCount; i++ )
            add( new Label( st.nextToken() ) );
```

```
            //    Leave some space...
            add( new Label( " " ) );
        }

    }
```

The `JifLabel` is extremely simple. First, you break the string into bits on the new-line (`\n`) boundaries. Using Java's `StringTokenizer` makes this task very simple.

Second, you store the number of lines of text in the variable `tCount` to create a new layout for the label. You create a grid layout that is `tCount` plus two rows high and one column wide. The extra two rows are spacer rows to make the `JifLabel` look even nicer.

Now you are ready to construct the actual label. Add a spacer label at the top. Next, spin through a loop and take out each line from the tokenizer. Each line you take out is added to the layout as a new `Label`. Lastly, add a new spacer to the bottom.

Pretty simple, eh? I think you'll like the results considering the code that created it. Figure 11.9 shows a sample `JifLabel` in the output of the `LabelTester` program.

Figure 11.9 : *The output of the* LabelTester *program.*

The following code is part of the `LabelTester` program so you can see how easy it is to create `JifLabel`s:

```
    public
        LabelTester()
        {
            super( "Label Tester" );

            String labelString =
                "Hi, this is the versatile JifLabel!\n" +
                "It is a multi-line label that was\n" +
                "very easy to program!\n\n" +
                "What do you think?";

            JifLabel myLabel = new JifLabel( labelString );
            myLabel.setFont( new Font( "Helvetica", Font.BOLD, 14  ) );
            add( "Center", myLabel );

            //    Pack it up!
            pack();
            show();
        }
```

The `JifTabPanel` Class

Another shortcoming of the stock Java widgets is the lack of a tabbed panel, or folder widget. This widget presents information one sheet at a time. This new metaphor is common in Microsoft Windows and is becoming popular on other platforms as well.

Sun Microsystems, Inc. noted that this would catch on, so they provided the `CardLayout` layout manager. This layout presents a stack of components with only one component visible at a time. You must bring one card in the stack to the top to view it.

The `JifTabPanel` class has used this layout. This class provides a simple Microsoft Windows-like tabbed panel for your applications. This simple device uses a `BorderLayout` for its components.

The `JifTabPanel` class consists of a `JifPanel`, a row of tabs from which to select in the North layout, and a `CardLayout` of components in the center layout. The `JifTabPanel` is limited in that it only provides a single layer of tabs. (See Figure 11.10.)

Figure 11.10 : *The output of the* TabTester *program*.

The following code is from the `TabTester` program. You can see how easy it is to use:

```
public
TabTester()
{
    super( "Tab Tester" );

    //    Create a Tab panel...
    JifTabPanel myPanel = new JifTabPanel();

    //    Add some panes to it...
    myPanel.addPane( "Pane 1",
        new JifPanel( JifPanel.RAISED, 275, 375, "Panel 1" ) );
    myPanel.addPane( "Pane 2",
        new JifPanel( JifPanel.RAISED, 275, 375, "Panel 2" ) );
    myPanel.addPane( "Pane 3",
        new JifPanel( JifPanel.RAISED, 275, 375, "Panel 3" ) );
    myPanel.addPane( "Pane 4",
        new JifPanel( JifPanel.RAISED, 275, 375, "Panel 4" ) );
    myPanel.addPane( "Pane 5",
        new JifPanel( JifPanel.RAISED, 275, 375, "Panel 5" ) );

    //    Add to the center...
    add( "Center", myPanel );

    //    Pack it up!
    pack();
    show();
}
```

The `StatusBar` Class

The final `JifPanel` extension is the `StatusBar` class, the area at the bottom of our model intranet application (see Chapter 7, "A Model Intranet Application," for more information). You can use the `StatusBar` class to display messages to the user. If you leave it at its default value, it simply says "Ready." You can set it to anything you want, however.

The `StatusBar` class has only a single method useful to your program: the `clear()` method. This method sets the text to nothing so the status bar shows nothing. It relies on the `setText()` method of the parent class, which is used to change the text displayed in the status bar.

The `JifDialog` Class

The next major user interface class is the `JifDialog` class. This class extends the basic Java `Dialog` class and adds a few cool behaviors:

- The font is defaulted to 12-point Dialog.
- Resizing is turned off.

- The actual window destroys itself when the user closes it.

You would have to implement these three features for each dialog box you create, so why not put them in a centralized base class? That's what the `JifDialog` class is-a centralized location.

A method also included in this class, `center()`, centers the dialog box within either the screen or the window that owns it.

Note
The Windows NT/95 JDK v1.0.2 has a bug that causes all windows to report their `x,y` location as `0,0`. This makes it impossible to center the dialog box within the parent window. Therefore, the parent centering will not work until the bug is fixed or used on another platform.

Three classes descend from `JifDialog`: `MessageBox`, `ResponseDialog`, and `PickList`. All of these classes, except `PickList`, are demonstrated in the `DialogTester` program. The source code for this program is shown in Listing 11.5.

Listing 11.5. The `DialogTester` program.

```
//************************************************************************
//* DialogTester                                                  &nbs
p;                    *
//************************************************************************

public class
DialogTester
extends Frame
{

//************************************************************************
//* Members                                                 &nb
sp;                          *
//************************************************************************

    ResponseDialog                 myDialog;

//************************************************************************
//* main
                             *
//************************************************************************

    public static void
    main( String args[] )
    {
        new DialogTester( args );
    }

//************************************************************************
//* Constructor
;                     *
//************************************************************************

    public
```

```
        DialogTester( String args[] )
        {
            super( "Dialog Tester!" );

            JifPanel p = new JifPanel( Effects.LOWERED );
            p.setLayout( new FlowLayout() );

            p.add( new Button( "Plain MessageBox" ) );
            p.add( new Button( "Info MessageBox" ) );
            p.add( new Button( "Stop MessageBox" ) );
            p.add( new Button( "Exclamation MessageBox" ) );
            p.add( new Button( "Question MessageBox" ) );
            p.add( new Button( "ResponseDialog" ) );

            //    Add the timer panel to the frame...
            add( "Center", p );

            //    Pack the panels...
            pack();
            show();
        }

    //************************************************************************
    //* action                                              &nbs
    p;                          *
    //************************************************************************

        public boolean
        action( Event event, Object arg )
        {
            MessageBox        mb = null;

            if ( arg.equals( "Plain MessageBox" ) )
            {
                mb = new MessageBox( this, "Plain MessageBox",
                    "This is a plain message box" );
            }
            else if ( arg.equals( "Info MessageBox" ) )
            {
                mb = new MessageBox( this, "Info MessageBox",
                    "This is an info message box",
                    MessageBox.INFO );
            }
            else if ( arg.equals( "Stop MessageBox" ) )
            {
                mb = new MessageBox( this, "Stop MessageBox",
                    "This is an stop message box",
                    MessageBox.STOP );
            }
            else if ( arg.equals( "Exclamation MessageBox" ) )
            {
                mb = new MessageBox( this, "Exclamation MessageBox",
                    "This is an exclamation message box",
```

```
                MessageBox.EXCLAMATION );
        }
        else if ( arg.equals( "Question MessageBox" ) )
        {
            mb = new MessageBox( this, "Question MessageBox",
                "This is an question message box",
                MessageBox.QUESTION );
        }
        else if ( arg.equals( "ResponseDialog" ) )
        {
            myDialog = new ResponseDialog( this, "ResponseDialog",
                "This is a response dialog with three buttons",
                "Yes,No,Cancel" );

            myDialog.show();
            return( true );
        }

        if ( mb != null )
        {
            mb.show();
            return( true );
        }

        if ( event.target == myDialog )
        {
            MessageBox mb2 = new MessageBox( this, "Response Received!",
                "A response was received from the response dialog!\n" +
                "You pressed button #" + ( ( Integer )arg ).toString() );

            mb2.show();
            return( true );
        }

        System.out.println( event.toString() );

        return( false );
    }

}
```

The `PickList` dialog, an abstract class, is demonstrated in its own example program.

The `MessageBox` Class

The `MessageBox` class displays a message to the user and has a single button along the bottom of the dialog box. This button is usually an OK button. It allows the user to dismiss the dialog.

As an option, the `MessageBox` can show an image along the left edge of the dialog box to further enhance it. Figure 11.9, the output from `LabelTester`, illustrates a `MessageBox` without an image. Figure 11.11 shows a `MessageBox` with an image.

Figure 11.11 : *A* MessageBox *with the* STOP *image.*

You can display four stock images, or icons, in your message box. Each image is represented by a constant defined in the `MessageBox` class. The stop sign shown in Figure 11.11 is just one of those. The other three are as follows:

- Informational: The letter `i` icon is displayed.
- Exclamation: An exclamation point icon is displayed.
- Question: A question mark icon is displayed.

The constants are defined as follows:

```
public static final String     INFO = "Information.gif";
public static final String     EXCLAMATION = "Exclamation.gif";
public static final String     STOP = "Stop.gif";
public static final String     QUESTION = "Question.gif";
```

Note
The images that come with JIF are from Windows 95. Microsoft Windows has an API function called `MessageBox` that generates similar output. This class closely models the Microsoft Windows message box.

The constructor for the `MessageBox` follows:

```
MessageBox( Frame parent, String title, String message
    [, String imageToUse[, boolean addButtons ] )
```

In the preceding code:

`parent` is the parent frame.

`title` is the title of the message box.

`message` is the message to be displayed to the user.

`imageToUse` is an optional constant indicating which image to display. If not specified, it defaults to `MessageBox.INFO`.

`addButtons` is a boolean with which you can have the message box create itself without an OK button along the bottom. With this feature, you can extend its functionality.

Some examples of the `MessageBox`'s creations follow:

```
MessageBox mb = new MessageBox( this, "Plain MessageBox",
    "This is a plain message box" );

MessageBox mb = new MessageBox( this, "Stop MessageBox",
    "This is an stop message box",
    MessageBox.STOP );

MessageBox mb = new MessageBox( this, "Info MessageBox",
    "This is an info message box",
    MessageBox.INFO );

MessageBox mb = new MessageBox( this, "Exclamation MessageBox",
    "This is an exclamation message box",
    MessageBox.EXCLAMATION );

MessageBox mb = new MessageBox( this, "Question MessageBox",
    "This is an question message box",
    MessageBox.QUESTION );
```

The `MessageBox` is quite a versatile class. As you'll see, it can help out in almost any situation.

Tip
In order for the `MessageBox` to work properly, the image files must reside in the same directory as your program. Otherwise, the image loader will not be able to find them.

The `ResponseDialog` Class

The `ResponseDialog` is the only descendant of the `MessageBox` class. This class extends the base class and adds a user-specified number of buttons. Furthermore, the `ResponseDialog` returns the number of the button that was pressed via an `ACTION_EVENT` event.

The following code is the constructor for the `ResponseDialog`:

```
public
ResponseDialog( Frame parent, String title, String message,
    String buttons )
```

In the preceding code

`parent` is the parent frame.

`title` is the title of the message box.

`message` is the message to be displayed to the user.

`imageToUse` is an optional constant indicating which image to display. If an image is not specified, the constant defaults to `MessageBox.INFO`.

`buttons` is the comma-separated list of button names to create along the bottom of the dialog.

Tip
With this button freedom, you can create all sorts of dialog boxes: Yes/No, Yes/No/Cancel, Maybe, and so on. The choice is yours, and there is no need to create a new class. This class is completely usable as is.

To notify you of which button was pressed, the `ResponseDialog` generates an `ACTION_EVENT` event with an argument of the zero-based index of the button. For example, if you create a dialog with the buttons OK and Cancel and the user presses the OK button, the `ACTION_EVENT` event generated would have an argument of `0`. If the user pressed the Cancel button, the argument would be `1`.

If the user closes the box without pressing a button, the `ResponseDialog` generates an argument of `-1`. This gives you total control to interpret the result.

The `PickList` Class

The last descendant from `JifDialog` is the `PickList` class. This abstract class is a framework for creating pick lists of data. Figure 11.12 shows the pick list from the employee maintenance application.

Figure 11.12 : *The employee pick list.*

To complete the class, thus making it usable, the derived class must supply the `init()` method. This should initialize the `List` object contained within the pick list.

The `init()` source code for the employee pick list is shown in Listing 11.6.

Listing 11.6. The `init()` source code for the employee pick list.

```
//*************************************************************************
//* init
```

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

    public void
    init()
    {
        int rows = retrieveEmployees();
    }

//***************************************************************************
//* retrieveEmployees                                                  &n
bsp;                 *
//***************************************************************************

    int
    retrieveEmployees()
    {
        String        sql;
        boolean        rv = false;
        int            rows = 0;

        sql = "select * from emp_t order by last_name, first_name";

        try
        {
            rv = myConnection.getStatement().execute( sql );
        }
        catch ( SQLException e )
        {
            //    No employees to return...
            return( 0 );
        }

        //    Is this a result set?
        if ( rv )
        {
            try
            {
                ResultSet rs =
myConnection.getStatement().getResultSet();

                //    Spin through the results and add them to the
list...
                while ( rs.next() )
                {
                    EmployeeRecord er = new EmployeeRecord( rs );

                    //    Add to list...
                    if ( er.emp_id != -1 )
                    {
                        myList.addItem( er.nice_name );

                        //    Add to row mapper...
```

```
                        rowMap.insertElementAt( er, rows );

                        //    Increment row counter...
                        rows++;
                    }
                }
            }
            catch ( SQLException e )
            {
                //    Indicate an error!
                rows = -1;
            }
        }

        //    We're done!
        return( rows );
    }
```

As you can see, the `init()` method calls another method called `retrieveEmployees()`. This method retrieves a list of employees from the database and loads them into the list.

The `PickList` generates an `ACTION_EVENT` event to the owner of the pick list when the user clicks on a row or presses a button.

Java `TextComponent` Extensions

The last two user interface classes are `JifTextArea` and `JifTextField`. These classes extend the default Java classes, `TextArea` and `TextField`, by implementing the `SQLFactory` interface from our database classes (see Chapter 10, "Database Classes," for more information). This interface, in conjunction with the `JifPanel` class, generates SQL code from a container of these text objects. The two classes also add change-detection functionality.

Change Detection

To generate correct SQL, the component needs to know whether the data that it contains has changed. If it has not changed, no SQL is needed. If the data has changed, however, a SQL statement should be generated.

To handle this, the `JifTextArea` and `JifTextField` classes contain an instance variable called `dataChange`. This Boolean declaration follows:

```
    protected boolean            dataChange = false;
```

Whenever there is a change to the component's text, this variable is set to `true` and an `ACTION_EVENT` event is sent to the owner of this component. This way, the parent container can act on the notice as well.

The `didDataChange()` method determines whether there has been a change. The source follows:

```
    //*************************************************************************
    //* didDataChange                                                  &nb
    sp;                  *
    //*************************************************************************

        public boolean
        didDataChange()
        {
            return( dataChange );
```

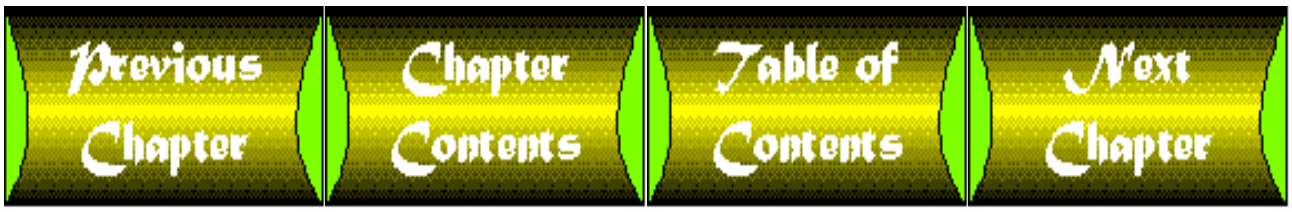
```
    }
```

You can call this method from anywhere.

Summary

This chapter introduced you to many of the user interface classes that you've created. You learned about many user interface classes that extend Java. These classes provide your intranet applications with some really cool widgets for which your users will thank you. With these classes you also can generate cool 3-D effects in addition to awesome `Dialog` extensions that really simplify a lot of coding efforts. All of these classes can be used to extend and enhance your intranet applications.

In the next chapter, "Putting Them All Together," you combine all the classes you've learned about into packages. In addition, you create a new package that contains some classes that uses all of those classes.



Chapter 12

Putting Them All Together

CONTENTS

- Introduction
- Java Compilation Basics
  - Java Source Code Files
  - Have You Got the Package?
  - Making Java Packages
- Introducing the Java Intranet Framework
  - Packaging the JIF Classes
- Extending the Framework
  - Java Applets
  - Making JIF Easy to Use
- The JifApplication Interface
- The Jiflet Class
  - Instance Variables
  - Constructors
  - Methods
  - Wrapping Up Jiflets
- Programming with Jiflets
  - The Smallest Jiflet
  - The HelloWorld Jiflet
- Extending Jiflets for Real-World Use
  - DBRecord
  - SimpleDBUI
  - SimpleDBJiflet
- Summary

---

framework \fram-wurk\ n: *a basic structure*

Introduction

The previous four chapters have introduced you to some new and exciting Java classes, all of which can be used in the development of intranet applications. However, as individual classes, they have no context, no greater purpose in life. You're probably wondering how you can use all of their features in your own applications as well. This chapter gives you

the direction you need to use the classes.

First, the concept of Java *packages* is discussed. These allow you to bundle classes together in a manner that is easily maintained and (re)used. The classes then have a purpose. They are no longer individual classes; together they become a class library.

You then explore a few new classes that draw upon the power of the new package. These new classes, in conjunction with the foundation classes discussed in the previous chapters, become the *Java Intranet Framework*, or *JIF*. JIF applications implement the model intranet application as discussed in Chapter 7, "A Model Intranet Application." As you see, this framework does much of the drudgery for you and allows you to concentrate your efforts on the content of the application.

Java Compilation Basics

Programming languages typically exist at a higher level. This means that after you create source code, it is translated or *compiled* into a language that the computer can understand. After all of your source code has been translated, the final result is assembled or *linked* into a single entity. This entity is your program. It has been translated from the higher-level language into a language your computer can read and execute for you. Figure 12.1 illustrates this concept.

Figure 12.1 : *The creation of a program.*

The files created in this intermediate step are typically called *object modules*. These modules are pockets of CPU instructions that are created from your source code. These modules are then concatenated and massaged to produce your program.

Java Source Code Files

Creating Java programs entails writing a set of classes. You create Java classes in source code format. These files have the `.java` extension on them. Typically, each `.java` file contains the source code for a single class. This class name is also the name of the file. For example, if you have a class called `LightBulb`, it is stored in a file called `LightBulb.java`.

Tip
A `.java` file might contain the source code for more than one class. However, only one of the classes in the source code file can be declared `public`. This allows you to bundle a private class with another class that uses it. A good example of this bundling is for manager classes. You can create a class that manages a collection of objects. The objects can be represented by an entire class. Because it is a private class, you can store it in the same source code file with your manager class. Look in the `java.sql.DriverManager` class on the CD-ROM for an example of this.

Some languages such as Visual Basic or PowerBuilder actually compile your program into tokens or pseudocode. These instructions are used to tell an interpreter what to do. The final result of these languages is actually an interpreter bundled with the tokenized code.

Java is different from typical languages but similar to languages such as Visual Basic. Your Java source code is compiled into a pseudocode format called *byte code format*. But instead of creating object modules, the Java compiler creates *class* files that hold this byte code format. These files end with a `.class` extension. Using the `LightBulb` class as an example, it is compiled into a file called `LightBulb.class`.

Each of these class files contains the byte code for a single Java class. These class files can then be executed by the *Java Virtual Machine* or *JVM*. It might seem like a fancy name for an interpreter but it truly is a machine. Think of the JVM as a software engine. Your Java classes are the gasoline that makes it run. But unlike typical programming languages, the final

linkage is not performed for Java programs.

What? No Linker?

That's right, Java performs *runtime linking*. This is a function of the JVM. As your Java program runs, the JVM loads each class as it needs it. When your class no longer exists in your program, the JVM discards the memory copy. But how can it find all those classes? They can be stored almost anywhere.

The JVM searches for these classes on your hard disk using a predefined set of directories. These directories are specified using an environment variable called `CLASSPATH`. It holds the list of places to look for your classes.

`CLASSPATH`

In order for the JVM to find your classes, you must tell it where to start looking. This is done with the `CLASSPATH` environment variable. This variable contains a list of the directories on your hard drive that contain Java `.class` files. The list is separated by a colon (`:`) or a semicolon (`;`) depending on your computer platform.

Note
UNIX systems use the colon (`:`), and Microsoft Windows-based systems use the semicolon (`;`).

Here is an example of a Microsoft Windows platform `CLASSPATH`:

```
.;c:\jdk\lib\classes.zip;c:\JavaWork\Classes;
```

This tells the JVM to look in the current directory first (`.`), then search in the ZIP file `c:\jdk\lib\classes.zip`, and then look in the `c:\JavaWork\Classes` directory.

A similar `CLASSPATH` for UNIX looks like this:

```
.:/jdk/lib/classes.zip:/JavaWork/Classes
```

The JVM searches in the current directory and then searches in the `/jdk/lib/classes.zip` file, and finally searches in the `/JavaWork/Classes` directory.

Tip
It is possible to specify the `CLASSPATH` without using an environment variable. To do this, you must use the `-classpath` argument when running the Java interpreter `java`. For example, the command `java -classpath .;c:\jdk\lib\classes.zip;c:\JavaWork\Classes LightBulb` sets the `CLASSPATH` and runs the program `LightBulb`

If you stick with this model and make 500 classes, you have to store them in the `c:\JavaWork\Classes` directory. This becomes cluttered and very disorganized. Isn't there a better way to organize the classes? Glad you asked, because there is.

You can organize your code into *packages*.

Have You Got the Package?

In typical programming environments, object modules can be stored in what are known as *code libraries*. These libraries can consist of any object module you really care to place into them. Libraries are excellent places to store reusable code, because many linkers can read code out of code libraries in addition to reading the object modules themselves.

But because Java doesn't really have any object module per se, it must use a different method for archiving classes. Enter the Java package.

A package is a collection of Java classes. These classes usually have nothing in common but their purpose because Java packages are used to bundle functionality. You really shouldn't place utility classes in a graphics package.

Java itself comes bundled in six major packages. These are shown in Table 12.1.

Table 12.1. The Java packages.

Package	*Description*
`java.applet`	Applet classes
`java.awt`	Advanced Windowing Toolkit classes
`java.io`	Input and Output classes
`java.lang`	The Java language classes
`java.net`	Networking classes
`java.util`	Utility classes

The name of the package is very important. As you can see in Table 12.1, all of the Java packages start with `java` followed by a period and then the rest of the package name. Not only does this clarify the name of the package, but it defines the placement of these classes within the `CLASSPATH`.

A Sample Package

For example, suppose you create a set of packages for producing music. The set consists of three packages: a utilities package, a user interface package, and a sound package. These are named `music.util`, `music.ui`, and `music.sound`, respectively.

When the JVM looks for these classes, it replaces the period with a slash and appends the result to each directory in the `CLASSPATH` variable. It then tries to load this file. The name of the package additionally defines where it is stored in the `CLASSPATH`.

In this example, the music classes are under the `c:\JavaWork\Classes` directory. Then they are stored as follows:

`music.util` is stored in `c:\JavaWork\Classes\music\util`
`music.ui` is stored in `c:\JavaWork\Classes\music\ui`
`music.sound` is stored in `c:\JavaWork\Classes\music\sound`

The classes that make up each package are then stored in their respective directories.

Making Java Packages

Creating packages out of your source code is quite simple. The first step is to create a directory structure that is like the package name you use. Using the preceding example, you create the `c:\JavaWork\Classes\music` directory and then the three directories-`util`, `ui`, and `sound`-underneath it.

All of the source code that belongs in each package is then moved to its appropriate directory. All of the utility classes move to `util`, user interface classes move to `ui`, and sound classes move to `sound`.

The final step is to add a `package` statement at the top of your source code file that tells the compiler which package your class belongs to. This statement must be the first non-comment or blank line in your source code.

The format for a `package` statement is as follows:

```
package package_name
```

`package_name` is the fully qualified name of the package. For example, classes in the `music.ui` class have the

following `package` statement:

```
package music_ui
```

Placing your class into a package has a very important implication. Your class now has access to all `public` and non-`public` classes in its package. Your class also has access to all non-`private` methods and instance variables of each other class in the package.

Caution
When you define an instance variable and do not assign it an access modifier (for example, `private`, `protected`, `public`), it defaults to `package`. This is essentially `private` but all classes within the same package can see it.

A similar arrangement exists in the C++ programming language. You can optionally declare a class to be a `friend` of another class. This allows it access to any non-`private` methods or instance variables in the target class. Think of packages as automatic friends.

Introducing the Java Intranet Framework

Chapter 7, "A Model Intranet Application," defines the functionality that a typical intranet application has. To recap, here are the features:

- Standard configuration file processing
- Standard logging to screen or disk
- Standard database connectivity
- Standard look and feel

In Chapters 8 through 10, you designed and created many classes that implement these features. For instance, the `ConfigProperties` class implements the configuration file processing. And the `DiskLog` class implements your standard logging.

If you take a step back and look at what you've created, it's really a framework for developing intranet applications with Java. It truly is a Java intranet framework! Therefore, you name this collection of classes the *Java Intranet Framework*. And like any good programming toolkit or library, it has a unique and pronounceable acronym: JIF.

But now that you've created all of these classes, you need to organize them into a usable set of packages. This helps you to document, use, and later extend, JIF to its fullest potential. So how do you package the classes in JIF?

Packaging the JIF Classes

The best way to package JIF is in groups of functionality. As you recall, you have four functionality groups: utilities, logging, database, and user interface. These fit perfectly into four functionality packages.

Naming your packages is quite simple. You use `jif` as your base package name and then add on the specifics. For example, the utilities package is called `util`, as in `java.util`. Table 12.2 shows the package names you have chosen and the class functionality that is contained within them.

Table 12.2. The Java Intranet Framework packages.

Package	*Description*
`jif.awt`	Standard user interface classes and `java.awt` extensions
`jif.log`	Standard logging classes
`jif.sql`	Standard database connectivity classes and `java.sql` extensions
`jif.util`	Standard utility classes and `java.util` extensions

The `jif.sql` package name does not really contain SQL functionality. However, the name is chosen to be consistent with the JDBC class hierarchy that lives in the `java.sql` package.

To package your classes, you must go back and edit each Java source code file. In each file you add a line of code that defines the package to which the class belongs. These statements are `package` statements, as discussed earlier in this chapter.

In addition to adding the `package` statements to our source code, you must also move all of the classes into a directory hierarchy that represents your package hierarchy. This hierarchy is illustrated in Figure 12.2.

Figure 12.2 : *The JIF class/directory hierarchy.*

Remember that the base of this hierarchy can live anywhere. It can be a root directory or a subdirectory. However, the directory that contains it must be in your `CLASSPATH` variable in order for the JVM to find the JIF classes.

Extending the Framework

Now that you've sorted your classes into usable packages, you can start building intranet applications. However, the classes really give you no direction or structure.

An excellent implementation of application direction and structure is Java's own `Applet` class. This class is a self-contained mini-application that runs in a special applet-viewer program or from an HTML World Wide Web page.

Java Applets

Creating applets is an easy task. Granted, the more complex your program is, the more complex your applet is. However, applets can be quite simple because they have an entire framework beneath them to support their simplicity. The smallest applet possible is shown in Listing 12.1. The HTML file that launches the small applet is shown in Listing 12.2, and the output of the applet is shown in Figure 12.3.

Figure 12.3 : *The smallest applet.*

Listing 12.1. The smallest applet.

```
//*********************************************************************
//* Imports                                              &nb
sp;                          *
//*********************************************************************

//    Java Imports...
import                              java.applet.Applet;
import                              java.awt.Label;

//*********************************************************************
//* SmallApplet
;                         *
//*********************************************************************

public class
SmallApplet
extends Applet
{

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

```
    //* init                                                                *
    //**********************************************************************

        public void
        init()
        {
            add( new Label( "I'm a small applet" ) );
        }

    //**********************************************************************
    //* run                                                                 &
    nbsp;                                                                   *
    //**********************************************************************

        public void
        run()
        {
            show();
        }
    }
```

As you can see, the `init()` method is used to implement the user interface for your applet. After the `init()` method is called by the applet framework, the applet framework then calls the `run()` method. In the preceding example, the `run()` method does nothing more than show the applet on the screen.

Listing 12.2. The smallest applet's HTML page.

```
<HTML>
<HEAD>
<TITLE> A small applet </TITLE>
</HEAD>
<BODY>

<APPLET CODE="SmallApplet.class" WIDTH=200 HEIGHT=60></APPLET>

</BODY>

</HTML>
```

The applet framework provides several other useful facilities. One feature is the `getParameter()` method. This retrieves a parameter from the HTML file that launches the applet.

For example, consider the HTML file shown in Listing 12.2. It contains two parameters, `WIDTH` and `HEIGHT`. Through the use of the `getParameter()` method, you can retrieve the value specified after the equal sign (=), like this:

```
int myWidth = Integer.parseInt( getParameter( "WIDTH" ) );
```

This holds true for any property pair that appears in the HTML file. This is a convenient way to pass parameters to applets with an equally easy method of retrieving them.

What if you make JIF as easy to use as an applet? What if you create intranet applications that implement the model application features as easily as creating applets? Let's explore that possibility.

Making JIF Easy to Use

The first and most important thing you need to do is choose a name for your new bundle of joy. Let's call it a *jiflet*. Because you basically take many of the applet ideas from Java and place them into your own new class, it seems rather appropriate. Now that you have that out of the way, you define what your jiflet does.

The jiflet is a Java application that provides the features of your model intranet application in an easy-to-use wrapper. These features include your model features outlined earlier, plus many of the features in a standard Java applet.

One such feature is an `init()` method that is called automatically by the framework. This is a centralized location to place all of the initialization code.

Another feature is getting configuration parameters. You are able to retrieve parameters from the configuration file with a method such as the `java.Applet.getParameter()` method.

In fact, the following methods are available in both applets and jiflets:

- `public void init()`
- `public boolean isActive()`
- `public void destroy()`
- `public String getParameter( String name )`
- `public void resize(Dimension d)`
- `public void resize(int width, int height)`
- `public void showStatus(String msg)`

The only features really missing from jiflets that appear in applets are the multimedia methods. These methods provide a clean way for applets to load images and sound files off of a network. Because jiflets represent intranet applications, these features are not usually necessary.

With the design goals laid out, let's look into implementation of the classes.

The `JifApplication` Interface

To implement the jiflet, you employ the use of an interface. This interface defines the pattern or template to which all jiflets must adhere. There is a single method defined in `JifApplication`. In fact, Listing 12.3 shows the source code for `JifApplication`.

Listing 12.3. The `JifApplication` interface.

```
//*************************************************************************
//* JifApplication                                                    &n
bsp;                  *
//*************************************************************************

public interface
JifApplication
{

//*************************************************************************
//* init
                            *
//*************************************************************************

    public void
    init();
```

```
}
```

This interface requires all jiflets to have an `init()` method. The rest of the methods that you've designed into the class exist in the `Jiflet` class. However, the `init()` method must be created by the user of the class. Using this interface allows you to declare your `Jiflet` class `abstract`. This means that you cannot instantiate the `Jiflet` class alone. You must always derive a class from it.

The `Jiflet` Class

The `Jiflet` class is the intranet application version of Java's own `Applet`. It provides you with a framework to develop intranet applications quickly and easily. It brings together all of the usefulness of the other JIF packages into a single, easy-to-use, component.

The `Jiflet` class implements the `JifApplication` interface as well as the `Log` interface (described in Chapter 9, "Logging Classes"). This allows it to have an `init()` method and adhere to the standard log interface.

Let's take a look at the `Jiflet` class in detail. You start with the instance variables. You then move to the constructors and then on to each of the methods. After finishing this chapter, you should have a good understanding of the `Jiflet` class and how to use it.

Instance Variables

The `Jiflet` class contains the following instance variables:

```
protected boolean               activeFlag = false;
protected DiskLog               appLogFile;
protected String                appName;
protected boolean               appVerbosity = false;
protected ConfigProperties      configProperties;
protected ScreenLog             defaultLog;
private DBConnector             myConnector = null;
protected StatusBar             myStatusBar = null;
private int                     oldCursor = -1;
```

The following sections look at each one and how it is used.

`activeFlag`

```
protected boolean               activeFlag = false;
```

The `activeFlag` is used to denote the activity of a jiflet. Its state can be queried by way of the `Jiflet.isActive()` method. It is set to true right before the `run()` method is called and set to false after the `destroy()` method is called.

`appLogFile`

```
protected DiskLog               appLogFile;
```

The `appLogFile` is the log file object for the jiflet. During construction, this object is created like so:

```
appLogFile = new DiskLog( logPath, DiskLog.createLogFileName(), appName
);
```

`logPath` is a configurable location in which to place all log files. The `DiskLog.createLogFileName()` method creates a standard log file name. Finally, `appName` is used on the standard log entry to identify the application that generates it.

appName

```
protected String                appName;
```

This string holds the name of the application that is running. This is usually passed in at construction.

appVerbosity

```
protected boolean               appVerbosity = false;
```

If you choose, your jiflet can be configured to report more information about certain things. This *verbosity* level is turned on or off with this Boolean instance variable. It defaults to `false` and can be set with the `Jiflet.setVerboseMode()` method.

configProperties

```
protected ConfigProperties      configProperties;
```

This object holds the combined program arguments and the configuration parameters read from the application's configuration file. Access to this variable is through the `Jiflet.getParameter()` methods.

defaultLog

```
protected ScreenLog             defaultLog;
```

This variable is a default log in case the real application log cannot be created. This log writes its entries to the standard out of the operating system.

myConnector

```
private DBConnector             myConnector = null;
```

This variable holds the `DBConnector` object that is associated with this jiflet. You can set and get this variable with the `Jiflet.setConnector()` and `Jiflet.getConnector()` methods.

myStatusBar

```
protected StatusBar             myStatusBar = null;
```

This variable holds the instance of the `StatusBar` object that is created for the jiflet. The status can be set or cleared with the `Jiflet.showStatus()` and `Jiflet.clearStatus()` methods.

oldCursor

```
private int                     oldCursor = -1;
```

This private variable is used to store the value of the cursor while a "wait" cursor is displayed. This variable is used by the `Jiflet.startWait()` and `Jiflet.endWait()` methods.

Constructors

There are four ways to construct a jiflet. These are defined by four separate constructors. A constructor is the function that is called when an instance of your object is being created. Each of them are useful for different purposes. For the most part, most of your jiflets use the fourth incarnation. Let's take a look at each constructor.

Three of the constructors call the fourth constructor. This master constructor is where all the jiflet initialization takes place.

The following is the complete source code for this constructor:

```
    /**
     * Creates a Jiflet with a title, a name, arguments, and optionally
     * verbose.
     *
     * @param title The window title
     * @param name The name of the application
     * @param args The arguments passed in to the program
     * @param verbosity On/Off setting indicating verbosity of log entries
     * @see #setVerboseMode
     */
    public
    Jiflet( String title, String name, String args[], boolean verbosity )
    {
        //    Call the superclass...
        super( title );

        //    Copy title to name...
        if ( name.equals( "" ) )
            name = title;

        //    Set the color...
        setBackground( Color.lightGray );

        //    Center and show our window!
        center();

        //    Add a status bar...
        enableStatusBar();

        //    Save my application name...
        appName = name;

        //    Create a default log....
        defaultLog = new ScreenLog( appName );

        //    Parse any passed in arguments...
        //    Parse the configuration file if available...
        configProperties = new ConfigProperties( args, appName + ".cfg" );

        //    Reset the title...
        setTitle( getParameter( "Title", title ) );

        //    Construct a log file name...
        String logPath = getParameter( "LogPath", "" );

        //    Open the log file...
        try
        {
            if ( logPath.equals( "" ) )
            {
                appLogFile = new DiskLog( appName );
            }
```

```
            else
            {
                appLogFile = new DiskLog( logPath,
                    DiskLog.createLogFileName(),
                    appName );
            }
        }
        catch ( IOException e )
        {
            //    Write errors to the screen...
            errorLog( "Error opening log file for [" +
                appName + "] (" + e.toString() + ")" );

            appLogFile = null;
        }

        //    Turn on verbose mode...
        setVerboseMode( verbosity );

        //    Denote construction...
        log( "Application [" + appName + "] started at " +
            ( new Date() ).toString() );

        //    Call my init!
        init();

        //    We are now active!
        activeFlag = true;

        //    Call my run...
        run();
    }
```

Because the `Jiflet` class descends from Java's `Frame` class, you need to call the superclass's constructor with a title, which is what you do first.

The following are done next:

- The background color is set to light gray.
- The jiflet is centered on the screen.
- The status bar is created and enabled.
- The default log is opened.
- The configuration file is read into memory and combined with the program arguments.
- The `LogPath` configuration option is retrieved from the configuration.
- The actual disk log file is created.
- Verbose mode is turned on or off depending on the value passed in.
- A message is written to the log stating that the jiflet has started.
- The user's implemented `init()` method is called.
- The `run()` method is called. By default, the `show()` method of the `Frame` is called to display the jiflet on the screen.

The following are the other three constructors available for creating `Jiflet` objects:

```
public
Jiflet()
{
    this( "Generic Jiflet", "Jiflet", null, false );
}

public
Jiflet( String title )
{
    this( title, "", null, false );
}

public
Jiflet( String title, String name, String args[] )
{
    this( title, name, args, false );
}
```

As you see, they all call the main constructor, setting some values to `null` or blanks.

Methods

There are many methods available in the `Jiflet` class. The following sections document their arguments and the purposes they serve.

`setVerboseMode`

```
public void
setVerboseMode( boolean whichWay )
```

This method turns on or off verbose mode. If verbose mode is turned on, all log entries created with the `Jiflet.verboseLog()` method are actually passed to the log object. If verbose mode is turned off, all log entries created this way are ignored.

Tip
This, in conjunction with `Jiflet.verboseLog()`, offers an excellent debugging tool. Simply enable verbose mode when you need more detail, and in your code provide that detail with the `Jiflet.verboseLog()` method.

`verboseLog`

```
//*************************************************************************
//* verboseLog
                        *
//*************************************************************************

    public void
    verboseLog( char logLevel, String logEntry )

    public void
    verboseLog( String logEntry )
```

This method creates a log entry that is written to the log file only if the verbose mode flag is set to true.

The second constructor defaults to a log level of `I` for informational.

errorLog

```
    //************************************************************************
    //* errorLog                                                &n
    bsp;                        *
    //************************************************************************

        public void
        errorLog( String logEntry )
```

This method allows you to create error log entries without specifying the `Log.ERROR` logging level each time. It is simply a convenience method.

log

```
    //************************************************************************
    //* log                                             &
    nbsp;                           *
    //************************************************************************

        public void
        log( char logLevel, String logEntry )

        public void
        log( String logEntry )
```

This method creates a log entry that is written to the log file. If there is an error or the log file is not open, the output goes to the screen.

The second constructor defaults to a log level of `I` for informational.

handleEvent

```
    //************************************************************************
    //* handleEvent
    ;                       *
    //************************************************************************

        public boolean
        handleEvent( Event anEvent )
```

This overrides the default `Frame.handleEvent()` method. It listens for destruction events so that the jiflet can close itself down cleanly.

action

```
    //************************************************************************
    //* action                                          &nbs
    p;                          *
    //************************************************************************

        public boolean
        action( Event event, Object arg )
```

This method receives `ACTION_EVENT` events from the event system. It listens for menu events and passes them to `Jiflet.handleMenuEvent()`.

handleMenuEvent

```
//************************************************************************
//* handleMenuEvent                                                  &
nbsp;                 *
//************************************************************************

    protected boolean
    handleMenuEvent( Event event, Object arg )
```

This is a placeholder for menu events. This does nothing in the `Jiflet` class. It must be overridden by derived classes to include any functionality.

shutDown

```
//************************************************************************
//* shutDown                                                  &n
bsp;                    *
//************************************************************************

    public boolean
    shutDown( int level )
```

This method is the central point of exit for the jiflet. With the exception of a program crash, the jiflet always exits through this method. It is responsible for writing a log entry for the application ending, and it calls the `Jiflet.destroy()` method. The `level` argument is passed to the operating system as a return value for the calling program.

suicide

```
//************************************************************************
//* suicide                                                 &nb
sp;                     *
//************************************************************************

    public void
    suicide( Exception e, String logLine, int level )

    public void
    suicide( String logLine )

    public void
    suicide( String logLine, int level )

    public void
    suicide( Exception e )

    public void
    suicide( Exception e, String logLine )
```

This method allows a jiflet to gracefully kill itself. Depending on the circumstances, you may or might not want a log entry written, and you may or may not have an `Exception` that caused your death.

center

```
    //************************************************************************
    //* center                                              &nbs
    p;                        *
    //************************************************************************

        public void
        center()
```

This method centers the jiflet window on the screen.

enableStatusBar

```
    //************************************************************************
    //* enableStatusBar                                        &
    nbsp;                  *
    //************************************************************************

        public void
        enableStatusBar( String text )

        public void
        enableStatusBar()
```

This method creates a status bar with or without text and adds it to the jiflet's layout.

clearStatus

```
    //************************************************************************
    //* clearStatus
    ;                      *
    //************************************************************************

        public void
        clearStatus()
```

This method clears the text in the status bar.

showStatus

```
    //************************************************************************
    //* showStatus
                           *
    //************************************************************************

        public void
        showStatus( String text )
```

This method sets the text in the status bar.

setConnector

```
    //************************************************************************
    //* setConnector                                          &nbs
    p;                     *
```

```
    //************************************************************************
    *
        protected void
        setConnector( DBConnector aConnector )
```

This method sets the `DBConnector` object that is associated with this jiflet.

getConnector

```
    //************************************************************************
    //* getConnector                                          &nbs
    p;                    *
    //************************************************************************

        public DBConnector
        getConnector()
```

This method returns the previously associated `DBConnector` object to the caller.

startWait

```
    //************************************************************************
    //* startWait                                            &
    nbsp;                     *
    //************************************************************************

        public void
        startWait()
```

This is a luxury method. It changes the cursor to the system default "wait" cursor, which is usually an hourglass to indicate that a lengthy process is occurring.

endWait

```
    //************************************************************************
    //* endWait                                             &nb
    sp;                     *
    //************************************************************************

        public void
        endWait()
```

This method returns the cursor to its previous state if called after a `Jiflet.startWait()` call. Otherwise, this method does nothing.

getParameter

```
    //************************************************************************
    //* getParameter                                          &nbs
    p;                    *
    //************************************************************************

        public String
        getParameter( String key )

        public String
```

```
getParameter( String key, String defaultValue )
```

This method is identical to Java's `Applet.getParameter()` method. It returns the value associated with the key passed. You can also pass in a default value in case the key is not found. These parameters are looked for in the `configProperties` instance variable.

canClose

```
//************************************************************************
//* canClose                                                  &n
bsp;                        *
//************************************************************************

    public boolean
    canClose()
```

This method is called before the jiflet is allowed to close. It allows you to have a place to catch an unwanted departure. The default implementation returns true. Override this method to add your own functionality. An example of its use is to catch users before exiting if they haven't saved their work.

Within your overridden copy, you can ask users whether they want to save their work. If they say no, return `true`, closing the jiflet. If they say yes, save their work and then return `true`, closing the jiflet. You can also offer them a cancel option, which returns `false`.

isActive

```
//************************************************************************
//* isActive                                                  &n
bsp;                        *
//************************************************************************

    public boolean
    isActive()
```

This method returns `true` or `false` if the jiflet is currently active. This is similar to the `Applet.isActive()` method. A jiflet is considered active right before its `run()` method is called.

Wrapping Up Jiflets

Because you have these two new classes, you need to place them somewhere. Again you borrow from the example set by Sun and place them in a package called `jif.jiflet`. This package contains the `JifApplication` and `Jiflet` classes.

Programming with Jiflets

Now that you've created a class that implements the design goals, let's take it for a test drive. Remember your simple small applet earlier in this chapter? Let's see if you can make a jiflet just as easily.

The Smallest Jiflet

The smallest possible jiflet is similar to the smallest possible applet. It simply prints a string on the screen and does nothing else. Let's look at the source code for your smallest jiflet:

```
//************************************************************************
//* Imports                                                  &nb
```

```
sp;                          *
//***************************************************************************

import                              jif.jiflet.Jiflet;
import                              java.awt.Label;

//***************************************************************************
//* SmallJiflet
;                       *
//***************************************************************************

public class
SmallJiflet
extends Jiflet
{

//***************************************************************************
//* main
                              *
//***************************************************************************

    public static void
    main( String[] args )
    {
        new SmallJiflet();
    }

//***************************************************************************
//* init
                              *
//***************************************************************************

    public void
    init()
    {
        add( "Center", new Label( "I'm a small jiflet" ) );
        pack();
    }

//***************************************************************************
//* run                                          &
nbsp;                                *
//***************************************************************************

    public void
    run()
    {
        show();
    }

}
```

It is very similar to the small applet in Listing 12.1. The only additions are a `main()` method and a call to the jiflet's `pack()` method.

The `main()` method is called by the Java interpreter to run your program. It is required to create the class that is your program. Here you create an instance of `SmallJiflet`.

The `pack()` method readjusts the size of the jiflet to accommodate all the user interface objects that are contained within it. This is necessary because, unlike an applet, you have no predefined width or height.

The output of your small jiflet is shown in Figure 12.4.

Figure 12.4 : *The output of* SmallJiflet.

The `HelloWorld` Jiflet

Do you think that you aren't going to get a `Hello World` jiflet? Think again. It is one of the simplest of all jiflets. All programming languages start with a program that prints `Hello World!`, so why is Java any different?

The source code in Listing 12.4 is the complete source code for the `HelloWorld` application that is on the CD-ROM. This is the smallest jiflet possible.

This jiflet is a bit more complex than the previous smallest jiflet. In `HelloWorld` you add a menu and an About box. You also take advantage of the `handleMenuEvent()` functionality in your base class.

Listing 12.4. The `HelloWorld` application.

```
//************************************************************************
//* Imports                                                  &nb
sp;                        *
//************************************************************************

//    JIF Imports
import                              jif.jiflet.*;
import                             jif.awt.*;

//    Java Imports...
import                             java.awt.*;

//************************************************************************
//* HelloWorld
                       *
//************************************************************************

public class
HelloWorld
extends Jiflet
{

//************************************************************************
//* Members                                                  &nb
sp;                        *
//************************************************************************

    //    A message for the about box...
    String                         copyright = "HelloWorld v1.00\n" +
        "Copyright (c) 1996 by Jerry Ablan\n" +
        "All Rights Reserved";
```

```
    //    Menu stuff...
    Menu                        helpMenu;

//************************************************************************
//* main
                            *
//************************************************************************

    public static void
    main( String args[] )
    {
        new HelloWorld( args );
    }

//************************************************************************
//* Constructor
;                       *
//************************************************************************

    public
    HelloWorld( String args[] )
    {
        super( "Hello World!", "HelloWorld", args );
    }

//************************************************************************
//* init
                            *
//************************************************************************

    public void
    init()
    {
        //    Initialize my menus...
       initializeMenus();

        //    Initialize my UI...
        initializeUI();
    }

//************************************************************************
//* initializeMenus                                          &
nbsp;                   *
//************************************************************************

    public void
    initializeMenus()
    {
       MenuBar mb = new MenuBar();

       helpMenu = new Menu( "&Help" );
       helpMenu.add( new MenuItem( "&About..." ) );
       mb.add( helpMenu );
```

```
        mb.setHelpMenu( helpMenu );

        setMenuBar( mb );
    }

//************************************************************************
//* initializeUI                                                &nbs
p;                    *
//************************************************************************

    public void
    initializeUI()
    {
        JifPanel ap = new JifPanel( Effects.RAISED, "Hello World!" );
        ap.setFont( new Font( "Helvetica", Font.BOLD, 18 ) );
        add( "Center", ap );
        pack();
    }

//************************************************************************
//* handleMenuEvent                                               &
nbsp;                 *
//************************************************************************

    protected boolean
    handleMenuEvent( Event event, Object arg )
    {
        String label = ( String ) arg;

        if ( label.equalsIgnoreCase( "&About..." ) )
        {
            MessageBox id = new MessageBox( this, "About...", copyright
);
            id.center( true );
            id.show();
            return( true );
        }

        //    Didn't handle it!
        return( false );
    }

}
```

Figure 12.5 shows the output of the `HelloWorld` program.

Figure 12.5 : *Someone says hi!*

The program is simple. First of all, it extends the `Jiflet` class. This gives you access to the functionality of the base class. Second, it provides an `init()` method. This method is called when the jiflet is initialized and ready for the user portion to be initialized. After that, all that is left is to respond to events that come up.

Extending Jiflets for Real-World Use

Now that you have a basic understanding of the Java Intranet Framework or JIF, the rest of this chapter extends the jiflet concept into a form that can easily be used for building database-aware intranet applications.

A jiflet, as you know, is the smallest application that can be built with JIF. However, it does absolutely nothing but look pretty. After building several applications with JIF, it becomes apparent that they all share much of the same code. Each application goes through the following life cycle:

1. Construct the jiflet.
2. Construct the user interface.
3. Handle events until program ends.

Pretty boring, but such is life when you are only binary information. The pattern that becomes evident to you is that each application has a monotonous bunch of initialization code and a user interface that needs to be copied from application to application. But this base initialization is not in the `Jiflet` class. Jiflets need to remain pure.

The result is three new abstract classes: `SimpleDBJiflet`, `SimpleDBUI`, and `DBRecord`.

The philosophy is that a `SimpleDBJiflet` has a user interface that is defined by a `SimpleDBUI`. They work together to present a pleasant atmosphere for the user. Together, they allow the user to manipulate a single set of database data. This database information is represented by the `DBRecord` class.

By extending these three abstract classes and basically filling in the blanks, you can create powerful database applications in a matter of hours!

DBRecord

The `DBRecord` class is the smallest portion of the three new classes. This class represents a single set of database data. The data that is represented by this class is defined in its subclasses; therefore, it is an abstract class. The source code for this class is shown in Listing 12.5.

Listing 12.5. The `DBRecord` class.

```
//*************************************************************************
//* Package                                                 &nb
sp;                        *
//*************************************************************************

package                               jif.sql;

//*************************************************************************
//* Imports                                                 &nb
sp;                        *
//*************************************************************************

//    JIF imports
import                              jif.sql.*;
import                              jif.awt.*;

//    Java imports
import                               java.sql.*;

//*************************************************************************
//* DBRecord                                                  &n
bsp;                       *
```

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

public abstract class
DBRecord
{

//****************************************************************************
//* Members                                                  &nb
sp;                          *
//****************************************************************************

    //    An indicator for data changes...
    protected boolean            dataChange = false;
    protected boolean            isNewRecord = false;

//****************************************************************************
//* Constructor
;                       *
//****************************************************************************

    public
    DBRecord()
    {
        clear();
    }

    public
    DBRecord( ResultSet rs )
    {
        parseResultSet( rs );
    }

//****************************************************************************
//* parseResultSet                                             &n
bsp;                    *
//****************************************************************************

    /**
    * Parses a "SELECT * ..." result set into itself
    */
    public boolean
    parseResultSet( ResultSet rs )
    {
        isNewRecord = false;
        return( isNewRecord );
    }

//****************************************************************************
//* update                                                  &nbs
p;                           *
//****************************************************************************

    /**
```

```
    * Requests update SQL from the JifPanel and sends it to the database
    * via the DBConnector object passed in.
    */
    public abstract boolean
    update( DBConnector theConnector, JifPanel ap );

//************************************************************************
//* deleteRow                                              &
nbsp;                     *
//************************************************************************

    /**
    * Constructs delete SQL for the current row
    */
    public abstract boolean
    deleteRow( DBConnector theConnector );

//************************************************************************
//* setDataChange                                            &nb
sp;                   *
//************************************************************************

    /**
    * Sets a flag indicating that data has changed...
    */
    public boolean
    setDataChange( boolean onOff )
    {
        dataChange = onOff;
        return( dataChange );
    }

//************************************************************************
//* clear
;                         *
//************************************************************************

    /**
    * Clears all the variables...
    */
    public void
    clear()
    {
        isNewRecord = true;
        setDataChange( false );
    }


//************************************************************************
//* canSave                                          &nb
sp;                       *
//************************************************************************
```

```
    /**
    * Checks to see if all required fields are filled in
    */
    public boolean
    canSave()
    {
        //    Everything is filled in!
        return( true );
    }

//************************************************************************
//* didDataChange                                          &nb
sp;                  *
//************************************************************************

    public boolean
    didDataChange()
    {
        return( dataChange );
    }

//************************************************************************
//* setNewStatus                                          &nbs
p;                   *
//************************************************************************

    public void
    setNewStatus( boolean how )
    {
        isNewRecord = how;
    }

//************************************************************************
//* getNewStatus                                          &nbs
p;                   *
//************************************************************************

    public boolean
    getNewStatus()
    {
        return( isNewRecord );
    }

}
```

The `DBRecord` class can be constructed with or without data. The data required to construct this class is a JDBC `ResultSet` object. This object is passed to the `parseResultSet()` method. In your derived class, you must override this method to read the data from the `ResultSet` into instance variables.

To complete the class, you must provide two additional methods in your derived class: `update()` and `deleteRow()`. `update()` is called to when someone wants you to save the information contained within yourself. `deleteRow()` is called when someone wants you to delete yourself from the database.

The class provides a `clear()` method, which you override to clear out your instance variables. This is called, for example, when the user presses the New or Clear buttons.

The `DBRecord` class has two indicators: `dataChange` and `isNewRecord`. These `boolean`s indicate that the data has changed in the record and whether the record exists in the database.

The `dataChange` value must be manually set and reset. This is done to some degree by the `SimpleDBJiflet` class. For example, when a record is saved to the database, the `dataChange` value is set to `false`. This setting is done by way of the `setDataChange()` method.

Access methods are provided for you to get at these indicators. `getNewStatus()` and `setNewStatus()` allow access to the `isNewRecord` indicator. `setDataChange()` and `didDataChange()` provide access to the `dataChange` indicator.

Finally, the class provides a method called `canSave()`. This method returns a Boolean indicating that the record is eligible for saving to the database. This eligibility depends completely on the table that it is representing. This method allows you to validate the data that the user has entered and give it the thumbs up or down.

Note
The `DBRecord` class is part of the `jif.sql` package.

A complete example is in order. Listing 12.6 is a complete `DBRecord` derivation for a table that represents Conference Rooms. This is an actual class created for the Conference Room Scheduling application in this book.

Listing 12.6. A `DBRecord` subclass.

```
//*************************************************************************
//* Package                                                 &nb
sp;                        *
//*************************************************************************

package                                jif.common;

//*************************************************************************
//* Imports                                                 &nb
sp;                        *
//*************************************************************************

//    JIF imports
import                               jif.sql.*;
import                               jif.awt.*;

//    Java imports
import                                java.sql.*;

//*************************************************************************
//* ConfRoomRecord                                                  &n
bsp;                     *
//*************************************************************************

/**
* A class that encapsulates a row in the conference room table...
*/
public class
ConfRoomRecord
extends DBRecord
```

```
{

//*************************************************************************
//* Constants                                              &
nbsp;                    *
//*************************************************************************

    public final static String     TABLE_NAME = "conf_room";

//*************************************************************************
//* Members                                                &nb
sp;                      *
//*************************************************************************

    //    A variable for each table column...
    public int                     room_nbr = -1;
    public int                     floor_nbr = -1;
    public String                 desc_text = "";

//*************************************************************************
//* Constructor
;                       *
//*************************************************************************

    public
    ConfRoomRecord()
    {
        clear();
    }

    public
    ConfRoomRecord( ResultSet rs )
    {
        parseResultSet( rs );
    }

//*************************************************************************
//* parseResultSet                                           &n
bsp;                    *
//*************************************************************************

    public boolean
    parseResultSet( ResultSet rs )
    {
        clear();

        try
        {
            //    Suck out the data...
            room_nbr = rs.getInt( "room_nbr" );
            floor_nbr = rs.getInt( "floor_nbr" );
            desc_text = rs.getString( "desc_text" );
            return( super.parseResultSet( rs ) );
```

```
            }
            catch ( SQLException e )
            {
                //    Signal an error...
                clear();
                return( false );
            }
        }

    //************************************************************************
    //* update                                                  &nbs
    p;                          *
    //************************************************************************

        /**
        * Requests update SQL from the JifPanel and sends it to the database
        * via the DBConnector object passed in.
        */
        public boolean
        update( DBConnector theConnector, JifPanel ap )
        {
            boolean        success = true;

            try
            {
                //    No update if nothing to do...
                if ( dataChange )
                {
                    String    sql;

                    //    Generate some SQL!
                    if ( getNewStatus() )
                        sql = ap.generateInsertSQL( TABLE_NAME );
                    else
                        sql = ap.generateUpdateSQL( TABLE_NAME );

                    if ( !sql.equals( "" ) )
                        theConnector.getStatement().executeUpdate( sql );
                }
            }
            catch ( SQLException e )
            {
                theConnector.errorLog( e.toString() );
                success = false;
            }

            return( success );
        }

    //************************************************************************
    //* deleteRow                                                 &
    nbsp;                       *
    //************************************************************************
```

```
    /**
    * Removes this record from the database...
    */
    public boolean
    deleteRow( DBConnector theConnector )
    {
        boolean        success = true;

        //    Nothing to do...
        if ( getNewStatus() )
            return( false );

        String sql = "delete from " + TABLE_NAME + " where room_nbr " +
            "= " + Integer.toString( room_nbr ) + " and floor_nbr " +
            "= " + Integer.toString( floor_nbr );

        try
        {
            theConnector.getStatement().executeUpdate( sql );
        }
        catch ( SQLException e )
        {
            theConnector.errorLog( e.toString() );
            success = false;
        }

        return( success );
    }

//************************************************************************
//* clear
;                          *
//************************************************************************

    /**
    * Clears all the variables...
    */
    public void
    clear()
    {
        super.clear();

        room_nbr = -1;
        floor_nbr = -1;
        desc_text = "";
    }

}
```

SimpleDBUI

The `SimpleDBUI` class encapsulates the nonvisual side of the user interface that is necessary for proper application functionality. It extends the `JifPanel` class providing some default buttons and methods for moving data to and from the user interface components.

The source code for this class is shown in Listing 12.7.

Listing 12.7. The `SimpleDBUI` class.

```
//************************************************************************
//* Package                                                  &nb
sp;                     *
//************************************************************************

package                             jif.awt;

//************************************************************************
//* imports                                                  &nb
sp;                     *
//************************************************************************

import                              java.awt.*;

import                             jif.sql.*;
import                             jif.jiflet.*;
import                             jif.common.*;

//************************************************************************
//* SimpleDBUI
                    *
//************************************************************************

public abstract class
SimpleDBUI
extends JifPanel
{

//************************************************************************
//* Members                                                  &nb
sp;                     *
//************************************************************************

    SimpleDBJiflet                  myJiflet;

    //    Some standard buttons...
    public Button                  saveButton = new Button( "Save" );
    public Button                  clearButton = new Button( "Clear" );
    public Button                  newButton = new Button( "New" );
    public Button                  deleteButton = new Button( "Delete" );
    public Button                  chooseButton = new Button( "Choose" );
    public Button                  closeButton = new Button( "Close" );

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

```
    //* Constructor
    ;                       *
    //*************************************************************************

        public
        SimpleDBUI( SimpleDBJiflet jiflet )
        {
            setJiflet( jiflet );
            setFont( new Font( "Dialog", Font.PLAIN, 12 ) );
        }

    //*************************************************************************
    //* getJiflet                                                 &
    nbsp;                       *
    //*************************************************************************

        public SimpleDBJiflet
        getJiflet()
        {
            return( myJiflet );
        }

    //*************************************************************************
    //* setJiflet                                                 &
    nbsp;                       *
    //*************************************************************************

        public void
        setJiflet( SimpleDBJiflet jiflet )
        {
            myJiflet = jiflet;
        }

    //*************************************************************************
    //* moveToScreen                                                &nbs
    p;                      *
    //*************************************************************************

        /**
         * Moves data from a DBRecord object to the fields on the screen
         * for editing.
         */
        public abstract void
        moveToScreen();

    //*************************************************************************
    //* clearScreen
    ;                       *
    //*************************************************************************

        /**
        * Clears the screen fields
        */
```

```
    public abstract void
    clearScreen();

//************************************************************************
//* moveFromScreen                                                  &n
bsp;                  *
//************************************************************************

    /**
     * Moves data from the fields on the screen to a DBRecord object.
     */
    public abstract void
    moveFromScreen();

//************************************************************************
//* action                                                  &nbs
p;                        *
//************************************************************************

    public boolean
    action( Event event, Object arg )
    {
        //    Smart JIF components generate ACTION_EVENTs when changed...
        if ( event.target instanceof SQLFactoru )
        {
            //    Notify dad...
            sendJifMessage( event, DATA_chANGE );
            return( true );
        }

        //    User pressed Save...
        if ( event.target == saveButton )
        {
            //    Notify dad...
            sendJifMessage( event, SAVE );
            return( true );
        }

        //    User pressed New...
        if ( event.target == newButton )
        {
            //    Notify dad...
            sendJifMessage( event, NEW );
            return( true );
        }

        //    User pressed Choose
        if ( event.target == chooseButton )
        {
            //    Notify dad...
            sendJifMessage( event, chOOSE );
            return( true );
        }
```

```
            //    User pressed Close
            if ( event.target == closeButton )
            {
                //    Notify dad...
                sendJifMessage( event, CLOSE );
                return( true );
            }

            //    User pressed Delete
            if ( event.target == deleteButton )
            {
                //    Notify dad...
                sendJifMessage( event, DELETE );
                return( true );
            }

            //    User pressed Clear
            if ( event.target == clearButton )
            {
                //    Notify dad...
                sendJifMessage( event, CLEAR );
                return( true );
            }

            //    Not handled...
            return( false );
        }

    }
```

Because it is `abstract`, this class is not very complex. The first thing you notice is that you create a slew of buttons:

```
    public Button               saveButton = new Button( "Save" );
    public Button               clearButton = new Button( "Clear" );
    public Button               newButton = new Button( "New" );
    public Button               deleteButton = new Button( "Delete" );
    public Button               chooseButton = new Button( "Choose" );
    public Button               closeButton = new Button( "Close" );
```

These are the standard buttons that the `SimpleDBUI` knows about. They are defined as `public` so that you can access them outside of the user interface. Unless they are placed upon a panel or shown in some manner on the screen, they are really never used; therefore, they do not generate messages.

When these buttons are shown on the screen and subsequently pressed by the user, an `ACTION_EVENT` event is generated. This event is translated into a `JifMessage` by the `action()` event handler method. The message is then sent on to the parent, presumably a `SimpleDBJiflet`, and processed there.

The `SimpleDBUI` class is expected to move data in and out of a `DBRecord` class. It does this in three methods: `moveToScreen()`, `moveFromScreen()`, and `clearScreen()`. This class has access to the current `DBRecord` by way of the jiflet. By calling the `SimpleDBJiflet`'s `getDBRecord()` method, a reference to the current `DBRecord` is provided.

`moveToScreen()` moves data from the `DBRecord` to the screen components. `moveFromScreen()` moves data from the screen components to the `DBRecord`. A `clearScreen()` clears out the screen components. This last method does

not touch the `DBRecord` really, but a `clearScreen()` followed by a `moveFromScreen()` clears out the `DBRecord`.

Note
The `SimpleDBUI` class is part of the `jif.awt` package.

A simple `SimpleDBUI` derivation is shown in Listing 12.8. This is from the Online In/Out Board application from Chapter 16, "Online In/Out Board."

Listing 12.8. A `SimpleDBUI` subclass.

```
//************************************************************************
//* imports                                                  &nb
sp;                          *
//************************************************************************

import                               java.awt.*;

import                              jif.awt.*;
import                              jif.sql.*;
import                              jif.jiflet.*;
import                              jif.common.*;

//************************************************************************
//* InOutBoardUI                                                 &nbs
p;                        *
//************************************************************************

public class
InOutBoardUI
extends SimpleDBUI
{

//************************************************************************
//* Members                                                  &nb
sp;                          *
//************************************************************************

    List                            empList;

//************************************************************************
//* Constructor
;                         *
//************************************************************************

    public
    InOutBoardUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );
        setLayout( new BorderLayout() );

        empList = new List();
        empList.setFont( new Font( "Helvetica", Font.BOLD, 14 ) );
        add( "Center", empList );
```

```
            empList.enable();

            JifPanel p = new JifPanel();
            p.setLayout( new FlowLayout( FlowLayout.CENTER, 5, 5 ) );
            saveButton.setLabel( "Toggle" );
            saveButton.disable();
            p.add( saveButton );
            add( "South", p );

            //    Set the focus to the first field...
            setFocus( empList );
        }

    //************************************************************************
    //* moveToScreen                                          &nbs
    p;                      *
    //************************************************************************

        /**
         * Moves data from an InOutBoardRecord object to the fields on the
    screen
         * for editing.
         */
        public void
        moveToScreen()
        {
            if ( getJiflet().getDBRecord() == null )
                return;

            //    Cast one off...
            EmployeeRecord er = ( EmployeeRecord )getJiflet().getDBRecord();

            String s = er.first_name + " " + er.last_name + " is ";

            if ( er.in_out_ind.equalsIgnoreCase( "Y" ) )
                s += "in";
            else
                s += "out";

            empList.addItem( s );
        }

    //************************************************************************
    //* clearScreen
    ;                       *
    //************************************************************************

        /**
        * Clears the record out...
        */
        public void
        clearScreen()
        {
```

```
            empList.clear();
        }

    //*************************************************************************
    //* moveFromScreen                                                    &n
    bsp;                  *
    //*************************************************************************

        /**
         * Moves data from the fields on the screen to an EmployeeRecord
    object.
         */
        public void
        moveFromScreen()
        {
            //    Does nothing…
            return;
        }

    }
```

SimpleDBJiflet

The `SimpleDBJiflet` class pulls together the `DBRecord` and `SimpleDBUI` classes into a cool little hunk of code. This class encapsulates much of the necessary menu and database initialization that must be done for each application.

The `SimpleDBJiflet` class extends the `Jiflet` class and adds the following functionality:

- File menu with standard database connectivity
- Help menu with working About box
- A standard method of communicating with the user interface
- Record saving and deleting
- Data modification notification

Although they are not functional on their own, these features offer quite a bit of legwork that you have to cut and paste from app to app. The beauty of object-oriented programming and Java is that you can stuff all of this functionality into an abstract base class and fill in the blanks. That is all that has been done here.

Note
The `SimpleDBJiflet` class is part of the `jif.jiflet` package.

File and Help Menu

The `SimpleDBJiflet` class creates two menus: a File menu, and a Help menu. The File menu contains two items: Connect and Exit.

The first option, Connect, connects and disconnects the application with the database. This functionality is provided completely as long as the jiflet has a valid `DBConnector` set for itself.

Note
For more information about the `DBConnector` class, see Chapter 10, "Database Classes."

After a database connection is established, this menu option changes to Disconnect automatically. When selected, it disconnects your application from the database and changes back to Connect.

Caution
The JDK v1.0.2 for 32-bit Microsoft Windows systems has a bug that prevents a menu item from changing the text after it is displayed. This should be fixed in the JDK v1.1 release.

The second menu option, Exit, disconnects any connected `DBConnector` and closes the application. This can include writing information to a log file or to the screen. It depends on the configuration of the jiflet.

The Help menu has a single menu item. This item brings up an About box. If you are not familiar with these boxes, they are simply brag boxes for the authors of programs. Some actually show useful information but most just show the program name and a fancy icon along with some copyright information.

Should your jiflet be any different? You're just as proud of your creation as other authors are! Well, set a copyright message with the method `setCopyright()` and an About box displays automagically! Figure 12.6 shows the About box for the employee maintenance program-nothing too fancy, just the text and a little icon.

Figure 12.6 : *The employee maintenance About box.*

Tip
A nice extension to the `Jiflet` class allows custom icons to be associated with the application. Then in their About boxes, the custom icon displays instead of the stock information icon.

The only code required to get that nice About box in the derived `Employee` program is the following:

```
setCopyright( "Employee Maintenance v1.00\n" +
    "Copyright (c) 1996 by Jerry Ablan\n" +
    "All Rights Reserved" );
```

Not a lot of code for such a nice feature! By the way, the About menu item is disabled until you call the `setCopyright()` method.

Standard Communications: `JifMessage`

In object-oriented programming, objects communicate with each other by way of messages. But at what point should objects know about the inside workings of other objects? Some purists argue never! Some argue sometimes. However, it is usually a matter of convenience.

While designing many of the applications for this book, I felt that the user interface should be able to manage itself, not knowing about the application driving it. However, I felt that the application needed to know a little about the user interface. Otherwise, you really can't provide any nice bells and whistles. One such feature is to enable and disable the Save button when a data item is modified. This is an excellent visual clue to the user that a change has been made, intentionally or not.

In order to feel politically correct OOP-wise, and to not let my user interface design creep into my application design, I created the `JifMessage` interface. Listing 12.9 is the source code for the `JifMessage` interface.

Listing 12.9. The `JifMessage` interface.

```
//************************************************************************
//* Package                                                &nb
sp;                    *
//************************************************************************
```

```
package                                 jif.jiflet;

//*************************************************************************
//* Imports                                                 &nb
sp;                         *
//*************************************************************************

import                                  java.awt.Event;

//*************************************************************************
//* JifMessage
                          *
//*************************************************************************

public interface
JifMessage
{

//*************************************************************************
//* Members                                                 &nb
sp;                         *
//*************************************************************************

    public static final int    NEW = 0;
    public static final int     CLEAR = 1;
    public static final int     SAVE = 2;
    public static final int     DELETE = 3;
    public static final int     CUT = 4;
    public static final int     COPY = 5;
    public static final int     PASTE = 6;
    public static final int     HELP_WINDOW = 7;
    public static final int     HELP_CONTEXT = 8;
    public static final int     HELP_ABOUT = 9;
    public static final int     HELP_HELP = 10;
    public static final int     DATA_chANGE = 11;
    public static final int     chOOSE = 12;
    public static final int     CLOSE = 13;

//*************************************************************************
//* sendJifMessage                                            &n
bsp;                      *
//*************************************************************************

    public void
    sendJifMessage( Event event, int msg );

}
```

Again, it is nothing fancy-simply a list of constants and a consistent method of sending them, which is up to the implementor of this interface. As you can see, many standard actions are represented by the constants in this class: New, Save, Delete, Close, and so on.

After creating this interface, it needs to be implemented somewhere. The `JifPanel` class is an excellent spot. Because

most user interfaces are created with `JifPanel`s, this provides a consistent and standard method of communication with its parent. Listing 12.10 is the code that is added to the `JifPanel` class to implement this interface.

Listing 12.10. Sending a `JifMessage`.

```
//************************************************************************
//* sendJifMessage                                                   &n
bsp;                  *
//************************************************************************

    public void
    sendJifMessage( Event event, int msg )
    {
        event.target = this;
        event.arg = new Integer( msg );
        getParent().postEvent( event );
    }
```

Nothing too tricky here either. The `sendJifMessage()` method takes as arguments an event and the message to send. It changes the target of the event to itself (the `JifPanel` instance) and sets the argument of the event to the message. It then sends the message along to the parent.

Here's a quick example of how it is used. In the `SimpleDBUI`, there is a built-in Save button. When the user clicks on this button, the class automatically sends a `JifMessage.SAVE` event to its parent. The parent needs only to listen for these `JifMessage` events to know what the child wants it to do.

The code for sending from the child looks exactly like this:

```
//    User pressed Save...
if ( event.target == saveButton )
{
    //    Notify dad...
    sendJifMessage( event, JifMessage.SAVE );
    return( true );
}
```

The code for receiving in the parent looks something like this:

```
public boolean
action( Event event, Object arg )
{
    switch ( ( ( Integer )arg ).intValue() )
    {
        case JifMessage.DELETE:
            delDlg = new ResponseDialog( this,
                "Delete Confirmation",
                "Are you sure you want to delete this record?",
                "Yes,No" );

            delDlg.show();
            break;
    }
}
```

Now that there is a standard way of communicating, you can add some standard features such as saving and deleting.

Record Saving and Deleting

A nice feature for a program to have is a standard method of saving and deleting. Now that you know when the user wants you to save or delete (by way of a `JifMessage`), you only need some standard methods for doing this.

Enter `saveRecord()` and `deleteRecord()`. These two methods provide a method for saving and deleting the information that is stored in the `DBRecord` of the jiflet. When the `SimpleDBUI` sends the `SAVE` or `DELETE` `JifMessage` to the jiflet, these methods are called.

They are declared as follows:

```
//************************************************************************
//* saveRecord                    *
//************************************************************************

    public boolean
    saveRecord()
    {
        //    If we are not connected, do nothing...
        if ( !getConnector().connected() )
        {
            MessageBox mb = new MessageBox( this, "Hold on there!",
                "You must connect with the database\n" +
                "before you can save any data.\n\n" +
                "Connect first, then try this again!",
                MessageBox.EXCLAMATION );

            mb.show();
            return( false );
        }

        //    Move the data back to the DBRecord...
        getUIPanel().moveFromScreen();

        //    Check to see if all fields are filled in...
        if ( getDBRecord().canSave() )
        {
            //    Save it...
            if ( getDBRecord().update( getConnector(), getUIPanel() ) )
            {
                //    Indicate that it was saved...
                getDBRecord().setDataChange( false );
                setStatus( "Record saved..." );
            }
            else
                setStatus( "Record not saved..." );

            return( true );
        }
        else
        {
            MessageBox mb = new MessageBox( this, "Cannot Save!",
                "All required fields must be entered!",
                MessageBox.EXCLAMATION );
```

```
            mb.show();
        }

        return( false );
    }

//************************************************************************
//* deleteRecord                                                  &nbs
p;                    *
//************************************************************************

    public boolean
    deleteRecord()
    {
        //    If we are not connected, do nothing...
        if ( !getConnector().connected() )
        {
            MessageBox mb = new MessageBox( this, "Hold on there!",
                "You must connect with the database\n" +
                "before you can delete any data.\n\n" +
                "Connect first, then try this again!",
                MessageBox.EXCLAMATION );

            mb.show();
            return( false );
        }

        //    Move the data back to the DBRecord...
        getUIPanel().moveFromScreen();

        //    Kill it!
        if ( getDBRecord().deleteRow( getConnector() ) )
        {
            //    Indicate that it was saved...
            getDBRecord().clear();
            getUIPanel().moveToScreen();
            setStatus( "Record deleted..." );
            return( true );
        }
        else
            setStatus( "Record not deleted..." );

        return( false );
    }
```

This allows you to override them in your derived classes, thus enhancing the functionality. One functionality is to modify two tables instead of one. Or perhaps your jiflet does not save any data but you use the save button and mechanism for some other sort of notification. It is up to you. Be creative!

Data Change Notification

The last function that the `SimpleDBJiflet` provides is data change notification.

When the `SimpleDBUI` contains one of the JIF `Component` extensions (such as `JifTextField`), it is notified when

the user changes the data. This notification is passed along to the `SimpleDBJiflet`. The `SimpleDBJiflet` then manages the enabling and disabling of the Save, New, and Delete buttons.

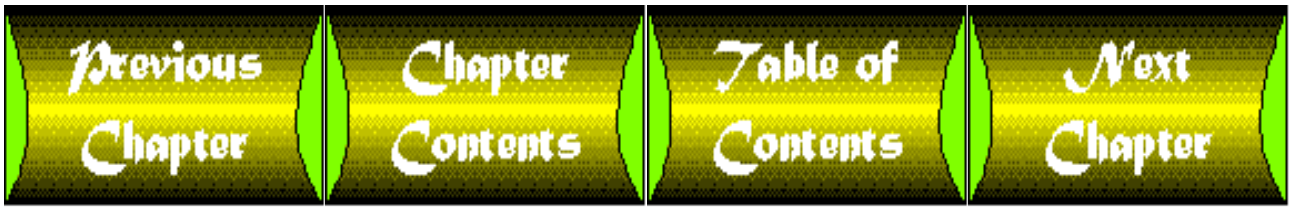
The status depends on the state of the `DBRecord` at the time. If the record is new, it can't be deleted but it can be saved and cleared (New). If the record has not been changed and is not new, it can be saved, deleted, or cleared. This is not a very complex set of rules but is a hassle to code for each application. You'll find it refreshing when your Save button lights up after you type in your first character.

Summary

This chapter covered every bit of the foundation from the smallest class to the largest. In this chapter, you have an intimate encounter with the `Jiflet` package. This package combines much of the other packages into a simple, usable application base for creating intranet applications. This application base is the goal set out in Chapter 7, "A Model Intranet Application."

This new base is the foundation for the rest of the book. You see that you create jiflets for each of the sample applications that are discussed in the following chapters.

In Chapter 13, "Employee Files," you discuss the first sample intranet application: Employee Files. This application allows you to maintain your employee information. It's not much, but it is a great starting point for the sample applications.



Chapter 13

Employee Files

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Introduction

Now that you've learned the foundation for creating intranet applications, let's get cooking. This chapter and the next several chapters guide you through some sample intranet applications. These applications are real-world examples. You don't find any animated buttons or scrolling text classes sprinkled in either. This is hardcore corporate database programming, but because you've developed such an excellent set of foundation classes, it isn't difficult at all to create the applications.

This chapter covers the following topics in regard to the Employee Files application:

- Application Design-This section covers the general design of the application, including functionality and user interface considerations.
- Database Design-This section covers the database requirements for this application. Here, you examine the data model used to support the application design.
- Implementation-This section covers how the application and database design are implemented.
- Programming Considerations-This section recaps the implementation and summarizes any difficult programming situations that have arisen.

This four-step format is used throughout the sample application chapters. Hopefully it provides you with valuable insight and ideas for creating your own intranet applications.

Application Design

This is the first sample application, and it is quite simple. It is the Employee Files. This application enables you to maintain records on the employees in your company. This application is sort of the "big toe" for the rest of the programs, because many of the other programs use information that is stored in the employee table. Programs such as the online phone book or the in/out board use this data.

Figure 13.1 is the proposed user interface for the Employee Files program.

Figure 13.1 : *The Employee files user interface.*

This application is semi-modeless, which means that it has no operating mode. You don't have to inform the program that you are going to be adding new records or removing records. You can flow through the program, and it determines what is done.

The basic functionality of this application is to create, read, update, and delete employee records. These records are stored in a database. You will learn about database design later in this chapter.

Think of this application as a pointer into the employee table. When the pointer is situated on a record, it is the current record. This current record is displayed to the user and the user can do whatever he chooses with it. The user can also insert records into the table at the end.

The user requires a method of moving this pointer from employee to employee. The best way to present this information to the user is through the use of a pick list.

Using a Pick List

The pick list is a selection of all the records in the employee table. This selection includes the first and last name of each employee. The user is allowed to select one name from the displayed list. After the selection is made, the chosen record is fully retrieved and displayed.

To select a record, double-click the list item or single-click and then press the OK button.

Figure 13.2 shows the concept of the employee pick list.

Figure 13.2 : *The employee pick list.*

This pick list is opened when the user presses the Choose button, a standard `SimpleDBUI` button.

Now that you have a sense of the application design, let's look at the database.

Database Design

This application is responsible for manipulating employee records. These records are stored in a single table. Currently, there is no need to extend the scope of the employee data to a second table.

The information stored in the employee table is basic information. Table 13.1 shows the columns needed in the employee table.

Table 13.1. The employee table layout.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
Employee ID	emp_id	number(5)	N	None
First Name	first_name	char(40)	N	None
Middle Name	mid_name	char(40)	Y	None
Last Name	last_name	char(40)	N	None
Social Security Number	ssn	char(15)	Y	None
Address Line 1	addr_line_1	char(80)	Y	None
Address Line 2	addr_line_2	char(80)	Y	None
City	city	char(80)	Y	None
State	state	char(80)	Y	None
Zip Code	zip_code	char(20)	Y	None
Salary	salary	number(7,2)	Y	None

Home Phone Number	home_phone_nbr	char(20)	Y	None
Work Extension Number	work_ext_nbr	char(20)	Y	None
In/Out Indicator	in_out_ind	char(1)	N	'N'

The entire data model is laid out into an *entity relationship diagram* or *ERD*. An ERD represents all the tables in your data model as an *entity*. The relationship between each entity is then shown. There are many types of relationships between entities: one-to-one, one-to-many, zero-or-more-to-one, zero-or-more-to-many, and so on.

Figure 13.3 shows the entity relationship diagram for the database as it stands in this chapter. As you get deeper into the sample applications, you see the entity relationship diagram grow to encompass all the tables.

Figure 13.3 : *The employee entity.*

In addition to creating a table, you create a database synonym for the table. This allows everyone to access the table with the same name and not have to worry about the schema in which the table resides.

Before you see the SQL for creating the table, there is a small matter of users and schemas. The data model for the sample applications in this book relies on no particular user or schema. However, the same model is used in the development of the applications.

All of the tables in this book are created by the master user for the database. Because you use Oracle for developing the database, this user is `system`. You also create an Oracle role and a user for the database. Full access to all of the tables is granted to the role. The user is granted access to the role.

Note

Although this book uses Oracle as a database, the table definitions are easily converted to other database management systems. After the tables are created in your own DBMS, this application and the others that follow will run just fine.

What's a Role?

A role is like a user group. It can have specific rights granted to it. These rights can be table access or possible system administration capabilities.

After a role is created, you can grant users access to the role. When a user is granted access to a role, that user can perform all of the functions granted to the role. Users can also be granted to several roles at one time. Many users can also be granted to a single role. This flexible structure allows complex database security schemes to be implemented without much work.

The role that you created is called `ia_user`, for "intranet application" user. The user that you created shares the same name, `ia_user`. The user is granted to the role. Therefore, all of the granting that is done in the creation SQL is only to the role.

Note

The SQL code to create the user and role is on the CD-ROM in the examples directory for this chapter. It is in a file called `user.sql`. This file is Oracle-specific, but should work on other databases with little or no modifications.

Finally, Listing 13.1 is the SQL commands to create the database.

Listing 13.1. The Employee table creation SQL.

```
/*    Create the table */
create table emp_t
(
    emp_id                number( 5 ) not null,
    first_name            char( 40 ) not null,
    mid_name            char( 40 ),
    last_name            char( 40 ) not null,
    ssn                    char( 15 ),
    addr_line_1            char( 80 ),
    addr_line_2            char( 80 ),
    city                char( 80 ),
    state                char( 80 ),
    zip_code            char( 20 ),
    salary                number( 7,2 ) not null,
    home_phone_nbr        char( 20 ),
    work_ext_nbr        char( 20 ),
    in_out_ind            char( 1 ) default 'N' not null
);
/*    Create a primary key */
alter table emp_t
    add
    (
        primary key
        (
            emp_id
        )
    );
/*    Create the synonym */
create public synonym emp for emp_t ;
```

Note
The SQL in Listing 13.1 is quite generic, but it might not work on every database. This particular SQL has been tested with Oracle.

Caution
The code in Listing 13.1 does not work with ODBC.

The first SQL clause creates the table `emp_t`. The second clause creates a primary key using the `emp_id` column. Making this the primary key ensures that the values in the column are unique across all rows. Lastly, the public synonym `emp` is created for the table `emp_t`.

After you create the table, you are ready to build the application.

Implementation

The rest of this chapter discusses the implementation of the Employee Files program. The first feature discussed is the user interface and how it is created. Second, the database access used in the program is discussed. Finally, you learn about any programming pitfalls that came up during the application construction.

Each sample application in this book uses a different approach to developing the user interface. This variety shows the different ways you can do your own interfaces. Hopefully, you get a cross-section of many different styles and can choose the one that suits you the best.

User Interface

The screen layout for this application is presented in a nice manner that is achieved through the use of a `GridBagLayout`. This ogre of a layout manager is difficult to work with, but when it is tamed it can provide wonderful layout capabilities.

Tip
The `JifPanel` class provides a method called `addWithConstraints()` that allows you to specify `GridBagLayout` constraints in a simple manner and add them with the component to the layout. It is used throughout many of the applications in this chapter.

Listing 13.2 gives the user interface construction code for the Employee program.

Listing 13.2. Employee files interface construction source code.

```
//****************************************************************************
//* Members                                                  &nb
sp;                        *
//****************************************************************************

    Label                          l_emp_id;
    Label                          l_fn;
    Label                          l_mn;
    Label                          l_ssn;
    Label                          l_ln;
    Label                          l_address;
    Label                          l_city;
    Label                          l_state;
    Label                          l_zc;
    Label                         l_salary;
    Label                         l_home_phone_nbr;
    Label                         l_work_ext_nbr;
    Label                         l_in_out_ind;

    JifTextField                  emp_id;
    JifTextField                  first_name;
    JifTextField                  mid_name;
    JifTextField                  ssn;
    JifTextField                  last_name;
    JifTextField                  addr_line_1;
```

```
    JifTextField                addr_line_2;
    JifTextField                city;
    JifTextField                state;
    JifTextField                zip_code;
    JifTextField               salary;
    JifTextField               home_phone_nbr;
    JifTextField               work_ext_nbr;

    JifCheckbox                   in_out_ind;

//************************************************************************
//* Constructor
;                     *
//************************************************************************

    public
    EmployeeUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );

        GridBagLayout gbl = new GridBagLayout();

        int cw[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14,
            14, 14, 14 }; // 17

        int rh[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 }; //
12

        double rc14_0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0 };

        gbl.columnWidths = new int[ 17 ];
        gbl.rowHeights = new int[ 12 ];

        gbl.columnWeights = new double[ 17 ];
        gbl.rowWeights = new double[ 12 ];

        System.arraycopy( cw, 0, gbl.columnWidths, 0, 17 );
        System.arraycopy( cw, 0, gbl.rowHeights, 0, 12 );

        System.arraycopy( rc14_0, 0, gbl.columnWeights, 0, 17 );
        System.arraycopy( rc14_0, 0, gbl.rowWeights, 0, 12 );

        setLayout( gbl );

        l_emp_id = new Label( "Employee ID:", Label.RIGHT );
        addWithConstraints( l_emp_id, "anchor=east;x=0;y=0" );

        l_ssn = new Label( "SSN:", Label.RIGHT );
        addWithConstraints( l_ssn, "anchor=east;x=6;y=0" );

        l_fn = new Label( "First Name:", Label.RIGHT );
```

```
        addWithConstraints( l_fn, "anchor=east;x=0;y=1" );

        l_mn = new Label( "Middle Name:", Label.RIGHT );
        addWithConstraints( l_mn, "anchor=east;x=0;y=2" );

        l_ln = new Label( "Last Name:", Label.RIGHT );
        addWithConstraints( l_ln, "anchor=east;x=0;y=3" );

        l_address = new Label( "Address:", Label.RIGHT );
        addWithConstraints( l_address, "anchor=east;x=0;y=4" );

        l_city = new Label( "City:", Label.RIGHT );
        addWithConstraints( l_city, "anchor=east;x=0;y=6" );

        l_state = new Label( "State:", Label.RIGHT );
        addWithConstraints( l_state, "anchor=east;x=0;y=7" );

        l_zc = new Label( "Zip Code:", Label.RIGHT );
        addWithConstraints( l_zc, "anchor=east;x=6;y=7" );

        l_salary = new Label( "Salary:", Label.RIGHT );
        addWithConstraints( l_salary, "anchor=east;x=0;y=9" );

        l_home_phone_nbr = new Label( "Home Phone:", Label.RIGHT );
        addWithConstraints( l_home_phone_nbr, "anchor=east;x=0;y=10" );

        l_work_ext_nbr = new Label( "Work Ext:", Label.RIGHT );
        addWithConstraints( l_work_ext_nbr, "anchor=east;x=6;y=10" );

        l_in_out_ind = new Label( "In/Out:", Label.RIGHT );
        addWithConstraints( l_in_out_ind, "anchor=east;x=0;y=12" );

        emp_id = new JifTextField( "", "emp_id", true );
        addWithConstraints( emp_id, "x=1;y=0;width=5;fill=horizontal" );

        ssn = new JifTextField( "", "ssn" );
        addWithConstraints( ssn, "x=7;y=0;width=7;fill=horizontal" );

        first_name = new JifTextField( "", "first_name" );
        addWithConstraints( first_name,
"x=1;y=1;width=13;fill=horizontal" );

        mid_name = new JifTextField( "", "mid_name" );
        addWithConstraints( mid_name, "x=1;y=2;width=13;fill=horizontal"
);

        last_name = new JifTextField( "", "last_name" );
        addWithConstraints( last_name, "x=1;y=3;width=13;fill=horizontal"
);

        addr_line_1 = new JifTextField( "", "addr_line_1" );
        addWithConstraints( addr_line_1,
"x=1;y=4;width=13;fill=horizontal" );
```

```
        addr_line_2 = new JifTextField( "", "addr_line_2" );
        addWithConstraints( addr_line_2,
"x=1;y=5;width=13;fill=horizontal" );

        city = new JifTextField( "", "city" );
        addWithConstraints( city, "x=1;y=6;width=13;fill=horizontal" );

        state = new JifTextField( "", "state" );
        state.setStyle( JifTextField.UPPER );
        addWithConstraints( state, "x=1;y=7;width=5;fill=horizontal" );

        zip_code = new JifTextField( "", "zip_code" );
        zip_code.setStyle( JifTextField.NUMERIC );
        addWithConstraints( zip_code, "x=7;y=7;width=7;fill=horizontal"
);

        salary = new JifTextField( "", "salary" );
        addWithConstraints( salary, "x=1;y=9;width=5;fill=horizontal" );

        home_phone_nbr = new JifTextField( "", "home_phone_nbr" );
        addWithConstraints( home_phone_nbr,
"x=1;y=10;width=5;fill=horizontal" );

        work_ext_nbr = new JifTextField( "", "work_ext_nbr" );
        addWithConstraints( work_ext_nbr,
"x=7;y=10;width=7;fill=horizontal" );

        in_out_ind = new JifCheckbox( "in_out_ind" );
        addWithConstraints( in_out_ind,
"x=1;y=12;width=7;fill=horizontal" );

        //    Disable buttons...
        saveButton.disable();
        chooseButton.disable();
        deleteButton.disable();

        //    Add the buttons...
        addWithConstraints( newButton, "x=15;y=0;width=2;fill=horizontal"
);
        addWithConstraints( saveButton,
"x=15;y=2;width=2;fill=horizontal" );
        addWithConstraints( chooseButton,
"x=15;y=4;width=2;fill=horizontal" );

        //    Tell which are numeric...
        emp_id.setNumeric( true );
        salary.setNumeric( true );

        //    Set the focus to the first field...
        setFocus( emp_id );
    }
```

A notable item about this user interface (and other `GridBagLayout` jiflets in this book) is that the grid settings are hard-coded. A permanent 12 row by 17 column grid is used for the user interface. The components are placed within the grid only after the grid is set.

Tip
In this user interface, the `Label`s all have instance variables associated with them. Unless you plan to change the text of a label during the program life-cycle, this is completely unnecessary. It is done here for clarity only.

The Employee Pick List

Another class has been developed for this application. This is the `EmployeePickList` class. This class derives from the `PickList` class of `jif.awt` package (as discussed in Chapter 11, "User Interface Classes") and presents the user with a selection of employees. When one is chosen, the object stores the selection and waits for someone to ask who has been chosen. Let's take a look at some of the source code.

```
//************************************************************************
//* EmployeePickList
                  *
//************************************************************************

public class
EmployeePickList
extends PickList
```

As stated earlier, you extend the `PickList` class. In order to do this, you must supply an `init()` method. The following is this method:

```
//************************************************************************
//* init
                           *
//************************************************************************

    public void
    init()
    {
        int rows = retrieveEmployees();

        if ( rows > 0 && getParent() instanceof Jiflet )
            ( ( Jiflet )getParent() ).verboseLog( "Retrieved " +
                Integer.toString( rows ) + " Employees" );
    }
```

This method calls the `retrieveEmployees()` method. Also, if this pick list is used with a jiflet and verbose mode is turned on, the number of employees that is retrieved is written to the log file.

The `retrieveEmployees()` method shown in Listing 13.3 is the meat of this class. It performs an SQL `SELECT` statement from the database, parses the results, and places them in the pick list for the user to select from.

Listing 13.3. The `retrieveEmployees()` method.

```
//************************************************************************
//* retrieveEmployees
;                 *
//************************************************************************

    int
```

```
    retrieveEmployees()
    {
        String        sql;
        boolean        rv = false;
        int            rows = 0;

        sql = "select * from emp order by last_name, first_name";

        try
        {
            rv = myConnection.getStatement().execute( sql );
        }
        catch ( SQLException e )
        {
            System.out.println( "Error during retrieve: " + e.toString()
);

            //    No employees to return...
            return( 0 );
        }

        //    Is this a result set?
        if ( rv )
        {
            try
            {
                ResultSet rs =
myConnection.getStatement().getResultSet();

                //    Spin through the results and add them to the
list...
                while ( rs.next() )
                {
                    EmployeeRecord er = new EmployeeRecord( rs );

                    //    Add to list...
                    if ( er.emp_id != -1 )
                    {
                        myList.addItem( er.nice_name );

                        //    Add to row mapper...
                        rowMap.insertElementAt( er, rows );

                        //    Increment row counter...
                        rows++;
                    }
                }
            }
            catch ( SQLException e )
            {
                //    Indicate an error!
                rows = -1;
            }
```

```
        }

        //    We're done!
        return( rows );
    }
```

The interesting twist here is that each employee row is stored in another class called `EmployeeRecord`. This class has a corresponding instance variable for each column in the employee table. The class is smart and knows how to read a row out of a JDBC `ResultSet` object.

So as you walk through the results returned by the SQL statement, you create a new `EmployeeRecord`. You store these records in a `Vector` for later use.

At the end, you return the number of rows that are retrieved and added to the pick list. If there is an error, you return `-1`.

The reason you store each record is for easy access. When the user selects the employee from the list, you ask the pick list to give you a copy of the record that it already retrieved. This is done in the `getRecord()` method:

```
//*************************************************************************
//* getRecord                                                 &
nbsp;                       *
//*************************************************************************

    public EmployeeRecord
    getRecord( int where )
    {
        return( ( EmployeeRecord )rowMap.elementAt( where ) );
    }
```

The pick list returns the index of the item selected. This class uses a neat trick to keep track of what row is where in the `List`. A `Vector` called `rowMap` is created. As a row of data is retrieved from the database and placed into the pick list's `List`, it is also stored in the `Vector` object at the same index level.

Later, when you need an `EmployeeRecord` from the pick list, instead of rereading the data from the database, you retrieve the row from the `Vector`. This is done in the `getRecord()` method. You see this used quite a bit in various programs.

The `EmployeePickList` object is created and displayed in the main program when the user presses the Choose button. Listing 13.4 shows how it is done.

Listing 13.4. The `action()` and `chooseEmployee()` methods.

```
//*************************************************************************
//* action                                              &nbs
p;                          *
//*************************************************************************

    /**
    * My child panel may start up picklists. It is my responsibility to
    * handle them. That is done here.
    */
    public boolean
    action( Event event, Object arg )
    {
        if ( event.target == getUIPanel() )
        {
```

```
            switch ( ( ( Integer )arg ).intValue() )
            {
                case JifMessage.chOOSE:
                    if ( getDBRecord().didDataChange() )
                    {
                        chgDlg = new ResponseDialog( this,
                        "Data Change",
                        "The record has changed.\n" +
                        "Do you wish to save your changes?",
                        "Yes,No,Cancel" );

                        chgDlg.show();
                    }
                    else
                        chooseEmployee();
                    return( true );
            }
        }

        //    Handle picklist events...
        if ( event.target instanceof EmployeePickList )
        {
            int                    rv = ( ( Integer )arg ).intValue();
             EmployeePickList     epl = ( EmployeePickList )event.target;

            if ( rv != -1 )
            {
                //    Disable save on choose...
                getUIPanel().saveButton.disable();

                //    Display it on the screen...
                setDBRecord( ( DBRecord )epl.getRecord( rv ) );
                getUIPanel().moveToScreen();
            }

            //    Kill the dialog box...
            epl.hide();
            epl.dispose();

            //    Reset the focus...
            getUIPanel().requestFocus();

            //    We handled it...
            return( true );
        }

        //    Not handled...
        return( super.action( event, arg ) );
    }

//*************************************************************************
//* chooseEmployee                                          &n
bsp;                 *
```

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

        public void
        chooseEmployee()
        {
            startWait();

            EmployeePickList epl = new EmployeePickList( this, getConnector()
    );

            epl.center( true );
            epl.show();

            endWait();
        }
```

When the Choose button is clicked, a `JifMessage` is sent to the parent. This is received in the `action()` event handler method. At this point, you need to check whether any changes have been made to the currently displayed record. If so, you ask the user whether he wants to save them.

If there are no changes to save, the method `chooseEmployee()` is called. This method creates and displays an `EmployeePickList` object.

When the user selects a pick list item or closes the pick list window, it generates an `ACTION_EVENT` event. You capture this event and act accordingly.

If the pick list returns a `-1` value, you know that the user has canceled his selection. Otherwise, the value returned is the row number that is selected. You retrieve the `EmployeeRecord` at that row, make it the current record, and request that the user interface display it.

Finally, a little cleanup is in order. You `hide()` and `dispose()` of the pick list window and then reset the focus back to your window.

Database Access

The Employee program communicates with the database through the use of an `EmployeeRecord` object. This `DBRecord` derivation knows how to create, read, update, and delete records from the employee table. The following are the instance variables of this class:

```
    //*************************************************************************
    //* Constants                                             &
    nbsp;                    *
    //*************************************************************************

        public final static String     TABLE_NAME = "emp";

    //*************************************************************************
    //* Members                                            &nb
    sp;                      *
    //*************************************************************************

        //    A variable for each table column...
        public int                      emp_id = -1;
        public String                  first_name = "";
        public String                  mid_name = "";
```

```
public String                last_name = "";
public String                ssn = "";
public String                addr_line_1 = "";
public String                addr_line_2 = "";
public String                city = "";
public String                state = "";
public String                zip_code = "";
public int                    salary = 0;
public String                home_phone_nbr = "";
public String                work_ext_nbr = "";
public String                in_out_ind = "N";

//    A computed column...
public String                nice_name;
```

Note
The `EmployeeRecord`, `EmployeePickList`, and other database classes are reused in several other applications. They are placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

Programming Considerations

This application is quite routine. Because you use the stock `SimpleDBUI` and `SimpleDBJiflet` classes as a base, not much extra work is required.

The one interesting programming consideration that comes up during the creation of this application is the caching of data in the pick list. This class purposely stores a copy of each row retrieved simply for the convenience of the calling program. When the user selects an item from the pick list, it does not have to be re-retrieved from the database because it has been cached.

Summary

This chapter introduces you to the first sample application in the intranet application suite-the Employee Files. This program is responsible for creating, updating, and deleting rows from the employee table, which is useful for human resources employees. Also, it can be modified for the employees to update their own information.

In Chapter 14, "Human Resources: Benefits Maintenance," you design and create an Employee Benefits Maintenance application. This program allows employees to change the parameters of their company-provided benefits such as 401K contributions and even W-4 exemptions.

Chapter 14

Human Resources: Benefits Maintenance

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

Introduction

In Chapter 13, "Employee Files," you set out to create an application that can create, update, and delete the employee files. In this chapter, you design and implement an application that takes this concept one step further. It not only extends the use of the infamous employee table, but it also allows the users of your intranet to make changes to their company benefits.

This chapter follows the same format as the last chapter. This chapter covers the following topics in regard to the Benefits Maintenance application:

- Application Design-This section goes over the general design of the application, including functionality and user interface considerations.
- Database Design-This section covers the database requirements for this application. It examines the data model used to support the application design.
- Implementation-This section covers how the application and database design are implemented.
- Programming Considerations-This section recaps the implementation and summarizes any difficult programming situations that have arisen.

This four step format is used throughout all of the sample application chapters. Hopefully it provides you with valuable insight and ideas for creating your own intranet applications.

Application Design

This application, not unlike the Employee Files, is semi-modeless. The user can flow through the program and only some options are available, depending on the state of the current row. The application prompts the user to store any unsaved changes he has made. This is done before any actions such as New or Choose are processed.

Figure 14.1 is the proposed user interface for the Benefits Maintenance program.

Figure 14.1 *The benefits maintenance user interface.*

This application requires a parent row to exist in the employee table. This forces you to make the user choose an employee

to work with. After choosing that employee, this application allows the user to manipulate four benefits-related data items:

- Exemptions-These are the number of exemptions the employee has claimed on his or her IRS W-4 form.
- Married-The rate at which your income is taxed in America is partially based upon the number of exemptions you claim and your marital status. This checkbox allows users to specify whether they are married.
- Plan Participant-More and more companies are starting retirement plans these days. These 401K or profit sharing plans allow employees to invest a portion of their income. This portion is usually indicated by a percentage. Many companies even match a certain amount of the percentage invested.
- Payroll Deduction-This field allows users to enter the amount of their payroll to be deducted and placed into the retirement plan.

This program also reuses the employee pick list. This pick list is stored in the `jif.common` package so that it can be reused easily.

Database Design

This application is responsible for manipulating employee benefit rows. These rows should be stored in a single table. Because not all employees have benefits to track, this information belongs in its own table. The table used in this sample application is called the employee benefits table.

The information stored in the employee benefits table corresponds to the information that is to be edited, as described earlier. Table 14.1 shows the columns needed to store in the benefits table.

Table 14.1. The employee benefits table layout.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
`Employee ID`	emp_id	number(5)	N	None
`Number of Exemptions`	exemptions_nbr	number(2)	N	0
`Married Indicator`	married_ind	char(1)	N	'N'
`Plan Participant Indicator`	plan_part_ind	char(1)	N	'N'
`Payroll Deduction Percentage`	payroll_ded_pct	number(3)	N	0

Figure 14.2 shows the entity relationship diagram for the database as it stands in this chapter. As you get deeper into the sample applications, you see the entity relationship diagram grow to encompass all the tables.

Figure 14.2 : *The entity relationship diagram including the benefits table*

Note
Entity relationship diagrams are discussed in Chapter 15.

In addition to creating a table, you create a database synonym for the table. This allows everyone to access the table with the same name and not have to worry about the schema in which the table resides.

This table is going to be a child of the employee table, which means that no rows can exist in this table unless there is a corresponding row in the employee table. Enforcing the parent-child relationship is called *referential integrity*. It ensures the *integrity* of the *references* in the database. Referential integrity is most often achieved through the use of *foreign keys*.

A foreign key is a link from a child table back to a parent table. This link allows the database to impose restrictions on

many database actions, such as the following:

- Parent rows cannot be deleted while child rows exist.
- The parent's primary key cannot be updated if child rows exist.
- Child rows cannot be inserted without an existing parent row.
- Child rows cannot be updated without an existing parent row.

Tip
There is one exception to the first foreign key restriction. Some databases allow you to perform what is called a *delete cascade*. This means that the deletion of a parent row cascades down and deletes all child rows. This can be very useful in complex database structures where tens or hundreds of child rows exist. However, with power comes danger and responsibility. You can do some serious harm to the database if this is done unwittingly. Be careful how you use the delete cascade

Listing 14.1 shows the SQL commands to create the employee benefits table.

Listing 14.1. The employee benefits table creation SQL.

```
/*    Create the table */
create table emp_benft_t
(
    emp_id                number( 5 ) not null,
    exemptions_nbr        number( 2 ) default 0 not null,
    married_ind            char( 1 ) default 'N' not null,
    plan_part_ind        char( 1 ) default 'N' not null,
    payroll_ded_pct    number( 3 ) default 0 not null
);

/*    Create a primary key */
alter table emp_benft_t
    add
    (
        primary key
        (
            emp_id
        )
    );

/*    Create a foreign key */
alter table emp_benft_t
    add
    (
        foreign key
        (
            emp_id
        )
        references emp_t
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on emp_benft_t to ia_user_r ;
```

```
/*    Drop any existing public synonym */
drop public synonym emp_benft ;

/*    Create a public synonym for our table */
create public synonym emp_benft for emp_benft_t ;
```

Note
The SQL in Listing 14.1 is quite generic, but it might not work on every database. This particular SQL has been tested with Oracle.

The first SQL clause creates the table `emp_benft_t`. The second clause creates a primary key using the `emp_id` column. Making this the primary key ensures that the values in the column are unique across all rows.

The third SQL clause creates the foreign key. The foreign key in this table is the `emp_id` column. This column points back to, or references, the `emp_id` column in the `emp_t` table.

Lastly, the public synonym `emp_benft` is created for the table `emp_benft_t`.

After you create this table, you are ready to build the application.

Caution
You must create the `emp_t` (Employee) table before you can create the `emp_benft_t` (Employee Benefits) table. Otherwise, the `emp_benft_t` SQL fails!

Implementation

The rest of this chapter discusses the implementation of the Benefits Maintenance program. The first feature discussed is the user interface and how it is created. Secondly, the database access used in the program is discussed. Finally, some of the programming considerations that come up during the application construction are discussed.

Each sample application in this book uses a different approach to developing the user interface. This variety shows you the different ways you can do your own interfaces. Hopefully, you get a cross-section of many different styles and can choose the one that suits you the best.

User Interface

The screen layout for this application is presented in a manner that is achieved through the use of a `GridBagLayout`. This is exactly the same approach that was taken in the Employee Files application. The difference here is that a `Label` variable for each label is not created.

Listing 14.2 shows the user interface construction code for the Employee Benefits program.

Listing 14.2. The Employee Benefits interface construction source code.

```
//*****************************************************************
//* Members                                          &nb
sp;                        *
//*****************************************************************

    JifTextField                emp_id;
    JifTextField                full_name;
    JifTextField               exemptions_nbr;
```

```
    JifCheckbox                   married_ind;
    JifCheckbox                   plan_part_ind;
    JifTextField                payroll_ded_pct;

//************************************************************************
//* Constructor
;                     *
//************************************************************************

    public
    BenefitsUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );

        GridBagLayout gbl = new GridBagLayout();

        int cw[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
            14, 14, 14, 14 }; // 17
        int rh[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 }; //
12

        double rc14_0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0 };

        gbl.columnWidths = new int[ 17 ];
        gbl.rowHeights = new int[ 12 ];

        gbl.columnWeights = new double[ 17 ];
        gbl.rowWeights = new double[ 12 ];

        System.arraycopy( cw, 0, gbl.columnWidths, 0, 17 );
        System.arraycopy( cw, 0, gbl.rowHeights, 0, 7 );

        System.arraycopy( rc14_0, 0, gbl.columnWeights, 0, 17 );
        System.arraycopy( rc14_0, 0, gbl.rowWeights, 0, 7 );

        setLayout( gbl );

        //    Do the labels...
        addWithConstraints( new Label( "Employee ID:", Label.RIGHT ),
            "anchor=east;x=0;y=0" );

        addWithConstraints( new Label( "Name:", Label.RIGHT ),
            "anchor=east;x=0;y=1" );

        addWithConstraints( new Label( "Exemptions:", Label.RIGHT ),
            "anchor=east;x=0;y=3" );

        addWithConstraints( new Label( "Married:", Label.RIGHT ),
            "anchor=east;x=0;y=4" );

        addWithConstraints( new Label( "Plan Participant:", Label.RIGHT
),
```

```
            "anchor=east;x=0;y=5" );

        addWithConstraints( new Label( "Payroll Deduction:", Label.RIGHT
),
            "anchor=east;x=0;y=6" );

        //    Add some fields...
        emp_id = new JifTextField( "", "emp_id", true );
        emp_id.disable();
        addWithConstraints( emp_id, "x=1;y=0;width=5;fill=horizontal" );

        full_name = new JifTextField();
        full_name.disable();
        addWithConstraints( full_name, "x=1;y=1;width=13;fill=horizontal"
);

        exemptions_nbr = new JifTextField( "", "exemptions_nbr" );
        addWithConstraints( exemptions_nbr,
"x=1;y=3;width=7;fill=horizontal" );

        married_ind = new JifCheckbox( "married_ind" );
        addWithConstraints( married_ind, "x=1;y=4" );

        plan_part_ind = new JifCheckbox( "plan_part_ind" );
        addWithConstraints( plan_part_ind, "x=1;y=5" );

        payroll_ded_pct = new JifTextField( "", "payroll_ded_pct" );
        addWithConstraints( payroll_ded_pct,
"x=1;y=6;width=7;fill=horizontal" );

        //    Disable buttons...
        saveButton.disable();
        chooseButton.disable();
        deleteButton.disable();

        //    Add the buttons...
        addWithConstraints( newButton, "x=15;y=0;width=2;fill=horizontal"
);
        addWithConstraints( saveButton,
"x=15;y=2;width=2;fill=horizontal" );
        addWithConstraints( deleteButton,
"x=15;y=4;width=2;fill=horizontal" );
        addWithConstraints( chooseButton,
"x=15;y=6;width=2;fill=horizontal" );

        //    Tell which are numeric...
        emp_id.setNumeric( true );
        exemptions_nbr.setNumeric( true );
        payroll_ded_pct.setNumeric( true );

        //    Set the focus to the first field...
        setFocus( exemptions_nbr );
    }
```

A notable item about this user interface (and other `GridBagLayout` jiflets in this book) is that the grid settings are hard-coded. A permanent 12 row by 17 column grid is used for the user interface. Components are placed within the grid only after the grid is set.

Also of note is the fact that you disable the employee ID and name `JifTextField` components:

```
emp_id.disable();
full_name.disable();
```

This forces them into a read-only mode. They accept no input and cannot be changed. In addition, their background color is shaded to indicate the disablement.

You disable these fields because they are references from the parent row. This parent row is chosen by using the Choose button and the employee pick list.

The Employee Pick List Revisited

The `EmployeePickList` class is developed for the Employee Files application in Chapter 15. However, it can be reused without modification in this application as well.

The `EmployeePickList` object is created and displayed in the main program.

When the user presses the Choose button-which is one of the `JifMessages` (`JifMessage.chOOSE`)-an `ACTION_EVENT` event is generated and sent to the parent of the panel. After it is received, you need to open up the employee pick list. Before you can switch employees though, you must store any changes the user has made to the current row. The following code snippet is from the `action()` method of the Benefits program:

```
if ( event.target == getUIPanel() )
{
    switch ( ( ( Integer )arg ).intValue() )
    {
        case JifMessage.chOOSE:
            if ( getDBRecord().didDataChange() )
            {
                chgDlg = new ResponseDialog( this,
                "Data Change",
                "The record has changed.\n" +
                "Do you wish to save your changes?",
                "Yes,No,Cancel" );

                chgDlg.show();
            }
            else
            {
                chooseEmployee();
            }
            return( true );
    }
}
```

If there are no changes to save, the method `chooseEmployee()` is called. This method creates and displays an `EmployeePickList` object.

When the user selects a pick list item or closes the pick list window, it generates an `ACTION_EVENT` event. You capture this event and act accordingly. Here's what this looks like:

```
//    Handle picklist events...
```

```
        if ( event.target instanceof EmployeePickList )
        {
            int                    rv = ( ( Integer )arg ).intValue();
             EmployeePickList     epl = ( EmployeePickList )event.target;

            if ( rv != -1 )
            {
                //    Disable save on choose...
                getUIPanel().saveButton.disable();

                //    Enable delete...
                getUIPanel().deleteButton.enable();

                //    Display it on the screen...
                EmployeeRecord er = epl.getRecord( rv );

                //    Get a benefits record...
                BenefitsRecord br = getBenefitsRow( er );

                //    Set it in there...
                setDBRecord( ( DBRecord )br );
                getUIPanel().moveToScreen();
            }

            //    Kill the dialog box...
            epl.hide();
            epl.dispose();

            //    Reset the focus...
            getUIPanel().requestFocus();

            //    We handled it...
            return( true );
        }
```

If the pick list returns a `-1` value, you know that the user has canceled his selection. Otherwise, the value returned is the row number that is selected. You retrieve the `EmployeeRecord` at that row, make it the current row, and request that the user interface display it.

Finally, a little cleanup is in order. You `hide()` and `dispose()` of the pick list window and then reset the focus back to your window.

Moving Data to the Screen

Each `SimpleDBUI` derived class has a `moveToScreen()` method that moves the data from the instance variables to the screen. The `BenefitsUI` class is no different. However, there are characteristics of this user interface that require special programming:

1. The use of indicator columns. These columns hold a yes or no value. However, you want them to be represented on the screen as a checkbox.
2. Some of the displayed information is from a second table (the employee table). You need to retrieve this information and display it.

Let's examine how you accomplish each of these programming tasks.

Indicator Columns

An indicator column is one that typically holds a Boolean value. Usually, this is represented by a `Y` or an `N` for yes or no, respectively. However, to make it easy for the user to interact with this format, a checkbox can be used.

The checkbox is a binary representation as well. It can be checked or not checked. You extend this use to say that if the checkbox is checked, it is a `Y`, or yes. If it is not checked, this is an `N`, or no.

So setting the checkbox to the right value is as simple as checking the value of the indicator. The code for checking the marriage indicator is as follows:

```
if ( er.married_ind != null )
    married_ind.setState( er.married_ind.equalsIgnoreCase( "Y" ) );
else
    married_ind.setState( false );
```

If the `er.married_ind` variable is not null, you set the state of the checkbox to checked if the married indicator is equal to `Y`. Otherwise, you set it to `N`.

You must distinguish this null value from `N` because a `NullPointerException` is thrown if it is null, and you try to compare its value with `Y`.

Displaying Parent Record Values

The second interesting programming technique used in this application is displaying information from a second database table.

As you know, your employee benefits table is a child table of the employee table. This means that no real employee information is stored in the benefits table. Because you want to display the name of the employee you're working with, you need to retrieve that employee's name from the employee table.

You can easily retrieve the employee name using the `CodeLookerUpper` class. This handy class accepts the following information in its constructor:

- A database connector
- A table name
- A key column name
- A string representing the columns to select from the database

Armed with this information, the `CodeLookerUpper` concatenates it to build an SQL `select` statement. The `select` statement ends up like this:

```
select <string columns> from <table name> where <key column> = ?
```

The bracketed items are filled in with information from the constructor. The question mark (`?`) is filled in when the SQL is actually used. The `CodeLookerUpper` contains a single method called `lookupCode()`. This is where the SQL statement is completed and executed.

Therefore, the code to initialize a `CodeLookerUpper` is as follows:

```
CodeLookerUpper clu = new CodeLookerUpper( getJiflet().getConnector(),
    "emp", "emp_id", "first_name || ' ' || last_name" );
```

You initialize it with the connector from your current jiflet, the employee table's synonym `emp`, the `emp` table's primary key `emp_id`, and the two columns you need to have concatenated at the server.

Then you ask the object to retrieve the data for you:

```
full_name.setText( clu.lookupCode( er.emp_id ) );
```

You pass the `lookupCode()` method the employee ID from your current `EmployeeRecord`. The result is moved directly into the `full_name` `JifTextField`.

Caution

> The `CodeLookerUpper` expects that the first column returned by a query is a string column. If it is not, an `SQLException` is thrown. You can return multiple columns as you have done here. However, they must be concatenated at the database level.

Database Access

The Employee Benefits program communicates with the database through the use of an `EmployeeBenefitsRecord` object. This `DBRecord` derivation knows how to create, read, update, and delete rows from the employee benefits table. The following are the instance variables of this class:

```
//****************************************************************************
//* Constants                                                &
nbsp;                    *
//****************************************************************************

    public final static String     TABLE_NAME = "emp_benft";

//****************************************************************************
//* Members                                                  &nb
sp;                    *
//****************************************************************************

    //    A variable for each table column...
    public int                     emp_id = -1;
    public int                     exemptions_nbr = 0;
    public String                 married_ind = "N";
    public String                 plan_part_ind = "N";
    public int                     payroll_ded_pct = 0;
```

> **Note**
>
> The `EmployeeBenefitsRecord` and other database classes are reused in several other applications. They are placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

Because the pick list returns an `EmployeeRecord`, you need to retrieve the `BenefitsRecord` if one exists for this employee. This is done like so:

```
//****************************************************************************
//* getBenefitsRow                                              &n
bsp;                 *
//****************************************************************************

    public BenefitsRecord
    getBenefitsRow( EmployeeRecord er )
    {
        BenefitsRecord     br = new BenefitsRecord();
        String            sql = "";

        sql = "select * from emp_benft where " +
            "emp_id = " + Integer.toString( er.emp_id );

        try
```

```
            {
                if ( getConnector().getStatement().execute( sql ) )
                {
                    ResultSet rs =
    getConnector().getStatement().getResultSet();

                    if ( rs.next() )
                        br.parseResultSet( rs );
                    else
                    {
                        br.clear();
                        br.emp_id = er.emp_id;
                    }
                }
            }
            catch ( SQLException e )
            {
                errorLog( sql + " generated: " + e.toString() );
                return( null );
            }

            //    Return the record...
            return( br );
        }
```

Given an `EmployeeRecord`, you search the table for an associated row in the employee benefits table. If it is found, you allow the `BenefitsRecord` to parse the results. Otherwise, the default values are returned.

Programming Considerations

This application builds upon your base of Employee Files and adds more functionality. It provides your intranet users with the ability to modify their benefits parameters at will. You again used the stock `SimpleDBUI` and `SimpleDBJiflet` classes as a base, allowing you to quickly put together this application.

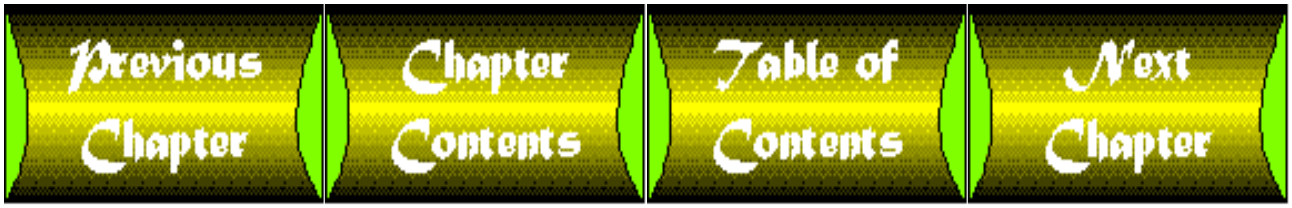
This application introduces the following Java intranet programming topics:

- Reusing classes from other applications:
- You reuse the `EmployeePickList` class created in Chapter 15.
- Database foreign keys:
- You create a foreign key from the new table referencing the employee table.
- Disabling fields to make them display only:
- You disable text fields in the user interface to make them read-only.
- Using indicator fields in the database:
- You use indicator fields in the database and change their representation on the screen to checkboxes. This allows the user to click with the mouse instead of typing a yes or no.

Summary

This chapter introduces you to the second sample application in the intranet application suite-the Benefits Maintenance application. This program is responsible for creating, updating, and deleting rows from the employee benefits table, and it is useful for normal employees who want to manage their own benefits.

In Chapter 15, "Conference Room Scheduling," you design and create an application that allows employees to schedule conference rooms for meetings well into the future.



Chapter 15

Conference Room Scheduling

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - Building the User Interface
  - Interacting with the User
  - Database Access
  - Reading the Existing Schedule
  - Storing Your Schedule
  - Generating the SQL
- Programming Considerations
- Summary

Introduction

In the last chapter, "Human Resources: Benefits Maintenance," you created a very simple intranet application that allows employees to modify some of the parameters that define their company benefits. Although a useful application, it was really only a snack. This chapter, however, is an entire four-course meal. In this chapter you will design and build an application that any company would kill for: Conference Room Scheduling.

Most companies these days have more than one conference room. It's when you need one most that they are always full; if you'd only gotten in there five minutes sooner. This application will hopefully let you and your users schedule conference room usage, alleviating any possible violent outbreaks.

This chapter will cover the following topics in regards to the Conference Room Scheduling application:

- Application Design. You'll go over the general design of the application. This includes functionality and user interface considerations.
- Database Design. This section will cover the database requirements for this application. Here you'll examine the data model used to support the application design.
- Implementation. This section will cover how the application and database design are implemented.
- Programming Considerations. In this section you'll recap the implementation and summarize any difficult programming situations that have arisen.

This four-step format will be used throughout all of the sample application chapters. Hopefully, it will provide you with valuable insight and ideas for creating your own intranet applications.

Application Design

This application needs to be visually stunning (read cool-looking) and functional at the same time. It should be simple to operate and be configurable. Remember: you want people to use it. Because scheduling a conference room is a two-step process, you need to convey that to the user visually. Therefore, I've chosen to implement this application using a tab panel.

The program has two portions. The first portion allows the user to select the date and to schedule the conference room for that date. The second portion allows the user to select the start and end times for the scheduling. To present this information in a fashion that conveys the idea that the first step must be completed before the second step can be done, we will use the `JifTabPanel` class.

This serves a couple of useful purposes. The first being that the user cannot mistakenly select incorrect information regarding the date and the room while scheduling the time. Secondly, the first step is on pane one, and the second on pane two. This means the order of completion is visually represented.

This application is not like the last two applications we designed. It is a very modal program. The last maintenance programs were deemed semi-modeless because they did not force the user into an operation mode. This application is more modal because the user must select a date and a conference room before scheduling the time.

Figure 15.1 is the proposed user interface for the Conference Room Scheduling program.

Figure 15.1 : *The Conference Room Scheduling user interface.*

The first pane is divided into two halves. The first half represents the date with a calendar. The second half lists the conference rooms that are available.

Because the first figure doesn't show the second pane, Figure 15.2 shows the second pane of your application, which you get to by clicking on the "Schedule" tab.

Figure 15.2 : *The dark side of conference room scheduling.*

The second pane presents the user with two lists of times. The left side is the start time of the scheduling and the right side is the end time of the scheduling. Along the bottom is a space for comments. These can include anything the user wants to include, usually their name.

Only after all the necessary information has been selected (date, room, start and end times), and a comment has been entered, the Save button will become enabled. The user may then save this schedule. The application will inform the user of any conflicts. Otherwise, the records are saved, and the list boxes are reloaded to show their true state. Upon successful saving, the message "Record saved…" is shown in the status bar of the application.

The application will not allow the user to select time slots that are taken. In addition, attempts to select an end time slot that is before the start time slot will be thwarted.

Figure 15.3 shows what the schedule looks like after the user has saved his choices.

Figure 15.3 : *The saved schedule is redrawn*.

This application requires information from two tables. The first table being the conference room table, the second is the schedule table. This presents you with a parent-child relationship similar to the one presented in Chapter 14, "Human Resources: Benefits Maintenance."

Database Design

This application will be responsible for manipulating schedule records. These records simply represent a time slot range on a certain day for a particular room. This simplistic design gives you a ton of flexibility in your application design. Each stored row will schedule one room for one period of time. Plus, your SQL to retrieve and update the table is simple. The

table you're going to use in this sample application is called the room schedule table.

In addition to this child table, you want to make your application configurable. Administrative users should be able to add new conference rooms to the scheduler program as they need to. To accomplish this, you need a table to hold conference rooms as well.

The design of the schedule table is interesting because no times are actually stored in it. You use the concept of *time slots*. Not unlike network television programming, you schedule rooms for use during these slots. The usage of the slots is entirely up to the table user. However, your application will have predefined slot numbers. You are going to use 0 through 47 with each representing a one-half-hour slot during a twenty-four hour period.

Table 15.1 shows the columns you'll need to store your conference room information.

Table 15.1. The layout of the conference room table.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
Room Number	room_nbr	number(5)	No	None
Floor Number	floor_nbr	number(5)	No	None
Description	desc_text	char(80)	Yes	None

Table 15.2 shows the columns you'll need to store your schedule information.

Table 15.2. The layout of the conference room schedule table.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
Room Number	room_nbr	number(5)	No	None
Floor Number	floor_nbr	number(5)	No	None
Schedule Date	sched_date	date	No	System Date
Starting Slot	start_slot	number(5)	No	None
Ending Slot	end_slot	number(5)	No	None
Comments	comment_text	char(60)	Yes	None

Figure 15.4 shows the entity relationship diagram for your database as it stands in this chapter. As you get deeper into your sample applications, you'll see your entity relationship diagram grow to encompass all the tables.

Figure 15.4 : *The entity relationship diagram including the employee tables.*

Note
Entity relationship diagrams are discussed in Chapter 13, "Employee Files."

In addition to creating a table, you'll create a database synonym for your table. This will allow everyone to access the table with the same name, and not have to worry about the schema that the table resides in.

Listing 15.1 is the SQL commands to create the conference room table.

Listing 15.1. The SQL used to create the conference room table.

```
/*    Create the table */
create table conf_room_t
(
    room_nbr            number( 5 ) not null,
    floor_nbr            number( 5 ) not null,
    desc_text            char( 80 )
);
```

```
/*    Create a primary key */
alter table conf_room_t
    add
    (
        primary key
        (
            room_nbr,
            floor_nbr
        )
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on conf_room_t to ia_user_r ;

/*    Drop any existing public synonym */
drop public synonym conf_room ;

/*    Create a public synonym for our table */
create public synonym conf_room for conf_room_t ;
```

This SQL is extremely similar to earlier table creation SQL presented in the last two chapters. After the table is created, a primary key is created. Access rights are granted to your demonstration user and a public synonym is created.

Caution
You must create the `conf_room_t` (Conference Room) table before you can create the `conf_room_sched_t` (Conference Room Schedule) table. Otherwise, the `conf_room_sched_t` SQL will fail!

However, you do have a child table to go with your parent above. This table is the conference room schedule table, and called `conf_room_sched_t` in the database. The SQL is shown in Listing 15.2.

Listing 15.2. The SQL used to create the conference room schedule table.

```
/*    Create the table */
create table conf_room_sched_t
(
    room_nbr            number( 5 ) not null,
    floor_nbr            number( 5 ) not null,
    sched_date            date default sysdate not null,
    start_slot            number( 5 ) not null,
    end_slot            number( 5 ) not null,
    comment_text        char( 60 )
);

/*    Create a primary key */
alter table conf_room_sched_t
    add
    (
        primary key
        (
            room_nbr,
            floor_nbr,
            sched_date,
```

```
            start_slot
        )
    );

/*    Create a foreign key */
alter table conf_room_sched_t
    add
    (
        foreign key
        (
            room_nbr,
            floor_nbr
        )
        references conf_room_t
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on conf_room_sched_t to ia_user_r ;

/*    Drop any existing public synonym */
drop public synonym conf_room_sched ;

/*    Create a public synonym for our table */
create public synonym conf_room_sched for conf_room_sched_t ;
```

The first SQL clause creates the table `conf_room_sched_t`. The second clause creates a four column primary key using the `room_nbr`, `floor_nbr`, `sched_date`, and `start_slot` columns. Making this the primary key ensures that the values in the column are unique across all rows.

The third SQL clause creates our foreign key. The foreign key in this table is the `room_nbr` and `floor_nbr` columns. These columns point back to, or reference, the columns in the `conf_room_t` table.

Lastly, the public synonym `conf_room_sched` is created for the table `conf_room_sched_t`.

After you have created this table, you are ready to start building your application.

Note
The SQL presented in this chapter is quite generic; however, it might not work on every database. This particular SQL has been tested with Oracle.

Implementation

The rest of this chapter will discuss the implementation of the Conference Room Scheduling program. The first thing we'll discuss is the user interface and how it was created. Secondly, we'll discuss the database access used in the program. Finally, we'll go over some of the programming considerations that came up during our application construction.

Building the User Interface

The Conference Room Scheduling program is probably the most complex program in this book. It utilizes some of the cooler user interface classes discussed in Chapter 11, "User Interface Classes." It doesn't really operate like the last two applications you developed, though. This application does use the `SimpleDBJiflet`, `SimpleDBUI`, and `DBRecord` classes. However, their use is unconventional compared to the last two applications.

This application requires two separate user interfaces to cooperate. However, because they work more closely together than not, they are managed by the same class. This class, `RoomSchedulerUI`, extends the `SimpleDBUI` class as discussed in Chapter 12, "Putting Them All Together."

To construct the user interface for this application, you need to make two separate panes. Each pane will be placed on its own tab in a tab panel. I'll discuss each pane's construction separately.

Calendar and Room List Pane

This pane is the more complex of the two panes. To achieve your interface design, you need to split the screen into four parts, then split those parts. The easiest way to do this is with a `BorderLayout` and a `GridLayout`.

The `BorderLayout` allows you to place five objects in an orientation that mimics a compass. This orientation is illustrated in Figure 15.5.

Figure 15.5 : *The* BorderLayout.

The five placements you can have are North, South, East, West, and Center. Each placement will grow to the largest size necessary. For example, if you put something in the Center that is 500 pixels high, the East and West sides will stretch to 500 pixels. The North and South placements remain zero pixels high, therefore hidden, until they contain something that has a height. For this pane in the user interface, you will only use the North and Center placements.

To create your first pane, you need to create a standard `JifPanel` object. This object's layout is set to a new `BorderLayout`:

```
//    Create a date/room selector...
JifPanel jp = new JifPanel();
jp.setLayout( new BorderLayout() );
```

The next step is to place some things in this panel. To do this you will create two additional `JifPanel` objects: one for the North, and one for the Center.

The North one is your label panel. It displays the title for the widgets that sit below it. If you refer back to Figure 15.1, you'll see the panels I'm talking about. These are created and placed into the North placement of your main panel:

```
//    Make a label panel...
JifPanel jp3 = new JifPanel();
jp3.setLayout( new GridLayout( 1, 2, 10, 1 ) );

JifPanel tp = new JifPanel( JifPanel.LOWERED );
tp.setFont( new Font( "Helvetica", Font.BOLD, 12 ) );
tp.setText( "Dates", JifPanel.TEXT_RAISED, JifPanel.CENTER );
jp3.add( tp );

tp = new JifPanel( JifPanel.LOWERED );
tp.setFont( new Font( "Helvetica", Font.BOLD, 12 ) );
tp.setText( "Conference Rooms", JifPanel.TEXT_RAISED, JifPanel.CENTER );
jp3.add( tp );
jp.add( "North", jp3 );
```

What's interesting about this placement is that you use an embedded `GridLayout` inside the panel that is placed in the "North" of your main panel. This layout is set at 1 row by 2 columns, with some additional spacing. This guarantees you that your panel will be evenly halved. It allows you to place two new label panels into the layout, and they are automatically sized correctly for you.

Creating the Center panel is quite similar. However, instead of using label panels, you are going to use actual user interface components:

```
    JifPanel p2 = new JifPanel();
    p2.setLayout( new GridLayout( 1, 2, 5, 0 ) );

    p2.add( new CalendarPanel() );

    roomList = new List();
    roomList.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
    roomList.addItem( "No Database Connection" );
    p2.add( roomList );

    jp.add( "Center", p2 );
```

You use the same technique, only the spacing is a little different. The calendar is placed in first, forcing it to the left, and the `List` of rooms is placed in second. This forces the list to be on the right. Finally it is added to the Center of your main panel.

Figure 15.6 illustrates the complete layout of the date and room selection pane.

Figure 15.6 : *The date and room pane layout.*

Loading the List of Rooms

You need to populate your list with all the available conference rooms from your conference room table (`conf_room_t`). To do this, you'll employ the same technique that was used in the `EmployeePicklist` to load it with data. Below is the source code for the `loadRoomList()` method.

```
    //*************************************************************************
    //* loadRoomList                                                  &nbs
    p;                    *
    //*************************************************************************

        public void
        loadRoomList()
        {
            //    Use a reference alias...
            DBConnector c = getJiflet().getConnector();

            //    Not connected?
            if ( c == null )
                return;

            String sql = "select * from conf_room";

            //    Clear out the old rooms...
            clearScreen();

            try
            {
                if ( c.getStatement().execute( sql ) )
                {
                    ResultSet rs = c.getStatement().getResultSet();

                    int row = 0;

                    while ( rs.next() )
```

```
                {
                    ConfRoomRecord crr = new ConfRoomRecord( rs );
                    getJiflet().setDBRecord( crr );
                    moveToScreen();

                    rowMap.insertElementAt( crr, row );
                    row++;
                }
            }
        }
        catch ( SQLException e )
        {
            getJiflet().errorLog( "Error during loading: " + e.toString()
);
        }

        return;
    }
```

This issues an SQL query that returns all of the conference rooms in your conference room table. Each returned row is stored into a `ConfRoomRecord` object. The stored row is then placed into a `Vector` for later use.

You might notice that the room list is really never populated here. That is because you've placed that code in the `moveToScreen()` method of your user interface class. Only there is data moved to the screen. But before it can move data to your list, you need to tell the base class which `DBRecord` to use. This is done with the call to `setDBRecord()`.

The `moveToScreen()` code is quite simple:

```
//*************************************************************************
//* moveToScreen                                              &nbs
p;                    *
//*************************************************************************

    public void
    moveToScreen()
    {
        if ( getJiflet().getDBRecord() == null )
            return;

        ConfRoomRecord crr = ( ConfRoomRecord )getJiflet().getDBRecord();
        roomList.addItem( crr.desc_text );
    }
```

Here you simply retrieve the `ConfRoomRecord` from your `jiflet` and add the description to your room list.

Start and End Time Pane

This pane is constructed very much like the first pane. However, there are some slight differences.

First, the Center panel is filled with two `List`s instead of the one. But for this pane, and to add a little variety to your program, I am using a `FlowLayout` instead of the grid. It does basically the same thing and is easier to set up.

Secondly, the comment field is placed in the South. This is where the user types in the reason or comment regarding the schedule of the conference room.

Figure 15.7 illustrates the layout of this pane.

Figure 15.7 : *The schedule times pane.*

The entire user interface construction code for this pane is presented below:

```
//    Create a scheduler panel...
jp = new JifPanel();
jp.setLayout( new BorderLayout() );

//    Make a label panel...
jp3 = new JifPanel();
jp3.setLayout( new GridLayout( 1, 2, 10, 1 ) );

tp = new JifPanel( JifPanel.LOWERED );
tp.setFont( new Font( "Helvetica", Font.BOLD, 12 ) );
tp.setText( "Start Time", JifPanel.TEXT_RAISED, JifPanel.CENTER );
jp3.add( tp );

tp = new JifPanel( JifPanel.LOWERED );
tp.setFont( new Font( "Helvetica", Font.BOLD, 12 ) );
tp.setText( "End Time", JifPanel.TEXT_RAISED, JifPanel.CENTER );
jp3.add( tp );
jp.add( "North", jp3 );

//    Make the list panel...
JifPanel jp2 = new JifPanel();
jp2.setLayout( new FlowLayout() );

//    Create some time selectors...
startList = new List( 12, false );
endList = new List( 12, false );

startList.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
endList.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );

jp2.add( startList );
jp2.add( endList );
jp.add( "Center", jp2 );
jp.add( "South", comment_text );
```

Using a Tab Panel

After you've created your panels, you need to put them into a tab panel so the user can switch between them. Using the tab panel is easy! First you must create it:

```
//    Create a tab panel...
JifTabPanel jtp = new JifTabPanel();
```

The next step is to add each completed pane to the tab panel. This is done with the `addPane()` method. It takes two arguments: the name to display on the tab, and the object to display on the pane.

```
//    Add the panes...
jtp.addPane( "Date/Room", jp );
jtp.addPane( "Schedule", jp );
```

The rest is handled by the tab panel class. There is nothing more for you to worry about.

Making the Buttons Look Nice

The last part of the user interface are the buttons along the side of the tab panel. These are two of the buttons that are provided to you from your base class, `SimpleDBUI`. You're going to use the Save and Delete buttons.

To place these nicely, create yet another `JifPanel` (YAJP?) object. Set a grid layout of 2 rows by 1 column. However, you set the spacing wide so your buttons will spread out:

```
//    Add the buttons...
JifPanel p = new JifPanel();
p.setLayout( new GridLayout( 2, 1, 5, 20 ) );
p.add( saveButton );
p.add( deleteButton );
```

Finalizing the Interface

The last step in creating your interface is to place your sub-panels into your master panel. Remember that your base class, `SimpleDBUI`, provides you with a container panel to work with. This panel is already set up with a `BorderLayout`. You utilize the Center and East placements of this layout for placing your sub-panels:

```
//    Add the tab panel to the center of myself...
add( "Center", jtp );

//    Add the button panel…
add( "East", p );
```

As you can see from your screen shots in Figures 15.1 and 15.2, the layouts work quite well. Figure 15.8 shows this final addition to the master layout.

Figure 15.8 : *The master layout.*

Interacting with the User

After you've constructed all the necessary components, you've got this great looking user interface. But how do you command it to do your bidding? Simple! Through the use of the events generated by the components in your interface.

Event Handling

Before you can select a schedule start and end time, you need to choose a date and a room to work with. You want to trap these selections and store them in instance variables for later use. To accomplish this you first need those two instance variables:

```
java.util.Date
            selectedRoom = null;
```

When the user selects a date on your calendar, or chooses a conference room from the list, an `ACTION_EVENT` event is generated. To capture these events, all you need to do is override the `action()` method in your class. The following is that method from your `RoomSchedulerUI()` class:

```
//*************************************************************************
//* action                                                  &nbs
p;                          *
//*************************************************************************

    public boolean
    action( Event event, Object arg )
    {
        //    Get the room record from the room list
        if ( event.target == roomList )
```

```
            {
                selectedRoom =( ConfRoomRecord )rowMap.elementAt(
                    roomList.getSelectedIndex() );

                showSelection();
                return( true );
            }

            //    Get date selection...
            if ( event.target instanceof CalendarPanel )
            {
                //    User selected a date...
                selectedDate = ( java.util.Date )arg;
                showSelection();
                return( true );
            }

            return( super.action( event, arg ) );
        }
```

First you check to see if the event was generated by your room list. If so, you retrieve the cached `ConfRoomRecord` object from your room list. Secondly, you see if the event was generated by your calendar. If it is, you store the date returned by that object.

Showing the Selection

After either of these events have fired, you call the `showSelection()` method. This method is responsible for showing some status information regarding the selections that have been made so far:

```
    //*************************************************************************
    //* showSelection                                                 &nb
    sp;                    *
    //*************************************************************************

        public void
        showSelection()
        {
            String s = "";
            String coolDate = FileDate.toNormalString( selectedDate );

            if ( !coolDate.equals( "" ) && selectedRoom != null )
            {
                s = "Room " + Integer.toString( selectedRoom.room_nbr );
                s += ", Floor " + Integer.toString( selectedRoom.floor_nbr );
                s += " on " + coolDate;

                loadLists();
            }

            if ( coolDate.equals( "" ) && selectedRoom != null )
            {
                s = "Room " + Integer.toString( selectedRoom.room_nbr );
                s += ", Floor " + Integer.toString( selectedRoom.floor_nbr );
            }
```

```
            if ( !coolDate.equals( "" ) && selectedRoom == null )
            {
                s = coolDate;
            }

            if ( s.equals( "" ) )
                 s = "Select a date and a room...";

            dateStatus.setText( s );
        }
```

It is a simple method. It converts the selected date into a normal human readable form. This date is then concatenated to the conference room information that was chosen. This new `String` is then shown in the status bar of your mainframe.

Database Access

This application communicates with the database through the use of the `DBRecord` extension class `ConfRoomRecord`. This class is used solely to retrieve information about conference rooms that are available to be scheduled.

The actual scheduling database work is done with custom constructed SQL statements, and not the use of your SQL generator classes as in the last two chapters. This allows you to explore some more interesting uses of the framework and SQL.

Note
The `ConfRoomRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

Reading the Existing Schedule

Once a date and room are selected for scheduling, it is necessary to retrieve the current schedule for this pair from the database. This is done by *stepping* through the database. You know what the date and room are, you just need to check each time slot. The following source code illustrates how to *step* through the database:

```
    //*************************************************************************
    //* getSlotStatus                                                 &nb
    sp;                  *
    //*************************************************************************

        public String
        getSlotStatus( int slot, int room, int floor, java.util.Date date )
        {
            int        count = 0;

            String     sql = "select count(*) from conf_room_sched " +
                " where room_nbr = " + Integer.toString( room ) +
                " and floor_nbr = " + Integer.toString( floor ) +
                " and " + Integer.toString( slot ) + " >= start_slot " +
                " and " + Integer.toString( slot ) + " <= end_slot " +
                " and sched_date = to_date('" + FileDate.toOracleString( date
    ) +
                "')";
```

```
            try
            {
                ResultSet rs =
                    getJiflet().getConnector().getStatement().executeQuery(
sql );

                rs.next();
                count = rs.getInt( 1 );
            }
            catch ( SQLException e )
            {
                getJiflet().errorLog( "SQL Error:" + e.toString() );
                return( "Open" );
            }

            return( ( count < 1 ) ? "Open" : "Used" );
        }
```

First construct a query based on the arguments passed to you. These arguments specify the slot number, room number, floor number and date of the row to find. The SQL merely returns a count of records that match the given criteria.

You then request that your `DBConnector` execute the query. The query will always return a single row and column unless there is a database error. Therefore, you can always expect the row count to be there.

Finally, barring any errors, you return a string representing the state of the requested criteria. This is either Open or Used. The determination is based on whether the row count was less than one.

Storing Your Schedule

Storing your schedule is quite simple. However, you need to override the default saving mechanism that is built into your framework classes. To do this, you override the `saveRecord()` method in your main class:

```
    //*********************************************************************
    //* saveRecord
                          *
    //*********************************************************************

        public boolean
        saveRecord()
        {
            RoomSchedulerUI myUI = ( RoomSchedulerUI )getUIPanel();

            if ( !myUI.canSave() )
            {
                MessageBox mb = new MessageBox( this,
                    "No Way, No How!",
                    "The information to schedule a room is incomplete.\n" +
                    "You need to select a date, room, a start time and an end
time.",
                    MessageBox.EXCLAMATION );

                mb.show();
                return( false );
            }
```

```
            String sql = myUI.getInsertSQL();

            try
            {
                getConnector().getStatement().execute( sql );

                //    Reload the schedule...
                myUI.loadLists();

                //    Disable UI components...
                myUI.saveButton.disable();
                getDBRecord().setDataChange( false );

                showStatus( "Record saved..." );
            }
            catch ( SQLException e )
            {
                errorLog( "Error creating schedule row: " +
                    e.toString() );

                showStatus( "Record not saved..." );
                return( false );
            }

            return( true );
        }
```

This overridden method first checks with the user interface to make sure that the user has selected all the necessary items to actually schedule a room. If not, a message box is displayed informing the user.

Next, you request that the user interface generate some SQL for you to use to update the database. The SQL statement is then executed. If no errors occur, the schedule lists are reloaded with data, and a message is displayed to the user that the save was successful.

Generating the SQL

To generate the SQL for storing your schedule, you need to bring together all of the selections made by the user:

- The date
- The room
- The floor number of the room
- The starting time slot
- The ending time slot
- A comment

Once these have all be selected or entered, the SQL generation may take place:

```
    //*************************************************************************
    //* getInsertSQL                                              &nbs
    p;                    *
    //*************************************************************************

        public String
        getInsertSQL()
        {
```

```
            int rc = -1;

            String sql = "insert into conf_room_sched ( room_nbr, floor_nbr,
    " +
                "sched_date, start_slot, end_slot, comment_text ) values ( "
    +
                Integer.toString( selectedRoom.room_nbr ) + ", " +
                Integer.toString( selectedRoom.floor_nbr ) + ", " +
                "to_date( '" + FileDate.toOracleString( selectedDate ) + "'
    ), " +
                Integer.toString( startList.getSelectedIndex() ) + ", " +
                Integer.toString( endList.getSelectedIndex() ) + ", " +
                "'" + comment_text.getText() + "' )";

            return( sql );
        }
```

This statement simply inserts the rows of the selected data into the database.

Programming Considerations

This application presented you with quite a challenge. You needed to present an intuitive interface to the user that flowed and showed programmatic direction. This was achieved though the use of the tab panel class, `JifTabPanel`.

You also enhanced the interface using nested layout managers. You nested a `GridLayout` within a `BorderLayout` to evenly space your components. This technique is one you'll use over and over again.

To recap, this application introduced the following Java intranet programming topics:

- Using different layout managers within layout managers.

  You used a technique where you placed a `FlowLayout` and a `GridLayout` within a `BorderLayout` to build a cool interface.
- Using the `JifTabPanel.`

  You used the `JifTabPanel` object to present two different sets of data together for the user in an intuitive manner.
- Overriding row storage method.

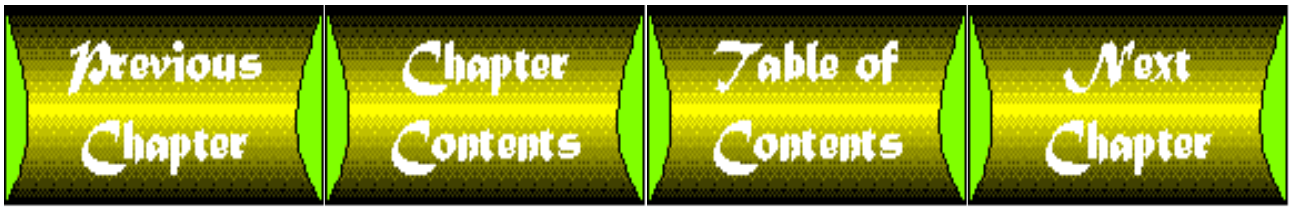
  You overrode the `saveRecord()` method of main class to do some special checking and storage for your schedules.
- Overriding SQL generation method.

  You overrode the SQL generation method in your user interface. This was done because the user interface has all the necessary information to build the SQL. You didn't use components that were capable of generating SQL on their own.

Summary

This chapter introduced you to the third sample application in your intranet application suite. This was the Conference Room Scheduling application. This program is responsible for maintaining information about the scheduling of an important company resource: conference rooms. This application should be useful to all employees on your intranet.

In Chapter 16, "Online In/Out Board," you will design and create an application that allows employees to check in and out for lunch, or even vacations.


Previous Chapter
Chapter Contents
Table of Contents
Next Chapter

Chapter 16

Online In/Out Board

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
  - A Refresh Timer
- Programming Considerations
- Summary

---

lunch \lunch\ n: *a meal eaten at midday*

Introduction

Welcome to your fourth sample intranet application. Lighter on the complexity side than the last one, this program utilizes the existing employee table to provide an Online In/Out Board.

You might not be familiar with the concept. Many companies use a chalkboard or whiteboard with each employee's name on it. Next to the name of the employee is his or her location, indicating whether he or she is in or out of the office. These boards are commonly placed where administrative assistants can easily see who is in or out. There is an online version that is representative of the same concept.

This chapter will cover the following topics in regard to the Online In/Out Board application:

- Application Design-I'll go over the general design of the application. This includes functionality and user interface considerations.
- Database Design-This section will cover the database requirements for this application. Here you'll examine the data model used to support the application design.
- Implementation-This section will cover how the application and database design are implemented.
- Programming Considerations-In this section, I'll recap the implementation and summarize any difficult programming situations that have arisen.

This four step format will be used throughout all of the sample application chapters. Hopefully, it will provide you with valuable insight and ideas for creating your own intranet applications.

Application Design

This application is one of the simpler applications in the book. Once the application has been connected to a data source, it will present the user with a list of all the employees on file. In addition, their current location, in or out, will be shown.

Figure 16.1 is the proposed user interface for the Online In/Out Board program.

Figure 16.1 : *The Online In/Out Board user interface.*

The interface will utilize a `List` component to display employees, and a single Toggle button will toggle the employee's status.

Once the list has been presented, the user may select an employee and press the Toggle button. This button toggles the current in/out indicator from yes to no, or vice versa, depending on the original value of the indicator.

The Toggle button will not become enabled until an employee has been chosen. This visually informs the user that he or she can then do something with his or her selection.

Figure 16.2 shows the application with a selection made.

Figure 16.2 : *The Online In/Out Board with the employee Karen Kenny selected*.

In addition to pressing the Toggle button, the user will be able to double-click the selection to achieve the same effect. (See Figure 16.3).

Figure 16.3 : *The Online In/Out Board after we've toggled Mr. Kenny out*.

Finally, the list needs to be refreshed periodically. A timer should be used to automatically refresh the list at a configurable period of time. The default is 60 seconds. You can override this default value by placing a `refresh.rate` property into your configuration file. 60 seconds is probably an optimal setting. If you set it any lower, you'll just be clogging your network and database with unnecessary requests. If you go higher, you may miss out on changes at lunch- time.

Database Design

This application utilizes the employee table that was defined in Chapter 13, "Employee Files." It toggles the `in_out_ind` column value from `Y` to `N`, and vice versa. For your convenience, the layout for the employee table is shown in Table 16.1.

Table 16.1 Layout for Employee Table

Description	*Column Name*	*Type*	*Null?*	*Default*
Employee ID	emp_id	number(5)	N	None
First Name	first_name	char(40)	N	None
Middle Name	mid_name	char(40)	Y	None
Last Name	last_name	char(40)	N	None
Social Security Number	ssn	char(15)	Y	None
Address Line 1	addr_line_1	char(80)	Y	None
Address Line 2	addr_line_2	char(80)	Y	None
City	city	char(80)	Y	None
State	state	char(80)	Y	None
Zip Code	zip_code	char(20)	Y	None
Salary	salary	number(7,2)	Y	None
Home Phone Number	home_phone_nbr	char(20)	Y	None
Work Extension Number	work_ext_nbr	char(20)	Y	None
In/Out Indicator	in_out_ind	char(1)	N	'N'

Implementation

In the rest of this chapter I will discuss the implementation of the Online In/Out Board program. I'll first discuss the user interface and how it was created. Secondly, I'll discuss the database access used in the program. Finally, I'll cover any programming pitfalls that came up during the application construction.

Each sample application in this book uses a different approach to developing the user interface. This variety will show you the different ways you can go about doing your own interfaces. Hopefully, you will get a nice cross-section of many different styles and choose the one that suits you the best.

User Interface

To achieve the design goal presented above, you do not need special user interface components; the stock `BorderLayout` is sufficient. You'll also employ the `List` class and a `Button`.

The following is the user interface construction code for the Employee program:

```
//************************************************************************
//* Members                                              &nb
sp;                        *
//************************************************************************

    List                          empList;

//************************************************************************
//* Constructor
;                      *
//************************************************************************

    public
    InOutBoardUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );
        setLayout( new BorderLayout() );

        empList = new List();
        empList.setFont( new Font( "Helvetica", Font.BOLD, 14 ) );
        add( "Center", empList );
        empList.enable();

        JifPanel p = new JifPanel();
        p.setLayout( new FlowLayout( FlowLayout.CENTER, 5, 5 ) );
        saveButton.setLabel("Toggle" );
        saveButton.disable();
        p.add( saveButton );
        add( "South", p );

        //    Set the focus to the first field...
        setFocus( empList );
    }
```

First, set the layout to a new `BorderLayout`. The `List` component is created and placed in the center of the layout. This is your employee list. Referring to Figure 16.1, you'll see that this list expands on all sides to fill the space.

Caution

> The default layout for the `JifPanel` class is `FlowLayout`. Because the `SimpleDBUI` class extends the `JifPanel` class, its default layout is also the `FlowLayout`. Therefore, if you want a different layout, you must create it and set it here.

Your Toggle button is next. In order to get the automatic record saving mechanism to work in your favor, I'll rename the Save button to Toggle. This button is also disabled.

Overriding the `saveRecord()` method and placing any customized row saving codes in it frees you from monitoring for special events, or even a new button's events. For example, if you want to change the Save button's name to something like Play, you could then override the `saveRecord()` method to receive notification of this being clicked. I'll cover this in the database access section later in this chapter.

Figure 16.4 illustrates the layout of this application.

Figure 16.4 : *The layout of the Online In/Out Board.*

Handling the Toggle Button

You want your Toggle button to enable, or light up, when the user has made a selection. This is easily done by looking for the correct events. In your `handleEvent()` method, use the `LIST_SELECT` and `LIST_DESELECT` events to enable and disable the button:

```
//************************************************************************
//* handleEvent                                              &nbs
p;                        *
//************************************************************************

    public boolean
    handleEvent( Event event )
    {
        //    Turn on/off buttons...
        if ( event.target instanceof List )
        {
            switch ( event.id )
            {
                case Event.LIST_SELECT:
                    getUIPanel().saveButton.enable();
                    return( true );

                case Event.LIST_DESELECT:
                    getUIPanel().saveButton.disable();
                    return( true );
            }
        }

        return( super.handleEvent( event ) );
    }
```

Your Toggle button is really the Save button in sheep's clothing: You change the text on it to say `Toggle`. Changing the text does not alter its behavior. It still generates `JifMessage.SAVE` application messages in your framework.

It still generates these messages because, instead of checking the text of the button when the initial `ACTION_EVENT` event is generated, you check the event's target with the member instance variables for all the buttons that you created in your base `SimpleDBUI` class. Checking the text is a potentially unreliable way to match commands.

> **Tip**

> Try not to rely on the text of a component to identify which component it is. This practice is somewhat unreliable and can be misleading. Also, if someone comes along later and changes your program, they could introduce a new component with duplicate text. This could potentially harm your program. Be careful!

What does this all mean to you? It means you can reliably change the text of the Save button. When this occurs, you will receive notification of the event by having your `saveRecord()` method called.

To handle the Toggling of the selected employee, you will utilize this `saveRecord()` method:

```
//************************************************************************
//* saveRecord
;                          *
//************************************************************************

    public boolean
    saveRecord()
    {
        return( toggleListItem( ( ( InOutBoardUI )getUIPanel() ).empList ) );
    }
```

Retrieve the user interface panel's `empList` variable and pass it to your Toggling function for processing. The result is returned.

Capturing Double-Clicks

In addition to the user pressing the Toggle button, you want the user to be able to double-click the mouse on an employee. This double-clicking will toggle the in/out status of that employee.

This is handled in your `action()` method. When a `List` component receives a double-click, it generates an `ACTION_EVENT` event with itself as the target. What you need to do is capture this event. Call your Toggling routine:

```
//      If list was double-clicked
if ( event.target instanceof List )
      return( toggleListItem( ( List )event.target ) );
```

The `List` is the target of the event, therefore you can simply pass it along to your Toggling method.

Database Access

This program reuses the `EmployeeRecord` object that was introduced in Chapter 13. It is a versatile class that represents a single row in the employee table. This `DBRecord` derivation knows how to create, read, update, and delete records from the employee table.

> **Note**
>
> The `EmployeeRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

There are two main database access areas to this program: the initial status retrieval and the storage of new statuses. I'll examine each one individually.

Retrieving All the Statuses

At startup, and during the lifetime of the application, the database needs to be queried, and the results displayed for the user. These results are massaged data columns taken from the database.

Retrieve the status of all employees for display:

```
    //************************************************************************
    //* loadPanel                                              &nbs
    p;                          *
    //************************************************************************

        public void
        loadPanel()
        {
            //    Make sure we're cool to go...
            if ( getConnector() == null || !getConnector().connected() )
                return;

            String sql = "select * from emp order by last_name";

            //    Clear out the old stuff...
            getUIPanel().clearScreen();

            try
            {
                if ( getConnector().getStatement().execute( sql ) )
                {
                    ResultSet rs = getConnector().getStatement().getResultSet();

                    int row = 0;

                    while ( rs.next() )
                    {
                        EmployeeRecord er = new EmployeeRecord( rs );
                        setDBRecord( er );
                        getUIPanel().moveToScreen();

                        //    Keep a map...
                        rowMap.insertElementAt( er, row );
                        row++;
                    }
                }
            } catch ( SQLException e )
            {
                errorLog( "Error during loading: " + e.toString() );
            }

            return;
        }
```

This method issues an SQL query that returns all of the employees in your employee table. Each returned row is stored into an `EmployeeRecord` object. The stored row is then placed into a `Vector` for later use.

Notice that the employee list is never populated here because you've placed that code in the `moveToScreen()` method of your user interface class. It is only there that data is moved to the screen. Before it can move data to your list, though, you need to tell the base class which `DBRecord` to use. This is done with the call to `setDBRecord()`.

The `moveToScreen()` code is simple:

```
    //************************************************************************
    //* moveToScreen                                          &
```

```
nbsp;                       *
//*****************************************************************************

    public void
    moveToScreen()
    {
        if ( getJiflet().getDBRecord() == null )
            return;

        //    Cast one off...
        EmployeeRecord er = ( EmployeeRecord )getJiflet().getDBRecord();

        String s = er.first_name.trim() + " " + er.last_name.trim() + " is ";

        if ( er.in_out_ind.equalsIgnoreCase( "Y" ) )
            s += "in";
        else
            s += "out";

        empList.addItem( s );
    }
```

Here, you retrieve the `EmployeeRecord` from your jiflet, concatenating the first and last names with a string representing their present location-in or out.

Toggling an Employee's Status

Once an employee has been selected, and the intention to toggle his or her status has been made clear, the `toggleListItem()` method comes into play:

```
//*****************************************************************************
//* toggleListItem
;                     *
//*****************************************************************************

    public boolean
    toggleListItem( List theList )
    {
        String         newInOut;
        int          idx = theList.getSelectedIndex();
        String         si = theList.getSelectedItem();

        //    Break down the selection...
        StringTokenizer st = new StringTokenizer( si, " " );

        String first_name = st.nextToken();
        String last_name = st.nextToken();
        String skip_me = st.nextToken();
        String in_out = st.nextToken();

        //    Rebuild the string...
        si = first_name.trim() + " " + last_name.trim() + " is ";

        if ( in_out.equals( "in" ) )
        {
            si += "out";
```

```
            newInOut = "N";
        }
        else
        {
            si += "in";
            newInOut = "Y";
        }

        try
        {
            //    Try and save it...
            String sql = "update emp set in_out_ind = '" + newInOut + "' " +
                "where emp_id = " +
                ( ( EmployeeRecord )rowMap.elementAt( idx ) ).emp_id;

            getConnector().getStatement().executeUpdate( sql );

            String statStr = first_name + " " + last_name + " has been marked ";

            if ( newInOut.equals( "Y" ) )
                statStr += "in";
            else
                statStr += "out";

            showStatus( statStr );
        }
        catch ( SQLException e )
        {
            errorLog( "Error during save: " + e.toString() );
            showStatus( "Error during save: " + e.toString() );
            return( false );
        }

        //    Replace the visual...
        theList.replaceItem( si, idx );

        //    Reselect...
        theList.select( idx );

        //    I handled it!
        return( true );
    }
```

The `toggleListItem()` method is unique because it rebuilds the current string and replaces it in the list. After the user selects an employee, this method strips the displayed line down to its components. It removes the "In" or "Out" from the end of the line and appends the toggled equivalent. The result is stored in the database.

Only after a successful save to the database is the string replaced in the list, showing the new status.

A Refresh Timer

The last user interface piece that you need to create is the refresh timer. This little guy is responsible for going out and refreshing the list of employees.

The refresh timer is necessary because your target users might have this running at all times. Other employees will utilize this and its underlying database. If it is never refreshed, it would show only the employee's status that the user toggled and the

initial values loaded. Using a refresh, however, you can keep your data up-to-date!

First, create an `EventTimer` to be your refresh timer. If you remember back in Chapter 8, "Utility Classes," the `EventTimer` is ideal for adding to programs where the underlying interface cannot or should not be changed. This is a case of that. You don't want to change the in-terface of your derived class. It could end up causing other problems down the road, not to mention other people may be using this class. You don't want to change things on them. The use of this `EventTimer` is perfect.

Creating your timer is a matter of using the **new** operator:

```
//    Get the refresh rate from the configuration file...
int refreshRate =
    Integer.parseInt( getParameter( "refresh.rate", "60000" ) );

//    Create a timer to refresh...
myTimer = new EventTimer( this, refreshRate );
```

You also query your configuration properties list for a `refresh.rate` property. If it isn't there, you default to a one minute default refresh rate. Otherwise the refresh rate is used.

Caution
Remember, the `refresh.rate` configuration parameter should hold values that specify the number of milliseconds between time-outs. Be careful what you place in there.

Finally, when a timer event does occur, you refresh your list:

```
//    Did my timer fire?
if ( myTimer == arg )
{
    showStatus( "Refreshing..." );

    //    Reload the panel...
    loadPanel();

    showStatus( "Refreshed!" );

    return( true );
}
```

Programming Considerations

Aside from the database access, this application is one of the simplest in your intranet application suite. It does nothing more than toggle a single column from `Y` to `N`, then back again.

This application presented you with very little challenge. You needed to present an intuitive interface to the user, while making it quick and easy to use. You did this by using the `List` class.

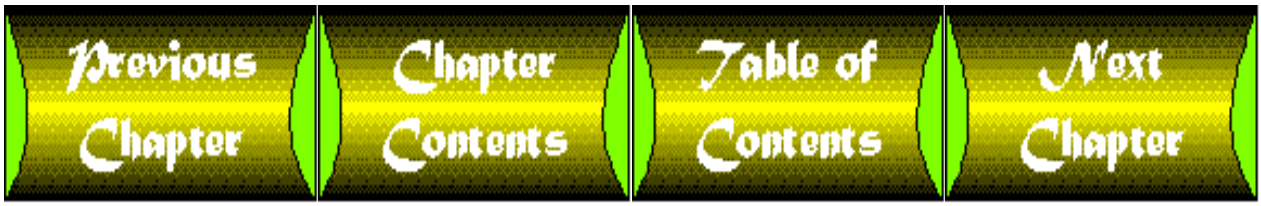
To recap, this application introduced the following Java intranet programming topics:

- Refresh timer usage:
- You used an `EventTimer` to refresh your list of employees periodically. This keeps the list fresh for people who need to know!
- Overriding row storage method:
- You overrode the `saveRecord()` method of main class to call your `toggleListItem()` method.
- No SQL generation here:
- This application is completely free of SQL generation. It is never generated for any components.

Summary

This chapter introduced you to the fourth sample application in your intranet application suite: the Online In/Out Board. This program is only responsible for updating a single column in the employee table. This program will be useful for any employee who needs to know the whereabouts of another employee. Likely uses will be for receptionists who take messages for individuals.

In Chapter 17, "Online Employee Phonebook," you will design and create an Employee Phonebook application. This program allows employees to look at the phone numbers of their coworkers.



Chapter 17

Online Employee Phonebook

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface]
  - Database Access
- Programming Considerations
- Summary

---

tel e phone \tel'e-fon'\ n: *an electronic device or system for sound reception or reproduction at a distance*

Introduction

Welcome to your fifth sample intranet application. If you thought the last program-the Online In/Out Board-was simple, wait until you see this application. This is the Online Employee Phonebook.

This read-only application displays the phone numbers of employees in a scrolling list. This is useful for all employees to look up the phone numbers of their coworkers.

This chapter will cover the following topics in regards to the Online Employee Phonebook application:

- Application Design. I'll go over the general design of the application. This includes functionality and user interface considerations.
- Database Design. This section will cover the database requirements for this application. Here you'll examine the data model used to support the application design.
- Implementation. This section will cover how the application and database design are implemented.
- Programming Considerations. In this section, I'll recap the implementation and summarize any difficult programming situations that have arisen.

This four step format will be used throughout all of the sample application chapters. Hopefully, it will provide you with valuable insight and ideas for creating your own intranet applications.

Application Design

This application, like the last, is one of the simpler applications in the book. Once this application has been connected to a data source, it will present the user with a list of all the employees on file, as well as their phone numbers and names formatted in an attractive manner.

Figure 17.1 is the proposed user interface for the Online Employee Phonebook program.

Figure 17.1 : *The Online Employee Phonebook user interface.*

The interface will utilize a `JifTextArea` component to display employees and their phone numbers, and a single Refresh button to refresh the list.

The list needs to be refreshed periodically. The Refresh button is just the ticket. By pressing that button, the user initiates the process of reloading the `JifTextArea`.

Database Design

This application utilizes the employee table that was defined in Chapter 13, "Employee Files." There really is nothing new here. The application retrieves the rows from the table and displays the data.

Implementation

In the rest of this chapter I will discuss the implementation of the Online Employee Phonebook program. I'll first discuss the user interface and how it was created. Secondly, I'll discuss the database access used in the program. Finally, I'll cover any programming pitfalls that came up during the application construction.

Each sample application in this book uses a different approach to developing the user interface. This variety will show you the different ways you can go about doing your own interfaces. Hopefully, you will get a nice cross-section of many different styles and choose the one that suits you the best.

User Interface

To achieve the design goal presented above, you do not need special user interface components. The stock `BorderLayout` is sufficient. You'll also employ the `JifTextArea` class and a button.

Tip
In order to show the data separated by tabs, you use a `TextArea` component. Using a `List` component produces unstable results. Sometimes the columns will line up, sometimes they won't. Whenever you need evenly spaced text, use a `TextArea` com-ponent.

The following is the user interface construction code for the Employee program:

```
//*************************************************************************
//* Members                                              &nb
sp;                        *
//*************************************************************************

    JifTextArea                     empList;

//*************************************************************************
//* Constructor
;                       *
//*************************************************************************

    public
    PhoneBookUI( SimpleDBJiflet jiflet )
    {
```

```
        super( jiflet );
        setLayout( new BorderLayout() );

        empList = new JifTextArea( 15, 40 );
        add( "Center", empList );

        JifPanel p = new JifPanel();
        p.setLayout( new FlowLayout( FlowLayout.CENTER, 5, 5 ) );
        saveButton.setLabel( "Refresh" );
        saveButton.disable();
        p.add( saveButton );
        add( "South", p );
    }
```

First, set the layout to a new `BorderLayout`. The `JifTextArea` component is created and placed in the center of the layout. This is your employee phone list. Referring to Figure 17.1, you'll see that this list expands on all sides to fill the space.

Your Refresh button is next. You'll use a neat trick here: In order to get an automatic notification that your button was pressed, you'll rename the Save button to Refresh. This button is also disabled.

Overriding the `saveRecord()` method and placing any customized row-saving codes in it free you from monitoring for special events, or even a new button's events. For example, if you want to change the Save button's name to something like Play, you could then override the `saveRecord()` method to receive notification of the button being clicked. I'll cover this in the database access section later in this chapter.

Figure 17.2 illustrates the layout of this application.

Figure 17.2 : *The layout of the Online Employee Phonebook.*

Handling the Refresh Button

You want your Refresh button to enable, or light up, when the database connection has been established. To achieve this, you override the default `connectToDatabase()` method:

```
//************************************************************************
//* connectToDatabase
;                *
//************************************************************************

    public void
    connectToDatabase()
    {
        //    Call my dad...
        super.connectToDatabase();

        //    Get all the news...
        if ( getConnector().connected() )
        {
            //    Enable the refresh button...
            getUIPanel().saveButton.enable();

            //    Load the panel up...
            loadPanel();
        }
    }
```

In the overridden method you call the base class' version. If it is successful, you enable the Refresh button, and load up the employee phone list.

Your Refresh button is really the Save button in sheep's clothing: You change the text on it to say Refresh. Changing the text does not alter its behavior. It still generates `JifMessage.SAVE` application messages in your framework.

It still generates these messages because instead of checking the text of the button when the initial `ACTION_EVENT` event is generated, you check the event's target with the member instance variables for all the buttons that you created in your base `SimpleDBUI` class.

What does this all mean to you? It means you can reliably change the text of the Save button. When this occurs, you will receive notification of the event by having your `saveRecord()` method called.

To handle the refreshing of the employee list, you will utilize the `saveRecord()` method:

```
//************************************************************************
//* saveRecord                                                           *
//************************************************************************

    public boolean
    saveRecord()
    {
        loadPanel();
        return( true );
    }
```

The `loadPanel()` method is called to load up the employee phone list.

Database Access

This program reuses the `EmployeeRecord` object that was introduced in Chapter 13. It is a versatile class that represents a single row in the employee table. This `DBRecord` derivation knows how to create, read, update, and delete records from the employee table.

Note
The `EmployeeRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

There is only a single database access area to this program: the employee phone list retrieval.

Retrieving the Phone Numbers

At startup, and during the lifetime of the application, the database needs to be queried, and the results displayed for the user. These results are massaged data columns taken from the database.

Retrieve all of the employees for display:

```
//************************************************************************
//* loadPanel                                                            &
nbsp;                    *
//************************************************************************

    public void
```

```
        loadPanel()
        {
            String sql = "select * from emp";

            //    Clear out the old stuff...
            getUIPanel().clearScreen();

            try
            {
                if ( getConnector().getStatement().execute( sql ) )
                {
                    ResultSet rs =
    getConnector().getStatement().getResultSet();

                    int row = 0;

                    while ( rs.next() )
                    {
                        EmployeeRecord er = new EmployeeRecord( rs );
                        setDBRecord( er );
                        getUIPanel().moveToScreen();
                    }
                }
            } catch ( SQLException e )
            {
                errorLog( "Error during loading: " + e.toString() );
            }

            return;
        }
```

This retrieval issues an SQL query that returns all of the employees in your employee table. Each returned row is stored into an `EmployeeRecord` object. The stored row is then placed into a `Vector` for later use.

Notice that the employee phone list is never populated here because you've placed that code in the `moveToScreen()` method of your user interface class. It is only there that data is moved to the screen. Before it can move data to your list, though, you need to tell the base class which `DBRecord` to use. This is done with the call to `setDBRecord()`.

The `moveToScreen()` code is quite simple:

```
    //*************************************************************************
    //* moveToScreen                                              &nbs
    p;                    *
    //*************************************************************************

        public void
        moveToScreen()
        {
            if ( getJiflet().getDBRecord() == null )
                return;

            //    Cast one off...
            EmployeeRecord er = ( EmployeeRecord )getJiflet().getDBRecord();

            String s = er.first_name + " " + er.last_name + "\t\t";
```

```
            if ( er.work_ext_nbr == null )
                s += "(None)";
            else
                s += er.work_ext_nbr;

            s +="\t";

            if ( er.home_phone_nbr == null )
                s += "(None)";
            else
                s += er.home_phone_nbr;

            s += System.getProperty( "line.separator" );

            empList.appendText( s );
        }
```

Here, you retrieve the `EmployeeRecord` from your `jiflet`, concatenating the first and last names with a string holding the employees' work extensions and their home phone numbers.

Programming Considerations

Aside from the database access, this application is another simple one in your intranet application suite. It does nothing more than list the rows in the employee table in a visually pleasing format.

This application presented you with very little challenge. You needed to present an intuitive interface to the user, while making it effortless to use. You did this by using the `JifTextArea` class.

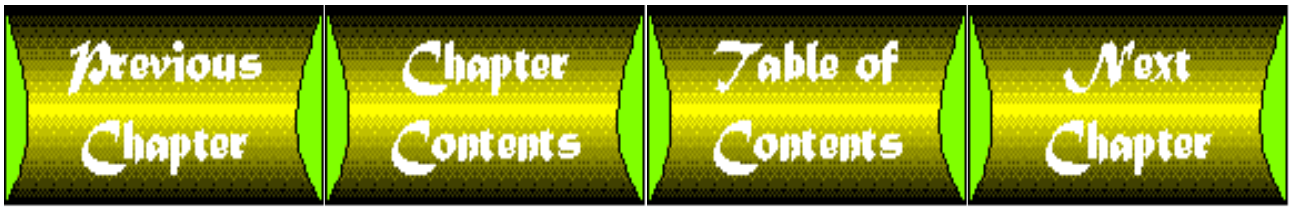
To recap, this application introduced the following Java intranet programming topics:

- Refresh button usage:
- You used a button to allow the user to manually refresh the list of phone numbers.
- Overriding row storage method:
- You overrode the `saveRecord()` method of the main class to call your refresh method, `loadPanel()`.
- No SQL generation here!:
- This application is completely free of SQL generation. It is never generated for any components.

Summary

This chapter introduced you to the fifth sample application in your intranet application suite: the Online Employee Phonebook. This program is not responsible for updating any data; it is completely read-only. This program will be useful for any employee who needs to know the phone numbers of another employee.

In Chapter 18, "News and Announcements," you will design and create an application that will display important company news and announcements. This program will allow employees to know what is going on at all times.


Previous Chapter
Chapter Contents
Table of Contents
Next Chapter

Chapter 18

News & Announcements

by *Jerry Ablan*

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - User Interface
  - Database Access
- Programming Considerations
- Summary

---

an nounce \a-nouns'\ v: *to make known publicly*

Introduction

Welcome to your sixth sample intranet application. This application, like the last two, is very simple in design; however, it will provide intranet users with plenty of information to talk about at the watercooler. This is the News & Announcements application.

This read-only application displays listings of company news and announcements that are stored in the database. It is useful for disseminating all types of company information like bonuses, casual days, office parties, and so on.

This chapter will cover the following topics in regard to the News & Announcements application:

- Application Design-I'll go over the general design of the application, including functionality and user interface considerations.
- Database Design-This section will cover the database requirements for this application. Here you'll examine the data model used to support the application design.
- Implementation-This section will cover how the application and database design are implemented.
- Programming Considerations-In this section, I'll recap the implementation and summarize any difficult programming situations that have arisen.

NOTE
This four step format will be used throughout all of the sample application chapters. Hopefully, it will provide you with valuable insight and ideas for creating your own intranet applications

Application Design

This application, like the one in the last chapter, is one of the simpler applications in the book. Once this application has been connected to a data source, it will present the user with a list of all the news on file.

Figure 18.1 is the proposed user interface for the News & Announcements program.

Figure 18.1 : *The News & Announcements user interface.*

The interface will utilize a `JifTextArea` component to display news. This brings up an interesting point: Where is the news entered anyway?

Well, right here in this program. You need this program to be dual-functioning. This means that it is read-only for some people, but lets others create new news and announcements. Figure 18.2 illustrates how this should look.

Figure 18.2 : *The News & Announcements user interface with saving capabilities.*

Database Design

This application is responsible for manipulating news rows. These rows should be stored in a single table. The table you're going to use in this sample application is called the News table.

The information stored in the news table corresponds to the information that is to be edited as described above. Table 18.1 shows the columns you'll need to store in your news table.

Table 18.1. The layout of the News table

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
News ID	news_id	number(5)	No	None
Date/Time of News Item	news_datetime	date	No	System Date
News Description	desc_text	char(255)	No	None
Originator of News Item	emp_id	number(5)	Yes	None

Figure 18.3 shows the entity relationship diagram for your database as it stands in this chapter. As you get deeper into your sample applications, you'll see your entity relationship diagram grow to encompass all the tables.

Figure 18.3 : *The entity relationship diagram including our new News table.*

NOTE
Entity relationship diagrams are discussed in Chapter 13, "Employee Files.

In addition to creating a table, you'll create a database synonym for your table. This will allow everyone to access the table with the same name without having to worry about the schema in which the table resides.

While this table has a foreign key back to the employee table, it is not a dependent table. The relationship is merely informational. That is because the `emp_id` column in your table may be `null`. This allows you to ignore it completely. In fact, in the implementation of the News & Announcement program, it is not used.

TIP
Chapter 14, "Human Resources: Benefits Maintenance," provides some in-depth coverage of foreign keys and referential integrity.

Listing 18.1 shows the SQL commands to create the News table.

Listing 18.1. The SQL to create the News table.

```
/*    Create the table */
create table news_t
(
    news_id                number( 5 ) not null,
    news_datetime        date default sysdate not null,
    desc_text            char( 255 ) not null,
    emp_id                number( 5 ) default null
);

/*    Create a primary key */
alter table news_t
    add
    (
        primary key
        (
            news_id
        )
    );

/*    Create a foreign key */
alter table news_t
    add
    (
        foreign key
        (
            emp_id
        )
        references emp_t
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on news_t to ia_user_r ;

/*    Drop any existing public synonym */
drop public synonym news ;

/*    Create a public synonym for our table */
create public synonym news for news_t ;
```

NOTE
The preceding SQL is quite generic; however, it may not work on every database. This particular SQL has been tested with Oracle.

The first SQL clause creates the table `news_t`. The second clause creates a primary key using the `news_id` column. Making this the primary key ensures that the values in the column are unique across all rows.

The third SQL clause creates your foreign key. The foreign key in this table is the `emp_id` column. This column points back to, or references, the `emp_id` column in the `emp_t` table.

Lastly, the public synonym `news` is created for the table `news_t`.

After you have created this table, you are ready to start building your application.

CAUTION
You must create the `emp_t` (Employee) table before you can create the `news_t` (News) table. Otherwise, the `news_t` SQL will fail!

Implementation

The rest of this chapter will discuss the implementation of the News & Announcements program. First, I'll discuss the user interface and how it was created. Secondly, I'll discuss the database access used in the program. Finally, I'll cover any programming pitfalls that came up during the application construction.

Each sample application in this book uses a different approach to developing the user interface. This variety will show you the different ways you can go about doing your own interfaces. Hopefully, you will get a nice cross-section of many different styles and choose the one that suits you best.

User Interface

To achieve the design goal presented above, you do not need special user interface components. For variety, I chose to use the `GridBagLayout` layout manager for this application. It uses the same hard-coded row and column heights as the Employee Files program. This provides another example of how to use this difficult layout manager.

The following is the user interface construction code for the News & Announcements program:

```
//************************************************************************
//* Members                                                              *
//************************************************************************

    JifTextField               news_id = new JifTextField( "" );
    JifTextField               news_datetime;
    JifTextArea                   news_text;
    JifTextField               new_news_text;

//************************************************************************
//* Constructor                                                          *
//************************************************************************

    public
    NewsUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );

        GridBagLayout gbl = new GridBagLayout();

        int cw[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
            14, 14, 14 }; // 17

        int rh[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 }; // 12

        double rc14_0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

        gbl.columnWidths = new int[ 17 ];
        gbl.rowHeights = new int[ 12 ];
```

```
        gbl.columnWeights = new double[ 17 ];
        gbl.rowWeights = new double[ 12 ];

        System.arraycopy( cw, 0, gbl.columnWidths, 0, 17 );
        System.arraycopy( cw, 0, gbl.rowHeights, 0, 12 );

        System.arraycopy( rc14_0, 0, gbl.columnWeights, 0, 17 );
        System.arraycopy( rc14_0, 0, gbl.rowWeights, 0, 12 );

        setLayout( gbl );

        news_text = new JifTextArea( "" );
        addWithConstraints( news_text, "x=0;y=0;width=17;height=10" );
        news_text.disable();

        new_news_text = new JifTextField( "" );

        //    Don't add if read only...
        if ( jiflet.getParameter( "readOnly",
            "true" ).equalsIgnoreCase( "false" ) )
            addWithConstraints( new_news_text,
                "x=0;y=11;width=16;fill=horizontal" );

        //    Add the buttons...
        saveButton.disable();

        //    Don't add if read only...
        if ( jiflet.getParameter( "readOnly",
            "true" ).equalsIgnoreCase( "false" ) )
            addWithConstraints( saveButton, "x=16;y=11" );

        //    Tell which are numeric...
        news_id.setNumeric( true );

        //    Set the focus to the first field...
        setFocus( news_text );
    }
```

Set the layout to a new `GridBagLayout`. The `JifTextArea` component is created and placed in the layout. This is your news list. It is disabled so the user cannot type or edit the information.

Conditional Construction

Here's where you come to an interesting part. Query your configuration parameters with the `getParameter()` method to see if there is a property called `readOnly`. If this property exists, and is set to `false`, then you must allow the user to create new news items. You do this by adding a `JifTextField` and a `saveButton` to the layout if this `readOnly` property is set to `false`.

Database Access

This application communicates with the database through the use of a `NewsRecord` object. This `DBRecord` derivation knows how to create, read, update, and delete records from the news table. The following are the instance variables of this class:

```
//***************************************************************************
//* Constants                                                               *
//***************************************************************************

    public final static String     TABLE_NAME = "news";

//***************************************************************************
//* Members                                                                 *
//***************************************************************************

    //    A variable for each table column...
    public int                     news_id = -1;
    public Date                     news_datetime = null;
    public String                  desc_text = "";
```

NOTE
The `NewsRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

Retrieving the News

At startup of the application, the database needs to be queried, and the results displayed for the user. These results are data columns taken from the database.

Retrieve all of the news for display:

```
//***************************************************************************
//* loadPanel                                                               *
//***************************************************************************

    public void
    loadPanel()
    {
        String sql = "select * from emp";

        //    Clear out the old stuff...
        getUIPanel().clearScreen();

        try
        {
            if ( getConnector().getStatement().execute( sql ) )
            {
                ResultSet rs = getConnector().getStatement().getResultSet();

                int row = 0;

                while ( rs.next() )
                {
                    EmployeeRecord er = new EmployeeRecord( rs );
```

```
                    setDBRecord( er );
                    getUIPanel().moveToScreen();
                }
            }
        } catch ( SQLException e )
        {
            errorLog( "Error during loading: " + e.toString() );
        }

        return;
    }
```

This issues an SQL query that returns all of the news in your news table. Each returned row is stored in a `NewsRecord` object. The stored row is then placed into a `Vector` for later use.

You may notice that the news list is not populated here. That is because you've placed that code in the `moveToScreen()` method of your user interface class. However, there is data moved to the screen. But, before it can move data to your list, you need to tell the base class which `DBRecord` to use. This is done with the call to `setDBRecord()`.

The `moveToScreen()` code is quite simple:

```
//************************************************************************
//* moveToScreen                                                         *
//************************************************************************

    public void
    moveToScreen()
    {
        if ( getJiflet().getDBRecord() == null )
            return;

        //    Cast one off...
        NewsRecord er = ( NewsRecord )getJiflet().getDBRecord();

        String nr = er.news_datetime.toString() +
            ". " + er.desc_text +
            System.getProperty( "line.separator" );

        news_text.appendText( nr );
    }
```

Here, you retrieve the `NewsRecord` from your `jiflet`, concatenating the date and time with the news description. Finally, tack on a line feed and append it to the text area.

Storing the News

If the user has access to the news input area then any change in that component will signal the actual Save button to enable. This is handled by your framework.

To actually save the news, you need to generate a little SQL. You do this in your main class. Here is the code:

```
//************************************************************************
//* saveRecord                                                           *
//************************************************************************
```

```
    /**
    * This overridden saveRecord() knows how to save the news...
    */
    public boolean
    saveRecord()
    {
        boolean        success = true;

        //    Read only?
        if ( getParameter( "readOnly", "true" ).equalsIgnoreCase( "true" ) )
        {
            MessageBox mb = new MessageBox( this, "Sorry Charlie!",
                "You are not allowed to save data!",
                MessageBox.STOP );
            mb.show();
            return( true );
        }

        try
        {
            String            sql;
            int                new_uid;

            //    Get a new UID!
            SequenceGenerator sg = new SequenceGenerator(
                getConnector(), "news", "news_id" );

            new_uid = sg.getNextValue();

            String newNews = ( ( NewsUI )getUIPanel() ).new_news_text.getText();

            sql = "insert into news ( news_id, news_datetime, desc_text ) " +
                "values ( " + Integer.toString( new_uid ) +
                ", sysdate, Ô" + newNews + "Ô )";

            if ( !sql.equals( "" ) )
            {
                getConnector().getStatement().execute( sql );
                showStatus( "News Saved..." );
                loadPanel();
            }
        }
        catch ( SQLException e )
        {
            showStatus( "News Not Saved..." );
            getConnector().errorLog( e.toString() );
            success = false;
        }

        return( success );
    }
```

First ensure that the user is allowed to create news rows. If he is, then create a new `news_id` using the `SequenceGenerator` class. This is then used to construct an SQL `INSERT` statement. When executed, the new news row

is inserted into the table.

If it is successful, the news items are reread from the database and displayed.

Programming Considerations

Aside from the database access, this application is quite simple. It does little more than list the rows in the news table in an attractive manner. If you set a certain property a certain way, creating the news becomes an option for you!

You needed to present an intuitive interface to the user, while making it effortless to use. You did this by using the `JifTextArea` class. You also needed a way to create news. This was handled by hidden fields that are placed into the layout based on configuration parameters. This allows the program to have a dual-functionality.

To recap, this application introduced the following Java intranet programming topics:

- Conditional Interface Construction:
  You only added the Save button and new news text field to the layout if the `readOnly` property was set to `false`.
- Overriding row storage method:
  You overrode the `saveRecord()` method of main class to check security, and to generate your own SQL for INSERTing rows into the news table.

Summary

This chapter introduced you to the sixth sample application in your intranet application suite: the News & Announcements. This program is responsible for updating the News table and displaying its contents. It is completely read-only to most users, while others may be granted access to save new news. This program will be useful for all of the employees. It allows them to keep tabs on what is going on at the company.

In Chapter 19, "Product Maintenance," you will design and create an application that will allow to you create, update, and delete records of the things your company handles: products.

Chapter 19

Product Maintenance

CONTENTS

- Introduction
- Who Would Use This Application?
  - Johnston, Ulysses, Norman, and Kaiser
- Application Design
  - Using a Pick List
- Database Design
- Implementation
  - User Interface
  - The Product Pick List
  - Database Access
- Programming Considerations
- Summary

---

pro duct \prod'ukt\ n: *something produced by natural, human, or mechanical effort*

Introduction

Welcome to the penultimate sample intranet application. This, the Product Maintenance application, isn't as simple as the last few have been. It is more in line with your first application, Employee Files, and should give you a good understanding of reading and writing database rows.

Most companies in business for profit have something to sell. Whether it is services or a tangible item, it is a product. This application enables you to manage information about these products. When you have your products under control, you can track certain aspects. One such aspect is customer support, which is explored in your final application-Customer Support Maintenance.

This application enables the user to create, update, and delete rows in the product table. This new table holds rows that describe products that you or your company sell, manufacture, distribute, and so on. The product is what you output.

This chapter covers the following topics in regard to the Product Maintenance application:

- Application Design-The general design of the application is covered in detail, including functionality and user interface considerations.
- Database Design-This section covers the database requirements for this application. Here the data model used to support the application design is examined.
- Implementation-This section covers how the application and database design are implemented.

- Programming Considerations-This section recaps the implementation and summarizes any difficult programming situations that have arisen.

This four-step format just outlined is used throughout all of the sample application chapters, and it should provide you with valuable insight and ideas for creating your own intranet applications.

Who Would Use This Application?

This application is useful for companies that sell products. To better clarify this, let's take a look at one such fictitious company: Johnston, Ulysses, Norman, and Kaiser.

Johnston, Ulysses, Norman, and Kaiser

Johnston, Ulysses, Norman, and Kaiser (JUNK) is a fictitious corporation that will serve as a model for the product and customer support sample applications in this book. They are an average-sized company and sell a variety of odd and obscure products throughout the world. These sales are through large chain-stores, smaller mom-and-pop stores, and mail-order catalogs. Their products are bought from small manufacturers and distributors around the United States.

The catalogs are small, presenting hundreds of items to the consumer, and they are one of JUNK's biggest sales tools. Generally, all the items in the catalog are under $5.00. JUNK makes its money on the shipping and handling charges, and the fact that consumers often buy 10 items from the catalog because items are so inexpensive.

By now you're wondering what kind of products JUNK sells. I was going to have them sell widgets. However, in the last 10 years or so, widgets, in relation to software construction, have taken on a new connotation. Widgets now refer to software components more than a generic product line.

So JUNK sells the types of products that you don't go out and buy. That is to say, 99 percent of its product line is impulse-buy material. JUNK sells the kind of stuff you see at check-out counters in stores, such as red rubber snakes or cartoon character figurines that you stick on the end of your finger. My favorite JUNK product is the electric lollipop. This is a large sucker on a motorized stick. Press the button and it spins the lollipop. All you need to do is stick out your tongue!

This is the type of company that can use the product maintenance and customer support applications on its intranet. This company constantly gets calls from customers with problems. These problems are answered by customer support people on a daily basis. However, they currently have no central repository, or knowledge base, that holds all of the answers. By using the product maintenance and customer support applications, they will be able to better use their combined knowledge for support calls.

Application Design

This application is quite similar to the Employee Files application, and it has many of the same qualities. At the core, it's a database row manipulation application. With it you can create, update, and delete rows in the product table.

The proposed interface is similar to the Employee Files as well. Figure 19.1 is the proposed user interface for the Product Maintenance program.

Figure 19.1 : *The Product maintenance user interface.*

This application is semi-modeless, which means that it has no operating mode. You shouldn't have to inform the program that you're going to be adding new records or removing records. You can just flow through the program and it determines what should be done.

The basic function of this application is to create, read, update, and delete product records. These records are stored in a database. The database design is discussed in detail later in this chapter.

Think of this application as a pointer into the product table. The record that the pointer is situated upon is the current record. This current record is displayed to the user, and the user can do with it what he or she will. The user can also insert

records into the table.

The user needs a method of moving this pointer from product to product. The best way to present this information to the user is through the use of a pick list.

Using a Pick List

The pick list, as previously discussed, is a selection of all the records in the product table. This selection should include the name of each product. The user should be allowed to select one name from the displayed list. After the selection is made, the chosen record should be fully retrieved and displayed.

The selection of a record can be accomplished by either a double-click on the list item, or a single-click followed by the user pressing the OK button.

Figure 19.2 represents the concept of the product pick list.

Figure 19.2 : *The product pick list.*

This pick list should be opened in response to the user pressing the Choose button, a standard `SimpleDBUI` button.

Database Design

This application is responsible for manipulating news rows. These rows should be stored in a single table. The table used in this sample application is called the News table.

The information stored in the News table corresponds to the information that is to be edited as described earlier. Table 19.1 shows the columns that need to be stored in the product table.

Table 19.1. The product table layout.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
Product ID	prod_id	number(5)	No	None
Description	desc_text	char(80)	Yes	None
Quantity On Hand	qty_on_hand	number(10)	No	None
Quantity On Order	qty_on_order	number(10)	No	None
Last Received Date	last_rcv_date	date	Yes	Null
Comment	comment_text	char(80)	Yes	None

Figure 19.3 shows the entity relationship diagram for the database as it stands in this chapter. In Chapter 20, "Customer Support Maintenance," you see the final entity relationship diagram. It will represent all of the tables that have developed for the applications in this book.

Figure 19.3 : *The entity relationship diagram, including the new product table.*

Note
Entity relationship diagrams are discussed in Chapter 13, "Employee Files."

In addition to creating a table, a database synonym for the table is created as well. This enables everyone to access the table with the same name, without having to worry about the schema in which the table resides.

Listing 19.1 is the list of SQL commands used to create the product table and synonym.

Listing 19.1. The product table creation SQL.

```
/*    Create the table */
create table prod_t
(
    prod_id                 number( 5 ) not null,
    desc_text              char( 80 ),
    qty_on_hand             number( 10 ) not null,
    qty_on_order           number( 10 ) not null,
    last_rcv_date           date default null,
    comment_text           char( 80 )
);

/*    Create a primary key */
alter table prod_t
    add
    (
        primary key
        (
            prod_id
        )
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on prod_t to ia_user_r ;

/*    Drop any existing public synonym */
drop public synonym prod ;

/*    Create a public synonym for our table */
create public synonym prod for prod_t ;
```

Note
The preceding SQL is quite generic, but it still might not work on every database. This particular SQL has been tested with Oracle.

The first SQL clause creates the table `prod_t`. The second clause creates a primary key using the `prod_id` column. Making this the primary key ensures that the values in the column are unique across all rows. Lastly, the public synonym `prod` is created for the table `prod_t`.

Implementation

The rest of this chapter discusses the implementation of the Product Maintenance program. The first item discussed is the user interface and how it was created. Secondly, the database access used in the program is covered. Finally, any programming pitfalls that came up during the application construction are examined.

User Interface

To achieve the design goal presented earlier, you need no special user-interface components. The `GridBagLayout` layout manager is used for this application. It uses the same hard-coded row and column heights as the Employee Files program.

Listing 19.2 shows the user-interface construction code for the Product Maintenance program.

Listing 19.2. The Product Maintenance interface construction source code.

```
//*************************************************************************
//* Members                                              &nb
sp;                        *
//*************************************************************************

    JifTextField                prod_id;
    JifTextField                desc_text;
    JifTextField                qty_on_hand;
    JifTextField                qty_on_order;
    JifTextField                last_rcv_date;
    JifTextField                comment_text;

//*************************************************************************
//* Constructor
;                       *
//*************************************************************************

    public
    ProductUI( SimpleDBJiflet jiflet )
    {
        super( jiflet );

        GridBagLayout gbl = new GridBagLayout();

        int cw[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14,
            14, 14, 14 }; // 17

        int rh[] = { 14, 14, 14, 14, 14, 14, 14, 14, 14 }; // 9

        double rc14_0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0 };

        gbl.columnWidths = new int[ 17 ];
        gbl.rowHeights = new int[ 9 ];

        gbl.columnWeights = new double[ 17 ];
        gbl.rowWeights = new double[ 9 ];

        System.arraycopy( cw, 0, gbl.columnWidths, 0, 17 );
        System.arraycopy( cw, 0, gbl.rowHeights, 0, 9 );

        System.arraycopy( rc14_0, 0, gbl.columnWeights, 0, 17 );
        System.arraycopy( rc14_0, 0, gbl.rowWeights, 0, 9 );

        setLayout( gbl );

        addWithConstraints( new Label( "Product ID:", Label.RIGHT ),
            "anchor=east;x=0;y=0" );

        addWithConstraints( new Label( "Description:", Label.RIGHT ),
            "anchor=east;x=0;y=1" );
```

```
        addWithConstraints( new Label( "Quantity On Hand:", Label.RIGHT
),
            "anchor=east;x=0;y=3" );

        addWithConstraints( new Label( "Quantity On Order:", Label.RIGHT
),
            "anchor=east;x=0;y=4" );

        addWithConstraints( new Label( "Last Received Date:", Label.RIGHT
),
            "anchor=east;x=0;y=6" );

        addWithConstraints( new Label( "Comments:", Label.RIGHT ),
            "anchor=east;x=0;y=8" );

        prod_id = new JifTextField( "", "prod_id", true );
        prod_id.setStyle( JifTextField.NUMERIC );
        addWithConstraints( prod_id, "x=1;y=0;width=5;fill=horizontal" );

        desc_text = new JifTextField( "", "desc_text" );
        addWithConstraints( desc_text, "x=1;y=1;width=13;fill=horizontal"
);

        qty_on_hand = new JifTextField( "", "qty_on_hand" );
        qty_on_hand.setStyle( JifTextField.NUMERIC );
        addWithConstraints( qty_on_hand,
"x=1;y=3;width=13;fill=horizontal" );

        qty_on_order = new JifTextField( "", "qty_on_order" );
        qty_on_order.setStyle( JifTextField.NUMERIC );
        addWithConstraints( qty_on_order,
"x=1;y=4;width=13;fill=horizontal" );

        last_rcv_date = new JifTextField( "", "last_rcv_date" );
        addWithConstraints( last_rcv_date,
"x=1;y=6;width=13;fill=horizontal" );

        comment_text = new JifTextField( "", "comment_text" );
        addWithConstraints( comment_text,
"x=1;y=8;width=13;fill=horizontal" );

        //    Disable buttons...
        saveButton.disable();
        chooseButton.disable();
        deleteButton.disable();

        //    Add the buttons...
        addWithConstraints( newButton, "x=15;y=0;width=2;fill=horizontal"
);
        addWithConstraints( saveButton,
"x=15;y=2;width=2;fill=horizontal" );
        addWithConstraints( chooseButton,
```

```
"x=15;y=4;width=2;fill=horizontal" );

        //    Tell which are which...
        prod_id.setPrimaryKey( true );
        prod_id.setNumeric( true );
        qty_on_hand.setNumeric( true );
        qty_on_order.setNumeric( true );
        last_rcv_date.setDate( true );

        //    Set the focus to the first field...
        clearScreen();
    }
```

A notable item about this user interface (and other `GridBagLayout` jiflets in this book) is that the grid settings are hard-coded. A permanent 9 row by 17 column grid was used for the user interface. Only after the grid has been set are components placed within the grid.

First the layout to a new `GridBagLayout` is set. Then the labels that go into the layout next to where the text areas will go are created and placed. Next the `JifTextField`s that represent the columns are placed into the layout.

The New, Save, and Choose buttons are added to the layout as well. These are first disabled because they aren't active until certain events occur.

Finally the screen is cleared of any values, and defaults are placed in the correct columns.

The Product Pick List

Another class was developed for this application, the `ProductPickList` class. This class derives from the `PickList` class of the `jif.awt` package (as discussed in Chapter 11, "User Interface Classes") and presents the user with a selection of products. When one is chosen, the object stores the selection and waits for someone to ask who was chosen. Now take a look at some of the source code.

```
//*************************************************************************
//* ProductPickList                                                       &
nbsp;                  *
//*************************************************************************

public class
ProductPickList
extends PickList
```

As stated, the `PickList` class is extended. When doing this, an `init()` method must be supplied. The following is that method:

```
//*************************************************************************
//* init
                          *
//*************************************************************************

    public void
    init()
    {
        int rows = retrieveProducts();

        if ( rows > 0 && getParent() instanceof Jiflet )
            ( ( Jiflet )getParent() ).verboseLog( "Retrieved " +
```

```
                Integer.toString( rows ) + " Products" );
    }
```

This method calls the `retrieveProducts()` method. Also, if this pick list is used with a jiflet and verbose mode is turned on, the number of products that were retrieved is written to the log file.

The `retrieveProducts()` method, shown in Listing 19.3, is the meat of this class. It performs an `SQL SELECT` statement from the database, parses the results, and places them in the pick list for the user to select from.

Listing 19.3. The `retrieveProducts()` method.

```
//*************************************************************************
//* retrieveProducts
                *
//*************************************************************************

    int
    retrieveProducts()
    {
        String        sql;
        boolean        rv = false;
        int            rows = 0;

        sql = "select * from prod order by desc_text";

        try
        {
            rv = myConnection.getStatement().execute( sql );
        }
        catch ( SQLException e )
        {
            myConnection.errorLog( e.toString() );

            //    No products to return...
            return( 0 );
        }

        //    Is this a result set?
        if ( rv )
        {
            try
            {
                ResultSet rs =
myConnection.getStatement().getResultSet();

                //    Spin through the results and add them to the
list...
                while ( rs.next() )
                {
                    ProductRecord er = new ProductRecord( rs );

                    //    Add to list...
                    if ( er.prod_id != -1 )
                    {
                        myList.addItem( er.desc_text );
```

```
                        //    Add to row mapper...
                        rowMap.insertElementAt( er, rows );

                        //    Increment row counter...
                        rows++;
                    }
                }
            }
            catch ( SQLException e )
            {
                //    Indicate an error!
                rows = -1;
            }
        }

        //    We're done!
        return( rows );
    }
```

The interesting twist here is that each product row is stored in another class called `ProductRecord`. This class has a corresponding instance variable for each column in the employee table. The class is smart and knows how to read a row out of a JDBC `ResultSet` object.

As the results are returned by the SQL statement, new `ProductRecord`s are created. These records are stored in a `Vector` for later use.

At the end, the number of rows that were retrieved are returned and added to the pick list. If there was an error, a `-1` is returned.

The reason each record is stored is for easy access. When the user selects the product he or she wants from the list, you simply ask the pick list to produce a copy of the record that it already retrieved. This is done in the `getRecord()` method:

```
    //*************************************************************************
    //* getRecord                                                 &
    nbsp;                       *
    //*************************************************************************

        public ProductRecord
        getRecord( int where )
        {
            return( ( ProductRecord )rowMap.elementAt( where ) );
        }
```

The pick list returns the index of the selected item. This class uses a neat trick to keep track of what row is where in the `List`. A `Vector` is created called `rowMap`. As a row of data is retrieved from the database and placed into the pick list's `List`, it is also stored in the `Vector` object at the same index level.

Later, when you need a `ProductRecord` from the pick list, instead of rereading the data from the database, you simply retrieve the row from the `Vector`. This is done in the preceding `getRecord()` method. This trick has been used in several applications in this book.

Calling the Pick List

The `ProductPickList` object is created and displayed in the main program when the user presses the Choose button. Listing 19.4 shows how it is done.

Listing 19.4. The `action()` and `chooseProduct()` methods.

```
//*************************************************************************
//* action                                          &nbs
p;                        *
//*************************************************************************

    public boolean
    action( Event event, Object arg )
    {
        if ( event.target == getUIPanel() )
        {
            switch ( ( ( Integer )arg ).intValue() )
            {
                case JifMessage.chOOSE:
                    if ( getDBRecord().didDataChange() )
                    {
                        chgDlg = new ResponseDialog( this,
                        "Data Change",
                        "The record has changed.\n" +
                        "Do you wish to save your changes?",
                        "Yes,No,Cancel" );

                        chgDlg.show();
                    }
                    else
                        chooseProduct();
                    return( true );
            }
        }

        //    Handle picklist events...
        if ( event.target instanceof ProductPickList )
        {
            int                    rv = ( ( Integer )arg ).intValue();
             ProductPickList     epl = ( ProductPickList )event.target;

            if ( rv != -1 )
            {
                //    Disable save on choose...
                getUIPanel().saveButton.disable();

                //    Display it on the screen...
                setDBRecord( ( DBRecord )epl.getRecord( rv ) );
                getUIPanel().moveToScreen();
            }

            //    Kill the dialog box...
            epl.hide();
            epl.dispose();
```

```
            //    Reset the focus...
            getUIPanel().requestFocus();

            //    We handled it...
            return( true );
        }

        //    Not handled...
        return( super.action( event, arg ) );
    }

//************************************************************************
//* chooseProduct                                                  &nb
sp;                  *
//************************************************************************

    public void
    chooseProduct()
    {
        startWait();

        ProductPickList epl = new ProductPickList( this, getConnector()
);

        epl.center( true );
        epl.show();

        endWait();
    }
```

When the Choose button is clicked, a `JifMessage.chOOSE` is sent to the parent. This is received in the `action()` event handler method. At this point, you need to see whether any changes have been made to the currently displayed record. If so, the user is asked whether he or she wants to save them.

If there are no changes to save, the method `chooseProduct()` is called. This method creates and displays a `ProductPickList` object.

When the user selects a pick list item or closes the pick list window, it generates an `ACTION_EVENT` event. This event is captured and appropriate action is taken.

If the pick list returns a `-1` value, you know the user canceled the selection. Otherwise, the value returned is the selected row number. The `ProductRecord` is then retrieved at that row, made the current record, and the user interface displays it.

Finally, a little cleanup is in order. `hide()` and `dispose()` of the pick list window, and then reset the focus back to the window.

Note
The `ProductPickList` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications. This and all of the other source code is on the CD-ROM that accompanies this book.

Database Access

This application communicates with the database through the use of a `ProductRecord` object. This `DBRecord` derivation knows how to create, read, update, and delete records from the News table. The following are the instance variables of this class:

```
    //***************************************************************************
    //* Constants                                                   &
    nbsp;                       *
    //***************************************************************************

        public final static String     TABLE_NAME = "prod";

    //***************************************************************************
    //* Members                                                  &nb
    sp;                         *
    //***************************************************************************

        //    A variable for each table column...
        public int                      prod_id = -1;
        public String                  desc_text = "";
        public int                      qty_on_hand = 0;
        public int                      qty_on_order = 0;
        public Date                      last_rcv_date = null;
        public String                  comment_text = "";
```

As you can see, each column is represented by an instance variable.

Note
The `ProductRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications. This and all of the other source code is on the CD-ROM that accompanies this book.

Programming Considerations

This simple application presented no unusual programming considerations. It is very similar to the Employee Files application. Both have pick lists to select the current row. The rows can be manipulated by the application in a normal fashion.

To recap, this application introduced the following Java intranet programming topics:

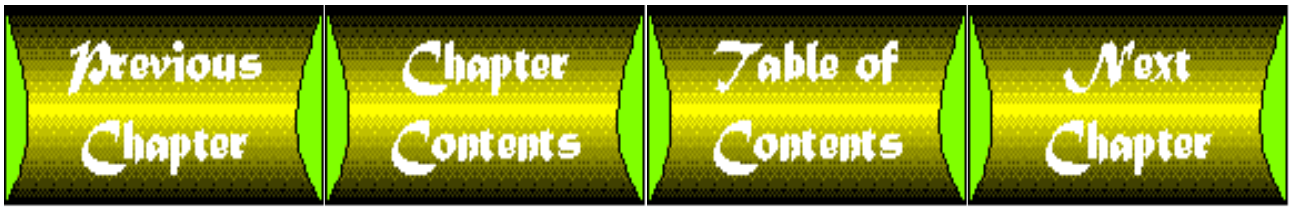
- Using a `GridBagLayout`: A `GridBagLayout` layout manager was used for the application, enabling the quick placement of elements into a hard-coded row/column setting.
- Using a pick list: A pick list was used to present the user with a list of products to choose from.

Summary

This chapter introduced you to the seventh sample application in the intranet application suite-Product Maintenance. This program is responsible for creating and maintaining the product table, and it will be useful for some of the employees. It enables them to view the current product line and stocking levels. It also can be used by telemarketers or sales persons

when taking orders.

In Chapter 20, you are introduced to an application that enables you to track problems with your product line.



Chapter 20

Customer Support Maintenance

CONTENTS

- Introduction
- Application Design
- Database Design
- Implementation
  - Building the User Interface
  - Database Access
  - Retrieving the Product and Problem Lists
  - Product and Problem Selection Changes
- Programming Considerations
- Summary

Introduction

In Chapter 19, "Product Maintenance," you created a data entry program that allows users to create and maintain data in a product table. It is a useful application in itself; however, the application that you are going to create for this chapter relies on the data entered by that program. This is the Customer Support Maintenance program.

Although this is the last sample application of the book, it certainly is not the least. It is the second most complex intranet application that you've created, the first being the Customer Support Maintenance application. This application allows customer service representatives on your intranet to track problem reports generated by your customers.

This chapter follows the same topic format as the last chapters and covers the following topics in regards to the Customer Support Maintenance application:

- Application Design-This section covers the general design of the application, including functionality and user interface considerations.
- Database Design-This section covers the database requirements for this application. Here you examine the data model used to support the application design.
- Implementation-This section covers how the application and database design are implemented.
- Programming Considerations-In this section, I'll recap the implementation and summarize any difficult programming situations that have arisen.

This four-step format is used throughout all of the sample application chapters. Hopefully, it will provide you with valuable insight and ideas for creating your own intranet applications.

Application Design

Remember that your goal is to make intranet users want to use the application. Simply placing an application out on the network is not enough. A program has to be functional, goodlooking, and most of all, easy to use.

This application has been designed primarily for ease of use. Figure 20.1 is the proposed user interface for the Customer Support Maintenance program.

Figure 20.1 : *The Customer Support Maintenance user interface*.

At the top are two lists. The list on the left contains all the available products. The list on the right contains all of the problems for a given product.

The product list on the left frees you from the need to have a product pick list in this application. At startup, all of the products are displayed.

The problem list on the right is not active until a product is selected on the left. After the user selects a product, any problems associated with that product are retrieved from the database and displayed in that list.

When choosing a problem from the list, the user might add a resolution or, perhaps, augment the current problem. Maintenance is fairly free-form. Changes are saved with the Save button.

By selecting New, a new problem can be entered for the chosen product. This new problem can also be saved with the Save button.

Figure 20.2 shows what the screen program will look like when editing an existing problem.

Figure 20.2 : *The application editing an existing problem.*

Database Design

This application will be responsible for manipulating problem records. These records represent a problem that a customer has or had with a product. Each stored row represents a single problem. Your SQL to retrieve and update the table will be simple because each problem has a unique identifying number. The table you're going to use in this sample application is called the product problem table.

This table is a child table from the product table described in Chapter 19.

Table 20.1 shows the columns that you need to store your problem information.

Table 20.1. The product table layout.

Description	*Column Name*	*Type*	*Can Be Null?*	*Default*
Product ID	prod_id	number(5)	No	None
Problem ID	prob_id	number(5)	No	None
Description	desc_text	char(255)	No	None
Reported By	reported_by_text	char(80)	Yes	None
Problem Resolution	resolution_text	char(255)	Yes	None
Problem Date	start_date	date	Yes	None
Resolution Date	end_date	date	Yes	None

Figure 20.3 shows the entity relationship diagram for your database. This data model diagram represents the entire

database developed for this book.

Figure 20.3 : *The entity relationship diagram with employee tables.*

Note
Entity relationship diagrams are discussed in Chapter 13, "Employee Files."

In addition to creating a table, you create a database synonym for your table. This allows everyone to access the table with the same name, without having to worry about the schema that the table resides in.

Listing 20.1 shows the SQL commands to create the product problem table.

Listing 20.1. SQL commands.

```
/*    Create the table */
create table prod_prob_t
(
    prod_id                number( 5 ) not null,
    prob_id                number( 5 ) not null,
    desc_text             char( 255 ) not null,
    reported_by_text    char( 80 ),
    resolution_text        char( 255 ),
    start_date             date default sysdate,
    end_date             date
);

/*    Create a primary key */
alter table prod_prob_t
    add
    (
        primary key
        (
            prod_id,
            prob_id
        )
    );

/*    Create a foreign key */
alter table prod_prob_t
    add
    (
        foreign key
        (
            prod_id
        )
        references prod_t
    );

/*    Grant access for the table to the user role */
grant select,insert,delete,update on prod_prob_t to ia_user_r ;

/*    Drop any existing public synonym */
drop public synonym prod_prob ;
```

```
/*    Create a public synonym for our table */
create public synonym prod_prob for prod_prob_t ;
```

This SQL is similar to table creation presented in the previous chapters. After the table is created, a primary key is created. Then a foreign key is created to reference the product table. Access rights are granted to your demonstration user, and a public synonym is created.

Caution
You must create the `prod_t` (Product) table before you can create the `prod_prob_t` (Product Problem) table. Otherwise, the `prod_prob_t` SQL will fail!

Implementation

The rest of this chapter covers the implementation of the Customer Support Maintenance program. First, I'll discuss the user interface and how it was created. Secondly, I'll discuss the database access used in the program. Finally, I'll go over some of the programming considerations that came up during the application construction.

Building the User Interface

The Customer Support Maintenance program is the second most complex program in this book. It utilizes some of the cooler user interface classes discussed in Chapter 11, "User Interface Classes." However, it doesn't operate like any other application in the book. This application does use the `SimpleDBJiflet`, `SimpleDBUI`, and `DBRecord` classes, although their use is unconventional compared to the last applications.

This application consists of two separate sets of components on one screen. This is done by creating two subpanels and inserting them into a `BorderLayout`. Figure 20.4 illustrates your user interface.

Figure 20.4 : *The Customer Support Maintenance layout.*

Listing 20.2 shows the source code for the construction of the user interface.

Listing 20.2. The Product Maintenance interface construction code.

```
//************************************************************************
//* Members                                          &nb sp;  *
//************************************************************************

   List                prodList = new List();
   List                probList = new List();

   JifTextField          prod_id = new JifTextField( "", "prod_id" );
   JifTextField          prob_id = new JifTextField( "", "prob_id" );
   JifTextArea           prob_desc_text =
                       new JifTextArea( "", "desc_text" );
   JifTextField          prob_rep_by_text =
                       new JifTextField( "", "reported_by_text" );
   JifTextArea           resolution_text =
                       new JifTextArea( "", "resolution_text" );
   JifTextField          start_date =
                       new JifTextField( "", "start_date" );
   JifTextField          end_date = new JifTextField( "", "end_date" );

   Vector               rowMap = new Vector( 5 );
```

```
   Vector               prodMap = new Vector( 5 );

//*************************************************************************
//* Constructor                                                           ; *
//*************************************************************************

   public
   ProblemTrackerUI( SimpleDBJiflet jiflet )
   {
      super( jiflet );
      setLayout( new BorderLayout() );

      //   Build a big panel...
      JifPanel p2 = new JifPanel();
      p2.setLayout( new BorderLayout() );

      //   Build a label panel...
      JifPanel p1 = new JifPanel();
      p1.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
      p1.setLayout( new GridLayout( 1, 2, 5, 5 ) );
      JifPanel tp = new JifPanel( JifPanel.LOWERED );
      tp.setText( "Products", JifPanel.TEXT_RAISED, JifPanel.CENTER );
      p1.add( tp );
      tp = new JifPanel( JifPanel.LOWERED );
      tp.setText( "Problems", JifPanel.TEXT_RAISED, JifPanel.CENTER );
      p1.add( tp );
      p2.add( "North", p1 );

      //   Build a listbox panel...
      p1 = new JifPanel();
      p1.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
      prodList.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
      probList.setFont( new Font( "Helvetica", Font.PLAIN, 14 ) );
      p1.setLayout( new GridLayout( 1, 2, 5, 5 ) );
      p1.add( prodList );
      p1.add( probList );
      p2.add( "Center", p1 );

      //   Build a UI panel...
      p1 = new JifPanel();
      GridBagLayout gbl = new GridBagLayout();

      int cw[] = { 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10 };
      double rc14_0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

      gbl.columnWidths = new int[ 13 ];
      gbl.rowHeights = new int[ 10 ];

      gbl.columnWeights = new double[ 13 ];
      gbl.rowWeights = new double[ 10 ];

      System.arraycopy( cw, 0, gbl.columnWidths, 0, 13 );
      System.arraycopy( cw, 0, gbl.rowHeights, 0, 10 );
```

```
        System.arraycopy( rc14_0, 0, gbl.columnWeights, 0, 13 );
        System.arraycopy( rc14_0, 0, gbl.rowWeights, 0, 10 );

        p1.setLayout( gbl );

        p1.addWithConstraints( new Label( "Problem:", Label.RIGHT ),
          "anchor=east;x=0;y=0" );
        p1.addWithConstraints( prob_desc_text,
          "x=1;y=0;width=9;height=3;fill=both" );
        p1.addWithConstraints( new Label( "Resolution:", Label.RIGHT ),
          "anchor=east;x=0;y=4" );
        p1.addWithConstraints( resolution_text,
          "x=1;y=4;width=9;height=3;fill=both" );
        p1.addWithConstraints( new Label( "Reported By:", Label.RIGHT ),
          "anchor=east;x=0;y=7" );
        p1.addWithConstraints( prob_rep_by_text,
          "x=1;y=7;width=9;fill=horizontal" );
        p1.addWithConstraints( new Label( "Problem Date:", Label.RIGHT ),
          "anchor=east;x=0;y=8" );
        p1.addWithConstraints( start_date, "x=1;y=8;width=9;fill=both" );
        p1.addWithConstraints( new Label( "Resolution Date:", Label.RIGHT
),
          "anchor=east;x=0;y=9" );
        p1.addWithConstraints( end_date, "x=1;y=9;width=9;fill=both" );

        /*
        * Note: The prod_id and prob_id fields need to get on the panel.
        * If they don't, the SQL is not generated for them, and you cannot
        * save anything. I trick them onto the panel below. I don't want
        * the user to see them, so I hide them after adding them behind
        * another component.
        *
        * Try commenting out the hide() calls below and see what happens.
        * It is pretty cool! ;)
        */

        //   Hide this behind the new button...
        p1.addWithConstraints( prod_id, "x=11;y=0;width=2;fill=horizontal"
);
        p1.addWithConstraints( newButton,
"x=11;y=0;width=2;fill=horizontal" );

        //   Hide this behind the save button...
        p1.addWithConstraints( prob_id, "x=11;y=2;width=2" );
        p1.addWithConstraints( saveButton,
"x=11;y=2;width=2;fill=horizontal" );

        //   Hide the two fields...
        prod_id.hide();
        prob_id.hide();

        //   Make the sauce...
```

```
        add( "North", p2 );
        add( "Center", p1 );

        //   Disable buttons...
        newButton.disable();
        saveButton.disable();

        //   Tell which are numeric...
        prob_id.setNumeric( true );
        prod_id.setNumeric( true );
        prob_id.setPrimaryKey( true );
        prod_id.setPrimaryKey( true );
        start_date.setDate( true );
        end_date.setDate( true );

        clearScreen();
    }
```

First, build a panel that contains two labels and two list boxes. These make up the top half of the interface. Then create a panel that holds the `GridBagLayout` of text fields and areas. These are where the user will type data.

The two panels are then placed into a master `BorderLayout`.

Hiding Components

This application uses a cool trick. There are two columns, `prod_id` and `prob_id`, that the user should never see. However, they are required to be part of the layout because you need the values they hold for proper SQL generation by your `SQLFactory` classes.

The solution is to add them to the layout, but make them invisible from the user. You accomplish this by placing them behind the two buttons and then hiding them. The following is the source code:

```
            //    Hide this behind the new button...
            p1.addWithConstraints( prod_id,
    "x=11;y=0;width=2;fill=horizontal" );
            p1.addWithConstraints( newButton,
    "x=11;y=0;width=2;fill=horizontal" );

            //    Hide this behind the save button...
            p1.addWithConstraints( prob_id, "x=11;y=2;width=2" );
            p1.addWithConstraints( saveButton,
    "x=11;y=2;width=2;fill=horizontal" );

            //    Hide the two fields...
            prod_id.hide();
            prob_id.hide();
```

Without hiding the components, you get a weird double-component look that is not natural. But when they are hidden, only the container itself knows that they are there.

Being there, they can store values and generate SQL code. It's a pretty cool trick.

Database Access

This application communicates with the database through the use of the `DBRecord` extension class `ProductProblemRecord`. This class is used solely to retrieve product problem table rows from the database.

Note
The `ProductProblemRecord` and other database classes are reused in several other applications. They have been placed in their own package along with other shared code. This package is called `jif.common`. It contains all the common classes between all the applications.

Before you can edit problems, you need the user to select a product.

Retrieving the Product and Problem Lists

Before the user can edit product problems, you must populate the list of products. This is done the same way the product pick list was done in the previous chapter:

```
//************************************************************************
//* loadLists                                                &
nbsp;                      *
//************************************************************************

    public void
    loadLists()
    {
        newButton.disable();

        getJiflet().showStatus( "Loading lists..." );
        getJiflet().startWait();

        prodList.clear();

        prodMap.removeAllElements();
        int rows = 0;

        probList.clear();

        try
        {
            String sql = "select * from prod order by desc_text";

            ResultSet rs =
                getJiflet().getConnector().getStatement().executeQuery(
sql );

            while ( rs.next() )
            {
                ProductRecord pr = new ProductRecord( rs );
                prodList.addItem( pr.desc_text );
                prodMap.insertElementAt( pr, rows );
                rows++;
            }
```

```
            if ( rows > 0 )
            {
                prodList.select( 0 );
                loadProblemList();
            }
        }
        catch( SQLException e )
        {
            getJiflet().getConnector().errorLog( e.toString() );
        }

        getJiflet().endWait();
        getJiflet().showStatus( "Products Loaded!" );
    }
```

Retrieve all the products in sorted order. Each one is stored in your product cache and then placed into the list. After all of the products have been loaded, select the first one in the list and load any problems associated with that product:

```
    //*************************************************************************
    //* loadProblemList                                                       &
    nbsp;                  *
    //*************************************************************************

        /**
        * Load the problem lists...
        */

        public void
        loadProblemList()
        {
            getJiflet().showStatus( "Loading problems..." );

            //    Set the current product id...
            ProductRecord pr =
                ( ProductRecord )prodMap.elementAt(
    prodList.getSelectedIndex() );

            prod_id.setText( Integer.toString( pr.prod_id ) );

            probList.clear();
            rowMap.removeAllElements();

            //    Clear out any records...
            getJiflet().getDBRecord().clear();
            clearScreen();

            int rows = 0;

            try
            {
                String sql = "select * from prod_prob " +
                    "where prod_id = " + Integer.toString( pr.prod_id ) + " "
    +
```

```
                    " order by desc_text";

                ResultSet rs =
                    getJiflet().getConnector().getStatement().executeQuery(
    sql );

                while ( rs.next() )
                {
                    ProductProblemRecord ppr =
                        ( ProductProblemRecord )getJiflet().getDBRecord();

                    ppr.parseResultSet( rs );

                    String s = Integer.toString( ppr.prob_id );
                    s += " ";
                    s += ( new FileDate( ppr.start_date ) ).toNormalString();
                    s += " ";
                    s += ( new FileDate( ppr.end_date ) ).toNormalString();
                    s += " ";
                    s += ppr.desc_text;
                    probList.addItem( s );

                    //    Make a row map...
                    rowMap.insertElementAt( ppr, rows );
                    rows++;
                }

                if ( rows > 0 )
                {
                    probList.select( 0 );
                    showProblemDetail();
                }
            }
            catch( SQLException e )
            {
                getJiflet().getConnector().errorLog( e.toString() );
            }

            getJiflet().showStatus( "Problems Loaded!" );
            newButton.enable();
        }
```

This routine relies on the fact that a product has been selected. It retrieves the selected index from the product `List` and uses that to get the product ID to use for its lookup.

After the problems for that particular product ID have been retrieved, they are loaded into the problem `List`. After all of these have been loaded, the first one is chosen by default.

When it is chosen, the problem detail is displayed:

```
    //*************************************************************************
    //* showProblemDetail
    ;                *
    //*************************************************************************
```

```
    public void
    showProblemDetail()
    {
        //    Get my record...
        ProductProblemRecord ppr =
            ( ProductProblemRecord )rowMap.elementAt(
                probList.getSelectedIndex() );

        //    Fill in the fields...
        getJiflet().setDBRecord( ppr );
        moveToScreen();
    }
```

Here, you utilize the display mechanism built into your `SimpleDBUI` class. This enables you to use the `setDBRecord()` method and call the `moveToScreen()` method. This method formats the data and moves it to the screen.

Product and Problem Selection Changes

When the user changes products, you need to reload the problem list. To do this, trap some of the events for your lists. When a list item is selected, an `Event.LIST_SELECT` is generated. Simply capture this event and reload accordingly:

```
//*************************************************************************
//* handleEvent
;                     *
//*************************************************************************

    public boolean
    handleEvent( Event event )
    {
        if ( event.target == prodList )
        {
            if ( event.id == Event.LIST_SELECT )
            {
                loadProblemList();
                return( true );
            }
        }

        if ( event.target == probList )
        {
            if ( event.id == Event.LIST_SELECT )
            {
                showProblemDetail();
                return( true );
            }
        }

        return( super.handleEvent( event ) );
    }
```

When the product list has a new selection, call the `loadProblemList()` method. When the problem list changes, show the problem detail with the `showProblemDetail()` method.

Programming Considerations

This application presented you with an interesting user interface challenge, not to mention two table database lookups. You needed to present an intuitive interface to the user that was easy to use.

You also enhanced the interface using nested layout managers. You nested a `GridLayout` within a `BorderLayout` to space your components evenly. You also nested a `GridBagLayout` within another `BorderLayout`.

To recap, this application introduced the following Java Intranet programming topics:

- Nested Layout Managers: You nested several layout managers into your master layout. This was illustrated in Figure 20.4.
- Hidden Components: You needed two components to hold a portion of your SQL statement, but they shouldn't be seen by the user. So, they were hidden behind the New and Save buttons.
- Cascading Lookups: When the user selects a product, the problems need to be retrieved for that product. Then, when the user selects a problem, the problem detail needs to be displayed. This cascading lookup is used in this application.

Summary

This chapter introduced you to the final sample application in your intranet application suite: the Customer Support Maintenance application. This program is responsible for tracking customer support information regarding problems with the products that you sell. This application should be useful to customer service as well as technical support employees on your intranet.

In the next chapter, "Extending the Java Intranet Framework," I'll talk about the future of the Java Intranet Framework (JIF) and your intranet.

Chapter 21

Extending the Java Intranet Framework

CONTENTS

- Introduction
- jif.util
  - ConfigProperties
  - FileDate
- jif.log
- jif.sql
- jif.awt
  - JifPanel
  - JifTabPanel
  - StatusBar
  - Miscellaneous
- jif.jiflet
- Extending the Applications
  - Benefits Maintenance
  - Conference Room Scheduling
  - News and Announcements
- Summary

---

ex tend \ik-stend' \ v: *to make longer, wider, or braoder*

Introduction

As a programmer, you know a program is never complete. It is a rare occasion when you can walk away from an application saying, "That's it! I can add nothing else!" It is a very rare occasion indeed. Well, the software in this book is no different. There is always room for improvement.

In any case, this chapter will cover a few areas where you might be able to extend the Java Intranet Framework to get even more use from it. Some of these ideas came to me during the writing of this book,

others I purposely left out for you, the reader/programmer, to create yourself. What fun is it if I do everything?

Take a look back at each of the separate packages and go over some of the ideas for extending those classes.

jif.util

The `jif.util` package contains several utility classes. They are created for demonstration purposes and for use in the applications in this book. Of all those classes, only the `FileDate` and `ConfigProperties` classes have room for extension.

The timers can be extended; however, you would be pushing the functionality beyond a simple timer. They should be used within classes rather than be subclassed. However, it is always the case that classes developed by one person are used by another in a manner that the author never thought of.

ConfigProperties

Although this class is quite useful, it does do one specific thing. If it finds a property called `jdbc.drivers` in your property table, it adds it to the system property table. This mechanism is very specific. It only looks for that single property.

An excellent enhancement to the `ConfigProperties` class would be to make this mechanism more configurable. This means that you would be able to tell the class which properties would be merged with the system property table. The class could then keep an array of these properties and deal with them at runtime.

Another thing that you could do is make the `ConfigProperties` class merge all of the private properties with the system property table. Then the overridden `getProperty()` method could call the `System.getProperty()` method directly.

FileDate

This class went through many changes over the course of writing this book. It went from having only a single purpose to having many. Although not perfect, this final version of `FileDate` is quite versatile.

One idea for extending this class is to create a more user-friendly date formatter, perhaps one that is not tied to constants. This formatter would possibly accept as input a string representing the format that you would like your date returned. This format string could use simple MM-DD-YY strings.

jif.log

The logging classes presented in this book are quite simple. They write a static format out to the log. These classes could easily be rearranged and extended to provide a much more robust logging system.

One example is to abstract the formatting of the log entry out to a separate class. Make it a stream class

like some of the Java classes. This class would be the only one that knows how to format log entries. You could then create output classes that plug in to the formatter class. This is probably a far more object-oriented approach than the one taken in this book.

If you did do this, you could then make the formatter configurable as well. Instead of always putting out the same information, let the consumer or user of the object specify in what fields, and in what order, they are written to the log. This would give you a robust and configurable logging system.

jif.sql

The classes in this package represent database communications. These classes are very useful. However, with a little work, you might not have to code much anymore.

The next logical step for the classes in this package is for them to automatically query the database. This query would retrieve the table structures from the database. The classes could then dynamically construct themselves to mimic each table. This is pretty extreme, but definitely possible.

Another, more likely extension, is to create a class that represents a window into a table. This window would allow the programmer to specify the SQL statement that loads data. This information is then available to be put through a user interface. The class could then maintain a cursor, or pointer, as to which record is current, and automatically scroll through the data using the same user interface.

Then, once changes have been made, all the SQL would be generated to update all the data changes that were made.

jif.awt

Being the largest package in this book, the `jif.awt` class has many areas where it can be extended. Take a look at some of them:

JifPanel

Currently, the `JifPanel` is static in size. One extension could be to make it more dynamic and let it grow with its surroundings. If a resized window contains a `JifPanel`, the `JifPanel` might want to grow with the window. Getting this to work with hard-coded sizes, though, might present a programming challenge.

Also, the `JifPanel` currently draws its three-dimensional frames in the old Microsoft Windows v3.x style. Another extension could be to upgrade to the newer Microsoft Windows 95/NT look. This 3-D look is cleaner and more modern-looking. It is used in the `JifTabPanel`.

JifTabPanel

The `JifTabPanel` is one of the more complex classes in the `jif.awt` package. It consists of two subclasses that do its bidding, but it is far from perfect. The following are some ideas for enhancing the `JifTabPanel`.

Currently there is no way to disable a tab. You could add the functionality to enable and disable tabs in the panel, but this would restrict users from looking at data on that pane if it wasn't visible.

The `JifTabPanel` has two operating modes. In the first mode, the entire width of the bounding rectangle is used for tab space. This means that if there are two tabs, the width of each tab is relative to the width of the parent container.

In the second mode, the tab width is relative to the width of the text string that is displayed by the tab. This is the default mode.

However, if more tabs exist than can be shown, they are cut off. An excellent enhancement to this class would be one that places a set of buttons at the end of the tab panel to move backwards and forwards. This would avoid the loss of tabs and scroll the entire tab portion so the user can see what lies behind. This effect can be seen in several Microsoft Windows products today.

Another enhancement would be to let the programmer/user specify the font. Currently the fonts are hard-coded to be dialog-style fonts. Perhaps someone would like other fonts.

This class is far from perfect, but it can be extended to be an awesome class.

`StatusBar`

This class could be extended to allow the programmer/user to select the font and style of the border. This information is currently hard-coded in the `StatusBar` class.

Miscellaneous

There are only a handful of Java components that have been enhanced to generate SQL. This is an obvious area of extension.

One good choice for extending in this regard is the `Choice` class. It could be made to generate SQL for you. In addition, you could create an extension that would act like a code table. It would display descriptive text while returning a code from the selection. This is a very useful feature.

`jif.jiflet`

Although complex, and the user of many classes, the `Jiflet` class really offers no areas for extension. There is one possible enhancement that could be made.

You might want to enhance the `Jiflet` class to descend from a `Thread`. This would make the main jiflet run as a thread. You could then stop and start it at will. It might come in handy sometime.

You also might explore the possibility of extending the jiflet to work as an applet instead of as an application.

Extending the Applications

The applications in this book, though useful, are not all perfect. This section will detail some areas where they could be spruced up by you.

Currently, all the applications require the `DBConnector` to be created for them. You could enhance the core `Jiflet` class to build a `DBConnector` based on a configuration property. This would allow you to change databases on-the-fly with a configuration file change, instead of a source code change. This would eliminate the need for recompilation as well.

In addition, each of the applications require you to manually connect with the database. This could be made automatic at startup. Then the user would not be required to perform this action.

Take a look at some specific enhancements.

Benefits Maintenance

This application can be expanded to do much more. Currently, it only manages the lowest common denominator. Many retirement plans today allow you to change your investments in different areas. This is not represented at all by your maintenance program.

This would, however, make an awesome enhancement. Allow the user to choose the fund and percentage of his money that would go into that fund. You would need a separate table to hold the new information, but the usefulness of the application would jump tenfold. This information could then be sent electronically to the benefits administrator, or even directly to the benefits company.

Conference Room Scheduling

One of the features of this program was left for the reader to complete: the delete function. You currently cannot remove a previously scheduled conference room. This is a good place to jump in and get your feet wet.

In addition, there is no program for creating conference rooms. They must be entered through a database manipulation tool, or a program needs to be created. This is a great small jiflet. You can model it after the Employee Files application.

News and Announcements

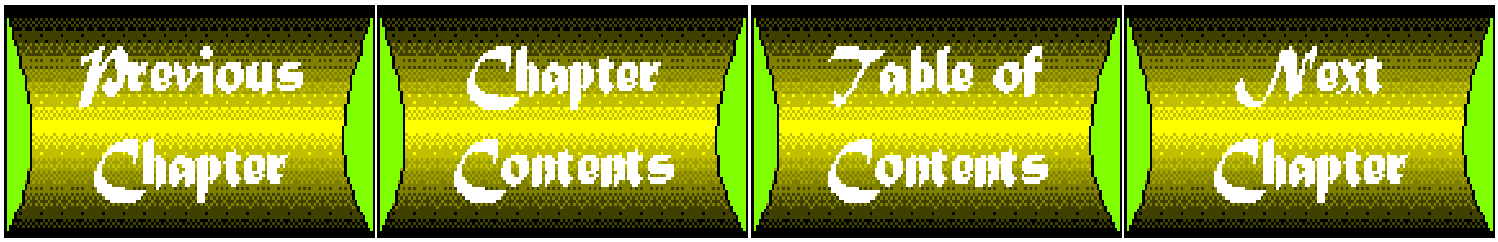
This application lets you enter News and Announcements, but there is no way to delete them. A likely addition to this program would let you delete news items that are of a certain age, or perhaps not show them.

Also, there is no refresh in this application. A good extension would be to program in an automatic timed refresh like the one in the Online In/Out Board.

Summary

This chapter recapped some of the areas where the Java Intranet Framework could be enhanced to provide more flexibility and functionality. I covered each package individually and noted areas of improvement. I also looked at areas of improvement in the sample applications.

This brings you to the end of the book. I hope you have enjoyed reading it. I also hope you have learned how you can use Java to develop some killer intranet applications for your company or customers. Please feel free to e-mail me your comments, suggestions, gripes, and so on. My e-mail address is listed in the front of the book. I would love to hear what you think about Java and intranets.



---

appendix A

Java Resources

CONTENTS

- Sun's Java Sites
- Java Information Collection Sites
- Java Discussion Forums
- Notable Individual Java Webs
- Java Index Sites
- Object-Oriented Information
- Java Players and Licensees

This is a list of online information sources about the Java programming language, Java-enabled browsers, and related technologies.

Much of the information in this appendix changes often. For updates on the Java information sources listed in this appendix, visit `http://www.december.com/works/java/info.html`

Note

Sun's Java Sites

Sun Microsystems, the developers of Java, offers the best online one-stop, comprehensive source of technical documentation and information about Java on its Web site at `http://java.sun.com/`.

Also, see Sun's Java contest Web site, `http://javacontest.sun.com/`.

Some sites around the world mirror this information, so you can pick a server from the list below that is close to you, or consult Sun's list of Java mirror sites. Here are sites that are known to be updated frequently:

- Dimension X, San Francisco, California, `http://www.dnx.com/java/`
- JavaSoft (Sun Microsystems), Aspen, Colorado, `http://www.javasoft.com/`
- Wayne State University, Detroit, Michigan, `http://www.science.wayne.edu/java/`
- SunSITE Singapore, National University of Singapore, `http://sunsite.nus.sg/hotjava/`

Java Information Collection Sites

These are high-level Java information collection sites outside of Sun Microsystems:

- Gamelan: an excellent collection of Java demos and information; includes a large collection of Java applets; well-organized and frequently updated.
  `http://www.gamelan.com/`
- JavaScript information: contains links to information to JavaScript; includes technical information links plus links to sample applications.
  `http://www.c2.org/~andreww/javascript/`
- WWW Virtual Library entry for Java: This includes links to events, reference information, resources, and selected applications/examples.
  `http://www.acm.org/~ops/java.html`
- JARS: Java Applet Rating Service; the hope here is to rate the best applets (top 1 percent, 5 percent, and so on). Includes categories of applets.
  `http://www.surinam.net/java/jars/jars.html`
- `Applets.com`: a collection of applets, including a list of new applets, an evaluated list of "cool" applets, and a library of applets; from Applets Corporation.
  `http://www.applets.com/`
- Java Study Group Homepage: This is a study group of the New York City C++ and C Special Interest Group whose parent organization is the New York City pc Users Group. This Web includes group meeting notes, upcoming speakers, Java information resources, and other information.
  `http://www.inch.com/~nyjava/`
- Java Developer: "A public service FAQ devoted to Java programming" includes resources, a job clearing house, and a large section on "How Do

I...."
http://www.digitalfocus.com/digitalfocus/faq/

- `comp.lang.java` FAQ: Frequently Asked Questions (FAQ) for `comp.lang.java`; includes basic information about participating in the discussion and reference information.
  http://www.city-net.com/~krom/java-faq.html

Java Discussion Forums

These are forums where you can take part in or monitor discussions about Java:

- `comp.lang.java`: Usenet newsgroup for discussing the Java programming language.
  `news:comp.lang.java`
- Digital Expresso: (the Web formerly known as "J*** Notes") a weekly summary of the traffic in Java newsgroups and mailing lists.
  http://www.io.org/~mentor/DigitalEspresso.html

Notable Individual Java Webs

These individual Webs focus on some specific aspect of Java development, information, or products:

- Commercial Java Products: a description of a variety of commercial products related to Java from Internet World.
  http://www.iworld.com/InternetShopper/1Java_products.html
- Java Class Hierarchy Diagrams: diagrams that show the class hierarchy for Java packages; very useful for quickly getting an idea of Java class relationships; developed by Charles L. Perkins.
  `http://rendezvous.com/java/hierarchy/index.html">>http://rendezvous.com/java/hierarchy/index.html`
- Programming Active Objects in Java, by Doug Lea: a discussion of object-oriented design issues and features as they relate to Java.
  http://g.oswego.edu/dl/pats/aopintro.html
- Java Online Bibliography: a listing of key online articles and press releases about Java and related technology.
  http://www.december.com/works/java/bib.html

Java Index Sites

These are indexes of Java information:

- Yahoo! Index entry for Java
  http://www.yahoo.com/Computers/Languages/Java/
- Yahoo! Index entry for HotJava Web browser
  http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Browsers/HotJava/

Object-Oriented Information

Because Java is an object-oriented language, these sites can help you connect to more information about object-oriented terminology, design, and programming:

- Object-Oriented Information Sources Index: a searchable index of a variety of object-oriented information sources, including research groups, archives, companies, books, and bibliographies.
  http://cuiwww.unige.ch/OSG/OOinfo/
- Object-Oriented Design Online Reference Guide: a guide to online information sources about object-oriented design. This guide was created by Howie Michalski, Lead Database Engineer, Infrastructure, CompuServe, Inc.
  http://www.clark.net/pub/howie/OO/oo_home.html
- C++ Glossary: This glossary lists terms related to the C++ language. Because C++ is closely related to Java, these terms are also important in Java development.
  http://info.desy.de/pub/uugna/html/cc/text/glossary/index.html

Java Players and Licensees

These are key press announcements from licensees and developers of Java products:

- Adobe Systems, Inc.: "Adobe To Integrate Java Technology for Web Publishing." Press release, December 7, 1995. (Portable Document Format.)
  http://www.adobe.com/PDFs/PR/9512/951206.java.pdf
- Borland International: "Borland International to Deliver Tools For Sun's Java Internet Programming Language." Press release, November 8, 1995.

http://www.borland.com/Product/java/javapress.html

- IBM Corporation: "IBM licenses Java technology from Sun Microsystems for use in Internet products." Press release, December 6, 1995. http://www.ibm.com/News/javapr.html
- Lotus Development Corporation: "Lotus Outlines Plans to Deliver Powerful Integration of Notes And World Wide Web-Announces Lotus Notes Server That Includes Native Notes Support for HTTP, HTML and Java Technology." Press release, December 13, 1995. http://www.Lotus.com/corpcomm/3406.htm
- Macromedia: "Macromedia & Sun Microsystems To Develop Internet Tools And Technology." Press release, October 30, 1995. http://www.macromedia.com/Industry/Macro/Ucon/News/sun.html
- Metrowerks, Inc.: "Metrowerks Collaborates with Sun Microsystems to Provide Java Programming Tools in CodeWarrior Product for Macintosh." Press release, November 10, 1995. http://www.metrowerks.com/news/press/java.html
- Microsoft Corporation: "Internet Strategy Day." Press release, December 7, 1995. http://www.microsoft.com/internet/press.htm
- Natural Intelligence, Inc.: "'ROASTER' Announcement." Press release, October 18, 1995. http://www.natural.com/pages/products/roaster/roasterpr.html
- Netscape Communications Corporation: "Netscape and Sun Announce Javascript, the Open, Cross-Platform Object Scripting Language for Enterprise Networks and the Internet." Press release, December 4, 1995. http://home.netscape.com/newsref/pr/newsrelease67.html
- Netscape Communications Corporation: "Netscape to License Sun's Java Programming Language." Press release, May 23, 1995. http://home.netscape.com/newsref/pr/newsrelease25.html
- Oracle Corporation: "Oracle PowerBrowser to Integrate Java Technology with Oracle's Network Loadable Objects." Press release, October 30, 1995. http://www.oracle.com/info/news/java.html.
- Silicon Graphics, Inc.: "Silicon Graphics, Sun Microsystems and Netscape join forces to take the Web to the next level." Press release, December 4, 1995. http://www.sgi.com/Products/cosmo/sgisun.html
- Spyglass, Inc.: "Spyglass Licenses Sun Microsystems' Java for Spyglass Mosaic." Press release, November 8, 1995. http://www.spyglass.com/current/nov895.html
- Symantec Corporation: "Symantec Releases first Java development environment for Windows 95/NT." Press release, December 13, 1995. http://www.Symantec.com/lit/dev/javapress.html

---

---

appendix B

JDK Tools Reference

CONTENTS

- JDK Tools Reference
- javac-The Java Compiler
  - Synopsis
  - Description
  - Options
  - Environment Variables
- java-The Java Interpreter
  - Synopsis
  - Description
  - Options
- jdb-The Java Debugger
  - Synopsis
  - Description
  - Options
- javah-C Header and Stub File Generator
  - Synopsis
  - Description
  - Options
- javap-The Java Class File Disassembler
  - Synopsis
  - Description
  - Options
- javadoc-The Java API Documentation Generator
  - Synopsis
  - Description
  - Options

- appletviewer-The Java Applet Viewer
  - Synopsis
  - Description
  - Options

JDK Tools Reference

This appendix is a reprint of the JDK Tools Reference documentation written by Sun Microsystems. The material is unaltered from the documentation version and is a copyright of Sun Microsystems, Inc. Reprinted with permission.

`javac`-The Java Compiler

`javac` is the program that compiles Java source code into byte-code format.

Synopsis

```
javac [ options ] filename.java ...
javac_g [ options ] filename.java ...
```

Description

The `javac` command compiles Java source code into Java bytecodes. You then use the Java interpreter-the `java` command-to interpret the Java bytecodes.

Java source code must be contained in files whose filenames end with the `.java` extension. For every class defined in the source files passed to `javac`, the compiler stores the resulting bytecodes in a file named `classname.class`. The compiler places the resulting `.class` files in the same directory as the corresponding `.java` file (unless you specify the `-d` option).

When you define your own classes you need to specify their location. Use `CLASSPATH` to do this. `CLASSPATH` consists of a semicolon-separated list of directories that specifies the path. If the source files passed to `javac` reference a class not defined in any of the other files passed to `javac`, then `javac` searches for the referenced class using the class path. For example:

```
.;C:\users\dac\classes
```

Note that the system always appends the location of the system classes onto the end of the class path unless you use the `-classpath` option to specify a path.

`javac_g` is a non-optimized version of `javac` suitable for use with debuggers like `jdb`.

Options

`-classpath path`

Specifies the path `javac` uses to look up classes. Overrides the default or the `CLASSPATH` environment variable if it is set. Directories are separated by semicolons. Thus the general format for a path is as follows:

```
.;<your_path>
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

`-d directory`

Specifies the root directory of the class hierarchy. Thus doing:

```
javac -d <my_dir> MyProgram.java
```

causes the `.class` files for the classes in the `MyProgram.java` source file to be saved in the directory `my_dir`.

`-g`

Enables generation of debugging tables. Debugging tables contain information about line numbers and local variables-information used by Java debugging tools. By default, only line numbers are generated, unless optimization (`-O`) is turned on.

`-nowarn`

Turns off warnings. If used, the compiler does not print out any warnings.

`-O`

Optimizes compiled code by inlining static, final, and private methods. Note that your classes may get larger in size.

`-verbose`

Causes the compiler and linker to print out messages about what source files are being compiled and what class files are being loaded.

Environment Variables

`CLASSPATH`

Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

See Also

`java, jdb, javah, javap, javadoc`

java-The Java Interpreter

`java` interprets (executes) Java bytecodes.

Synopsis

```
java [ options ] classname <args>
java_g [ options ] classname <args>
```

Description

The `java` command executes Java bytecodes created by the Java compiler-`javac`.

The `classname` argument is the name of the class to be executed. `classname` must be fully qualified by including its package in the name, for example:

```
java java.lang.String
```

Note that any arguments that appear after `classname` on the command line are passed to the class's `main()` method.

`java` expects the bytecodes for the class to be in a file called `classname.class` which is generated by compiling the corresponding source file with `javac`. All Java bytecode files end with the filename extension `.class` which the compiler automatically adds when the class is compiled. `classname` must contain a `main()` method defined as follows:

```
class Aclass
{
    public static void
    main( String argv[] )
    {
            . . .
    }
}
```

`java` executes the `main()` method and then exits unless `main()` creates one or more threads. If any threads are created by `main()` then `java` doesn't exit until the last thread exits.

When you define your own classes you need to specify their location. Use `CLASSPATH` to do this. `CLASSPATH` consists of a semicolon-separated list of directories that specifies the path. For example:

```
.;C:\users\dac\classes
```

Note that the system always appends the location of the system classes onto the end of the class path

unless you use the `-classpath` option to specify a path.

Ordinarily, you compile source files with `javac` then run the program using `java`. However, `java` can be used to compile and run programs when the `-cs` option is used. As each class is loaded, its modification date is compared to the modification date of the class source file. If the source has been modified more recently, it is recompiled and the new bytecode file is loaded. `java` repeats this procedure until all the classes are correctly compiled and loaded.

The interpreter can determine whether a class is legitimate through the mechanism of verification. Verification ensures that the bytecodes being interpreted do not violate any language constraints.

`java_g` is a non-optimized version of `java` suitable for use with debuggers like `jdb`.

Optins

-debug

Allows the Java debugger-`jdb` -to attach itself to this `java` session. When `-debug` is specified on the command line `java` displays a password which must be used when starting the debugging session.

-cs, -checksource

When a compiled class is loaded, this option causes the modification time of the class bytecode file to be compared to that of the class source file. If the source has been modified more recently, it is recompiled and the new bytecode file is loaded.

-classpath path

Specifies the path `java` uses to look up classes. Overrides the default or the `CLASSPATH` environment variable if it is set. Directories are separated by colons. Thus the general format for a path is:

```
.;<your_path>
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

-mx x

Sets the maximum size of the memory allocation pool (the garbage collected heap) to `x`. The default is 16 megabytes of memory. `x` must be > 1000 bytes.

By default, `x` is measured in bytes. You can specify `x` in either kilobytes or megabytes by appending the letter `k` for kilobytes or the letter `m` for megabytes.

-ms x

Sets the startup size of the memory allocation pool (the garbage collected heap) to `x`. The default is 1 megabyte of memory. `x` must be > 1000 bytes.

By default, `x` is measured in bytes. You can specify `x` in either kilobytes or megabytes by appending the letter `k` for kilobytes or the letter `m` for megabytes.

`-noasyncgc`

Turns off asynchronous garbage collection. When activated, no garbage collection takes place unless it is explicitly called or the program runs out of memory. Normally, garbage collection runs as an asynchronous thread in parallel with other threads.

`-ss x`

Each Java thread has two stacks: one for Java code and one for C code. The `-ss` option sets the maximum stack size that can be used by C code in a thread to `x`. Every thread that is spawned during the execution of the program passed to `java` has `x` as its C stack size. The default units for `x` are bytes. `x` must be > 1000 bytes.

You can modify the meaning of `x` by appending either the letter `k` for kilobytes or the letter `m` for megabytes. The default stack size is 128 kilobytes (`-ss 128k`).

`-oss x`

Each Java thread has two stacks: one for Java code and one for C code. The `-oss` option sets the maximum stack size that can be used by Java code in a thread to `x`. Every thread that is spawned during the execution of the program passed to `java` has `x` as its Java stack size. The default units for `x` are bytes. `x` must be > 1000 bytes.

You can modify the meaning of `x` by appending either the letter `k` for kilobytes or the letter `m` for megabytes. The default stack size is 400 kilobytes (`-oss 400k`).

`-t`

Prints a trace of the instructions executed (`java_g` only).

`-v`, `-verbose`

Causes `java` to print a message to `stdout` each time a class file is loaded.

`-verify`

Runs the verifier on all code.

`-verifyremote`

Runs the verifier on all code that is loaded into the system via a classloader. `verifyremote` is the default for the interpreter.

`-noverify`

Turns verification off.

`-verbosegc`

Causes the garbage collector to print out messages whenever it frees memory.

`-DpropertyName=newValue`

Redefines a property value. `propertyName` is the name of the property whose value you want to change and `newValue` is the value to change it to. For example, this command line:

```
java -Dawt.button.color=green ...
```

sets the value of the property `awt.button.color` to `green`. `java` accepts any number of `-D` options on the command line.

jdb-The Java Debugger

`jdb` helps you find and fix bugs in Java language programs.

Synopsis

```
jdb [ options ]
```

Description

The Java Debugger, `jdb`, is a `dbx`-like command-line debugger for Java classes. It uses the Java Debugger to provide inspection and debugging of a local or remote Java interpreter.

Starting a `jdb` Session

Like `dbx`, there are two ways `jdb` can be used for debugging. The most frequently used way is have `jdb` start the Java interpreter with the class to be debugged. This is done by substituting the command `jdb` for `java` in the command line. For example, to start HotJava under `jdb`, you use the following:

```
C:\> jdb browser.hotjava
```

or

```
C:\> jdb -classpath %INSTALL_DIR%\classes -ms4m
browser.hotjava
```

When started this way, `jdb` invokes a second Java interpreter with any specified parameters, loads the specified class, and stops before executing that class's first instruction.

The second way to use `jdb` is by attaching it to a Java interpreter that is already running. For security reasons, Java interpreters can only be debugged if they have been started with the `-debug` option. When started with the `-debug` option, the Java interpreter prints out a password for `jdb`'s use.

To attach `jdb` to a running Java interpreter (once the session password is known), invoke it as follows:

```
C:\> jdb -host <hostname> -password <password>
```

Basic `jdb` Commands

The following is a list of the basic `jdb` commands. The Java Debugger supports other commands which you can list using `jdb`'s `help` command.

Note
To browse local (stack) variables, the class must have been compiled with the `-g` option.

help

The most important `jdb` command, `help` displays the list of recognized commands with a brief description.

print

Browses Java objects. The `print` command calls the object's `toString()` method, so it will be formatted differently depending on its class.

Classes are specified by either their object ID or by name. If a class is already loaded, a substring can be used, such as `Thread` for `java.lang.Thread`. If a class isn't loaded, its full name must be specified, and the class will be loaded as a side effect. This is needed to set breakpoints in referenced classes before an applet runs.

`print` supports Java expressions, such as print `MyClass.clsVar`. Method invocation will not be supported in the 1.0 release, however, as the compiler needs to be enhanced first.

dump

Dumps an object's instance variables. Objects are specified by their object ID (a hexadecimal integer).

Classes are specified by either their object ID or by name. If a class is already loaded, a substring can be used, such as `Thread` for `java.lang.Thread`. If a class isn't loaded, its full name must be specified, and the class will be loaded as a side effect. This is needed to set breakpoints in referenced classes before an applet runs.

The `dump` command supports Java expressions such as `dump 0x12345678.myCache[3].foo`. Method invocation will not be supported in the 1.0 release, however, because the compiler needs to be enhanced first.

threads

Lists the current threads. This lists all threads in the default threadgroup, which is normally the first non-system group. (The `threadgroups` command lists all threadgroups.) Threads are referenced by their object ID, or if they are in the default thread group, with the form `t@<index>`, such as `t@3`.

where

Dumps the stack of either a specified thread, or the current thread (which is set with the `thread` command). If that thread is suspended (either because it's at a breakpoint or via the suspend command), local (stack) and instance variables can be browsed with the print and dump commands. The up and down commands select which stack frame is current.

Breakpoints

Breakpoints are set in `jdb` in classes, such as `"stop at MyClass:45"`. The source file line number must be specified, or the name of the method (the breakpoint will then be set at the first instruction of that method). The `clear` command removes breakpoints using a similar syntax, while the `cont` command continues execution.

Single-stepping is not currently implemented, but is hoped to be available for version 1.0.

Exceptions

When an exception occurs for which there isn't a `catch` statement anywhere up a Java program's stack, the Java runtime normally dumps an exception trace and exits. When running under `jdb`, however, that exception is treated as a non-recoverable breakpoint, and `jdb` stops at the offending instruction. If that class was compiled with the `-g` option, instance and local variables can be printed to determine the cause of the exception.

Specific exceptions may be optionally debugged with the `catch` command, for example:

```
catch FileNotFoundException
```

or

```
catch mypackage.BigTroubleException.
```

The Java debugging facility keeps a list of these exceptions, and when one is thrown, it is treated as if a breakpoint was set on the instruction which caused the exception. The `ignore` command removes exception classes from this list.

Note
The `ignore` command does not cause the Java interpreter to ignore specific exceptions, only the debugger.

Options

When you use `jdb` in place of the Java interpreter on the command line, `jdb` accepts the same options as the `java` command.

When you use `jdb` to attach to a running Java interpreter session, `jdb` accepts these options:

```
-host <hostname>
```

Sets the name of the host machine on which the interpreter session to attach to is running.

```
-password <password>
```

Logs in to the active interpreter session. This is the password the Java interpreter prints out when invoked with the `-debug` option.

`javah`-C Header and Stub File Generator

`javah` produces C header files and C source files from a Java class. These files provide the connective glue that allows your Java and C code to interact.

Synopsis

```
javah [ options ] classname. . .
javah_g [ options ] classname. . .
```

Description

`javah` generates C header and source files that are needed to implement native methods. The generated header and source files are used by C programs to reference an object's instance variables from native source code. The `.h` file contains a struct definition whose layout parallels the layout of the corresponding class. The fields in the `struct` correspond to instance variables in the class.

The name of the header file and the structure declared within it are derived from the name of the class. If the class passed to `javah` is inside a package, the package name is prepended to both the header filename and the structure name. Underscores (_) are used as name delimiters.

By default `javah` creates a header file for each class listed on the command line and puts the files in the current directory. Use the `-stubs` option to create source files. Use the `-o` option to concatenate the results for all listed classes into a single file.

`javah_g` is a non-optimized version of `javah` suitable for use with debuggers like `jdb`.

Options

`-o outputfile`

Concatenates the resulting header or source files for all the classes listed on the command line into `outputfile`.

`-d directory`

Sets the directory where `javah` saves the header files or the stub files.

`-td directory`

Sets the directory where `javah` stores temporary files. By default, `javah` stores temporary files in the directory specified by the `%TEMP%` environment variable. If `%TEMP%` is unspecified, then `javah` checks

for a %TMP% environment variable. And finally, if %TMP% is unspecified, javah creates the directory C:\tmp and stores the files there.

-stubs

Causes javah to generate C declarations from the Java object file.

-verbose

Causes javah to print a message to stdout concerning the status of the generated files.

-classpath path

Specifies the path javah uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by semicolons. Thus the general format for path is:

```
.;<your_path>
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

javap-The Java Class File Disassembler

Disassembles class files.

Synopsis

```
javap [ options ] class. . .
```

Description

The javap command disassembles a class file. Its output depends on the options used. If no options are used, javap prints out the public fields and methods of the classes passed to it. javap prints its output to stdout. For example, compile the following class declaration:

```
class C {
    static int a = 1;
    static int b = 2;
    static {
        System.out.println(a);
    }
    static {
        a++;
        b = 7;
        System.out.println(a);
        System.out.println(b);
    }
```

```
        static {
            System.out.println(b);
        }
        public static void main(String args[]) {
            C c = new C();
        }
    }
```

When the resulting class C is passed to `javap` using no options the following output results:

Compiled from `C:\users\dac\C.java`

```
    private class C extends java\lang\Object {
              static int a;
              static int b;
              public static void main(java\lang\String []);
              public C();
              static void ();
          }
```

Options

`-l`

Prints out line and local variable tables.

`-p`

Prints out the private and protected methods and fields of the class in addition to the public ones.

`-c`

Prints out disassembled code, (the instructions that comprise the Java bytecodes, for each of the methods in the class). For example, passing class C to `javap` using the `-c` flag results in the following output:

```
    Compiled from C:\users\dac\C.java
                  private class C extends java\lang\Object {
                      static int a;
                      static int b;
                      public static void main(java\lang\String
    []);
                      public C();
                      static void ();

                  Method void main(java\lang\String [])
                     0 new #4
                     3 invokenonvirtual #9 ()V>
                     6 return
```

```
Method C()
   0 aload_0 0
   1 invokenonvirtual #10 ()V>
   4 return

Method void ()
   0 iconst_1
   1 putstatic #7
   4 getstatic #6
   7 getstatic #7
  10 invokevirtual #8
  13 getstatic #7
  16 iconst_1
  17 iadd
  18 putstatic #7
  21 bipush 7
  23 putstatic #5
  26 getstatic #6
  29 getstatic #7
  32 invokevirtual #8
  35 getstatic #6
  38 getstatic #5
  41 invokevirtual #8
  44 iconst_2
  45 putstatic #5
  48 getstatic #6
  51 getstatic #5
  54 invokevirtual #8
  57 return
}
```

`-classpath path`

Specifies the path `javap` uses to look up classes. Overrides the default or the `CLASSPATH` environment variable if it is set. Directories are separated by semicolons. Thus the general format for path is:

```
.;your_path
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

javadoc-The Java API Documentation Generator

Generates API documentation from source files.

Synopsis

```
javadoc [ options ] package | filename.java...
```

Description

`javadoc` parses the declarations and doc comments in Java source files and formats the public API into a set of HTML pages. As an argument to `javadoc` you can pass in either a package name or a list of Java source files.

Within doc comments, `javadoc` supports the use of special doc tags to augment the API documentation. `javadoc` also supports standard HTML within doc comments. This is useful for code samples and for formatting text.

The package specified on the command line must be in your `CLASSPATH`. Note that `javadoc` uses `.java` files, not `.class` files.

`javadoc` reformats and displays all public and protected declarations for

- Classes and interfaces
- Methods
- Variables

Doc Comments

Java source files can include doc comments. Doc comments begin with `/**` and indicate text to be included automatically in generated documentation.

Standard HTML

You can embed standard HTML tags within a doc comment. However, don't use tags heading tags like `<h1>` or `<hr>`, because `javadoc` creates an entire structured document and these structural tags interfere with the formatting of the generated document.

javadoc Tags

`javadoc` parses special tags that are recognized when they are embedded within a Java doc comment. These doc tags enable you to autogenerate a complete, well-formatted API from your source code. The tags start with an `@`.

Tags must start at the beginning of a line. Keep tags with the same name together within a doc comment. For example, put all your `@author` tags together so `javadoc` can tell where the list ends.

Class Documentation Tags

`@see classname`
Adds a hyperlinked `See Also` entry to the class.

`@see fully-qualified-classname`
Adds a hyperlinked `See Also` entry to the class.

`@see fully-qualified-classname#method-name`
Adds a hyperlinked `See Also` entry to the method in the specified class.

`@version version-text`
Adds a `Version` entry.

`@author your-name`
Creates an `Author` entry. There can be multiple author tags.

An example of a class comment:

```
/**
  * A class representing a window on the screen.
  * For example:
  * <pre>
  *    Window win = new Window(parent);
  *    win.show();
  * </pre>
  *
  * @see            awt.BaseWindow
  * @see            awt.Button
  * @version        1.2 12 Dec 1994
  * @author         Sami Shaio
  */
 class Window extends BaseWindow {
        ...
 }
```

Variable Documentation Tags

In addition to HTML text, variable comments can contain only the `@see` tag (see above).

An example of a variable comment:

```
    /**
     * The X-coordinate of the window
     * @see window#1
     */
    int x = 1263732;
```

Method Documentation Tags

Can contain `@see` tags, as well as:

`@param parameter-name description...`
Adds a parameter to the `Parameters` section. The description may be continued on the next line.

`@return description...`
Adds a `Returns` section, which contains the description of the return value.

`@exception fully-qualified-class-name description...`
Adds a `Throws` entry, which contains the name of the exception that may be thrown by the method. The exception is linked to its class documentation.

Here is an example of a method comment:

```
/**
  * Return the character at the specified index. An index
ranges
  * from <tt>0</tt> to <tt>length() - 1</tt>.
  * @param index      The index of the desired character
  * @return           The desired character
  * @exception        StringIndexOutOfRangeException When the
index
  *                   is not in the range <tt>0</tt>> to
<tt>length() - 1</tt>.
  */
 public char charAt(int index) {
     ...
 }
```

Options

`-classpath path`

Specifies the path `javadoc` uses to look up the `.java` files. Overrides the default or the `CLASSPATH` environment variable, if it is set. Directories are separated by semicolons, for example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

`-d directory`

Specifies the directory where `javadoc` stores the generated HTML files. For example:

```
javadoc -d C:\usrs\dac\public_html\doc java.lang
```

`-verbose`

Causes the compiler and linker to print out messages about what source files are being compiled and what object files are being loaded.

appletviewer-The Java Applet Viewer

The `appletviewer` command allows you to run applets outside of the context of a World Wide Web browser.

Synopsis

```
appletviewer [ options ] urls ...
```

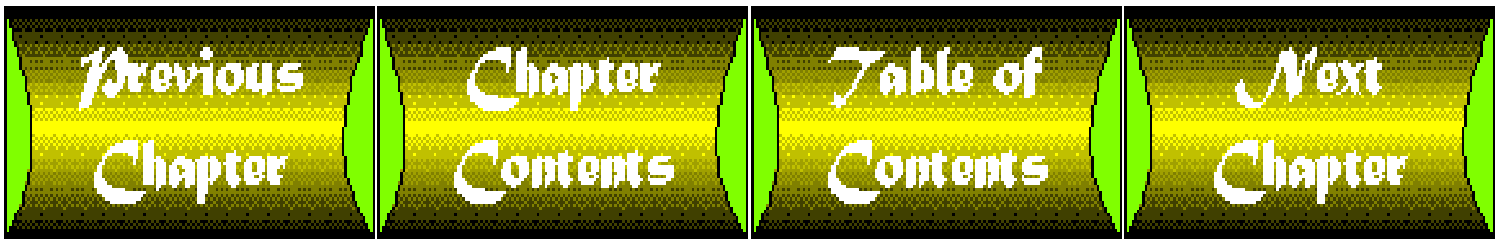
Description

The `appletviewer` command connects to the document(s) or resource(s) designated by `urls` and displays each applet referenced by that document in its own window. Note: if the document(s) referred to by `urls` does not reference any applets with the `APPLET` tag, `appletviewer` does nothing.

Options

`-debug`

Starts the applet viewer in the Java debugger-`jdb`-thus allowing you to debug the applets in the document.



---

appendix C

Java API Reference

CONTENTS

- Reserved Words
- Comments
- Literals
- Variable Declaration
- Variable Assignment
- Operators
- Objects
- Arrays
- Loops and Conditionals
- Class Definitions
- Method and Constructor Definitions
- Packages, Interfaces, and Importing
- Exceptions and Guarding

This appendix contains a summary or quick reference for the Java language, as described in this book.

Note
This is not a grammar overview, nor is it a technical overview of the language itself. It's a quick reference to be used after you already know the basics of how the language works. If you need a technical description of the language, your best bet is to visit the Java Web site (`http://java.sun.com`) and download the actual specification, which includes a full BNF grammar.

Language keywords and symbols are shown in a monospace font. Arguments and other parts to be substituted are in italic monospace.

Optional parts are indicated by brackets (except in the array syntax section). If there are several options that are mutually exclusive, they are shown separated by pipes (`|`) like this:

```
[ public | private | protected ] type varname
```

Reserved Words

The following words are reserved for use by the Java language itself (some of them are reserved but not currently used). You cannot use these terms to refer to classes, methods, or variable names:

`abstract`	do	import	public	try
`boolean`	double	instanceof	return	void
`break`	else	int	short	volatile
`byte`	extends	interface	static	while
`case`	final	long	super	
`catch`	finally	native	switch	
`char`	float	new	synchronized	
`class`	for	null	this	
`const`	goto	package	throw	
`continue`	if	private	throws	
`default`	implements	protected	transient	

Comments

```
/* this is the format of a multiline comment */
// this is a single-line comment
/** Javadoc comment */
```

Literals

`number`	Type int
`number[l | L]`	Type long
`0xhex`	Hex integer
`0Xhex`	Hex integer
`0octal`	Octal integer
`[ number ].number`	Type double
`number[ f | f]`	Type float
`number[ d | D]`	Type double
`[ + | - ] number`	Signed
`numberenumber`	Exponent
`numberEnumber`	Exponent
`'character'`	Single character

`"characters"`	String
`""`	Empty string
`\b`	Backspace
`\t`	Tab
`\n`	Line feed
`\f`	Form feed
`\r`	Carriage return
`\"`	Double quote
`\'`	Single quote
`\\`	Backslash
`\uNNNN`	Unicode escape (NNNN is hex)
`true`	Boolean
`false`	Boolean

Variable Declaration

`[ byte | short | int | long ]` *`varname`*	Integer (pick one type)
`[ float | double ]` *`varname`*	Floats (pick one type)
`char` *`varname`*	Characters
`boolean` *`varname`*	Boolean
`classname varname`	Class types
`type` *`varname, varname, varname`*	Multiple variables

The following options are available only for class and instance variables. Any of these options can be used with a variable declaration:

`[ static ] variableDeclaration`	Class variable
`[ final ] variableDeclaration`	Constants
`[ public | private | protected ] variableDeclaration`	Access control

Variable Assignment

`variable = value`	Assignment
`variable++`	Postfix Increment
`++variable`	Prefix Increment

`variable--`	Postfix Decrement
`--variable`	Prefix Decrement
`variable += value`	Add and assign
`variable -= value`	Subtract and assign
`variable *= value`	Multiply and assign
`variable /= value`	Divide and assign
`variable %= value`	Modulus and assign
`variable &= value`	AND and assign
`variable | = value`	OR and assign
`variable ^= value`	XOR and assign
`variable <<= value`	Left-shift and assign
`variable >>= value`	Right-shift and assign
`variable <<<= value`	Zero-fill, right-shift, and assign

Operators

`arg + arg`	Addition
`arg - arg`	Subtraction
`arg * arg`	Multiplication
`arg / arg`	Division
`arg % arg`	Modulus
`arg < arg`	Less than
`arg > arg`	Greater than
`arg <= arg`	Less than or equal to
`arg >= arg`	Greater than or equal to
`arg == arg`	Equal
`arg != arg`	Not equal
`arg && arg`	Logical AND
`arg || arg`	Logical OR
`! arg`	Logical NOT
`arg & arg`	AND
`arg | arg`	OR
`arg ^ arg`	XOR
`arg << arg`	Left-shift
`arg >> arg`	Right-shift

`arg >>> arg`	Zero-fill right-shift
`~ arg`	Complement
`(type)thing`	Casting
`arg instanceof class`	Instance of
`test ? trueOp : falseOp`	Ternary (if) operator

Objects

`new class();`	Create new instance
`new class(arg,arg,arg...)`	New instance with parameters
`object.variable`	Instance variable
`object.classvar`	Class variable
`Class.classvar`	Class variable
`object.method()`	Instance method (no args)
`object.method(arg,arg,arg...)`	Instance method
`object.classmethod()`	Class method (no args)
`object.classmethod(arg,arg,arg...)`	Class method
`Class.classmethod()`	Class method (no args)
`Class.classmethod(arg,arg,arg...)`	Class method

Arrays

Note
The brackets in this section are parts of the array creation or access statements. They do not denote optional parts as they do in other parts of this appendix.

`type varname[]`	Array variable
`type[] varname`	Array variable
`new type[numElements]`	New array object
`array[index]`	Element access
`array.length`	Length of array

Loops and Conditionals

`if ( test ) block` `if ( test ) block`	Conditional
`else block`	Conditional with else

`switch (test) {`
`  case value : statements`
`  case value : statements`
 	
`default : statement`
`}`
	switch (only with integer or char types)

`for (initializer; test; change ) block`	for loop
`while ( test ) block`	while loop
`do block` `while (test)`	do loop
`break [ label ]` `continue [ label ]`	break from loop or switch continue loops
`label:`	Labeled loops

Class Definitions

`class classname block`	Simple class definition

Any of the following optional modifiers can be added to the class definition:

`[ final ] class` *`classname block`*	No subclasses
`[ abstract ] class` *`classname block`*	Cannot be instantiated
`[ public ] class` *`classname block`*	Accessible outside package
`class` *`classname`* `[ extends` *`Superclass`* `]` *`block`*	Define superclass
`class` *`classname`* `[ implements` *`interfaces`* `]` *`block`*	Implement one or more interfaces

Method and Constructor Definitions

The basic method looks like this, where *`returnType`* is a type name, class name, or `void`.

`returnType methodName() block`	Basic method
`returnType methodName(parameter, parameter, ...) block`	Method with parameters

Method parameters look like this:

```
type parameterName
```

Method variations can include any of the following optional keywords:

`[ abstract ] returnType methodName() block`	Abstract method
`[ static ] returnType methodName() block`	Class method
`[ native ] returnType methodName() block`	Native method
`[ final ] returnType methodName() block`	final method
`[ synchronized ] returnType methodName() block`	Thread lock before executing
`[ public | private | protected ] returnType methodName()`	Access control

Constructors look like this:

`classname() block`	Basic constructor
`classname(parameter, parameter, parameter...) block`	Constructor with parameters
`[ public | private | protected] classname() block`	Access control

In the method/constructor body you can use these references and methods:

`this`	Refers to current object
`super`	Refers to superclass
`super.`*`methodName`*`()`	Calls a superclass's method
`this(...)`	Calls class's constructor
`super(...)`	Calls superclass's constructor
`return [ `*`value`*` ]`	Returns a value

Packages, Interfaces, and Importing

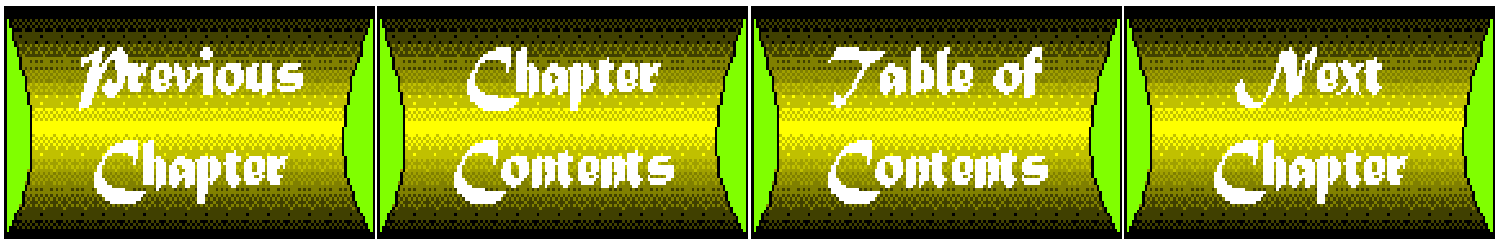
`import package.className`	Imports specific class name
`import package.*`	Imports all classes in package
`package packagename`	Classes in this file belong to this package

```
interface interfaceName [ extends anotherInterface ] block
[ public ] interface interfaceName block
[ abstract ] interface interfaceName block
```

Exceptions and Guarding

`synchronized ( object ) block`	Waits for lock on *object*
`try block`	Guarded statements
`catch ( exception ) block`	Executed if *exception* is thrown
`[ finally block ]`	Always executed

`try block`
`[ catch ( exception ) block ]`
`finally block`
	Same as previous example (can use optional catch or finally, but not both)



appendix D

Java Class Reference

CONTENTS

- java.lang
  - Interfaces
  - Classes
- java.util
  - Interfaces
  - Classes
- java.io
  - Interfaces
  - Classes
- java.net
  - Interfaces
  - Classes
- java.awt
  - Interfaces
  - Classes
- java.awt.image
  - Interfaces
  - Classes
- java.awt.peer
- java.applet
  - Inetrfaces
- Classes

This appendix provides a general overview of the classes available in the standard Java packages (that is, the classes that are guaranteed to be available in any Java implementation). This appendix is intended for general reference; for more specific information about each variable (its inheritance, variables, and methods), as well as the various exceptions for each package, see the API documentation from Sun at

http://java.sun.com.

java.lang

The java.lang package contains the classes and interfaces that are the core of the Java language.

Interfaces

Cloneable	Interface indicating that an object may be copied or cloned
Runnable	Methods for classes that want to run as threads

Classes

Boolean	Object wrapper for boolean values
Character	Object wrapper for char values
Class	Runtime representations of classes
ClassLoader	Abstract behavior for handling loading of classes
Compiler	System class that gives access to the Java compiler
Double	Object wrapper for double values
Float	Object wrapper for float values
Integer	Object wrapper for int values
Long	Object wrapper for long values
Math	Utility class for math operations
Number	Abstract superclass of all number classes (Integer, Float, and so on)
Object	Generic Object class, at top of inheritance hierarchy
Process	Abstract behavior for processes such as those spawned using methods in the System class
Runtime	Access to the Java runtime
SecurityManager	Abstract behavior for implementing security policies
String	Character strings
StringBuffer	Mutable strings
System	Access to Java's system-level behavior, provided in a platform-independent way
Thread	Methods for managing threads and classes that run in threads
ThreadDeath	Class of object thrown when a thread is asynchronously terminated

`ThreadGroup`	A group of threads
`Throwable`	Generic exception class; all objects thrown must be Throwables

java.util

The `java.util` package contains various utility classes and interfaces, including random numbers, system properties, and other useful classes.

Interfaces

`Enumeration`	Methods for enumerating sets of values
`Observer`	Methods for enabling classes to be `Observable` objects

Classes

`BitSet`	A set of bits
`Date`	The current system date, as well as methods for generating and parsing dates
`Dictionary`	An abstract class that maps between keys and values (superclass of HashTable)
`Hashtable`	A hash table
`Observable`	An abstract class for observable objects
`Properties`	A hash table that contains behavior for setting and retrieving persistent properties of the system or a class
`Random`	Utilities for generating random numbers
`Stack`	A stack (a last-in-first-out queue)
`StringTokenizer`	Utilities for splitting strings into individual "tokens"
`Vector`	A growable array of Objects

java.io

The `java.io` package provides input and output classes and interfaces for streams and files.

Interfaces

`DataInput`	Methods for reading machine-independent typed input streams
`DataOutput`	Methods for writing machine-independent typed output streams

`FilenameFilter`	Methods for filtering filenames

Classes

`BufferedInputStream`	A buffered input stream
`BufferedOutputStream`	A buffered output stream
`ByteArrayInputStream`	An input stream from a byte array
`ByteArrayOutputStream`	An output stream to a byte array
`DataInputStream`	Enables you to read primitive Java types (ints, chars, booleans, and so on) from a stream in a machine-independent way
`DataOutputStream`	Enables you to write primitive Java data types (ints, chars, booleans, and so on) to a stream in a machine-independent way
`File`	Represents a file on the host's file system
`FileDescriptor`	Holds onto the UNIX-like file descriptor of a file or socket
`FileInputStream`	An input stream from a file, constructed using a filename or descriptor
`FileOutputStream`	An output stream to a file, constructed using a filename or descriptor
`FilterInputStream`	Abstract class which provides a filter for input streams (and for adding stream functionality such as buffering)
`FilterOutputStream`	Abstract class which provides a filter for output streams (and for adding stream functionality such as buffering)
`InputStream`	An abstract class representing an input stream of bytes; the parent of all input streams in this package
`LineNumberInputStream`	An input stream that keeps track of line numbers
`OutputStream`	An abstract class representing an output stream of bytes; the parent of all output streams in this package
`PipedInputStream`	A piped input stream, which should be connected to a PipedOutputStream to be useful

`PipedOutputStream`	A piped output stream, which should be connected to a PipedInputStream to be useful (together they provide safe communication between threads)
`PrintStream`	An output stream for printing (used by System.out.println(...))
`PushbackInputStream`	An input stream with a 1-byte push back buffer
`RandomAccessFile`	Provides random access to a file, constructed from filenames, descriptors, or objects
`SequenceInputStream`	Converts a sequence of input streams into a single input stream
`StreamTokenizer`	Converts an input stream into a series of individual tokens
`StringBufferInputStream`	An input stream from a String object

`java.net`

The `java.net` package contains classes and interfaces for performing network operations, such as sockets and URLs.

Interfaces

`ContentHandlerFactory`	Methods for creating ContentHandler objects
`SocketImplFactory`	Methods for creating socket implementations (instance of the SocketImpl class)
`URLStreamHandlerFactory`	Methods for creating URLStreamHandler objects

Classes

`ContentHandler`	Abstract behavior for reading data from a URL connection and constructing the appropriate local object, based on MIME types
`DatagramPacket`	A datagram packet (UDP)
`DatagramSocket`	A datagram socket
`InetAddress`	An object representation of an Internet host (host name, IP address)
`ServerSocket`	A server-side socket
`Socket`	A socket

`SocketImpl`	An abstract class for specific socket implementations
`URL`	An object representation of a URL
`URLConnection`	Abstract behavior for a socket that can handle various Web-based protocols (http, ftp, and so on)
`URLEncoder`	Turns strings into x-www-form-urlencoded format
`URLStreamHandler`	Abstract class for managing streams to objects referenced by URLs

`java.awt`

The `java.awt` package contains the classes and interfaces that make up the Abstract Windowing Toolkit.

Interfaces

`LayoutManager`	Methods for laying out containers
`MenuContainer`	Methods for menu-related containers

Classes

`BorderLayout`	A layout manager for arranging items in border formation
`Button`	A UI pushbutton
`Canvas`	A canvas for drawing and performing other graphics oper-ations
`CardLayout`	A layout manager for HyperCard-like metaphors
`Checkbox`	A checkbox
`CheckboxGroup`	A group of exclusive checkboxes (radio buttons)
`CheckboxMenuItem`	A toggle menu item
`Choice`	A popup menu of choices
`Color`	An abstract representation of a color
`Component`	The abstract generic class for all UI components
`Container`	Abstract behavior for a component that can hold other components or containers
`Dialog`	A window for brief interactions with users
`Dimension`	An object representing width and height
`Event`	An object representing events caused by the system or based on user input

`FileDialog`	A dialog for getting filenames from the local file system
`FlowLayout`	A layout manager that lays out objects from left to right in rows
`Font`	An abstract representation of a font
`FontMetrics`	Abstract class for holding information about a specific font's character shapes and height and width information
`Frame`	A top-level window with a title
`Graphics`	Abstract behavior for representing a graphics context and for drawing and painting shapes and objects
`GridBagConstraints`	Constraints for components laid out using GridBagLayout
`GridBagLayout`	A layout manager that aligns components horizontally and vertically based on their values from GridBagConstraints
`GridLayout`	A layout manager with rows and columns; elements are added to each cell in the grid
`Image`	An abstract representation of a bitmap image
`Insets`	Distances from the outer border of the window; used to lay out components
`Label`	A text label for UI components
`List`	A scrolling list
`MediaTracker`	A way to keep track of the status of media objects being loaded over the Net
`Menu`	A menu, which can contain menu items and is a container on a menubar
`MenuBar`	A menubar (container for menus)
`MenuComponent`	The abstract superclass of all menu elements
`MenuItem`	An individual menu item
`Panel`	A container that is displayed
`Point`	An object representing a point (x and y coordinates)
`Polygon`	An object representing a set of points
`Rectangle`	An object representing a rectangle (x and y coordinates for the top corner, plus width and height)
`Scrollbar`	A UI scrollbar object
`TextArea`	A multiline, scrollable, editable text field
`TextComponent`	The superclass of all editable text components

`TextField`	A fixed-size editable text field
`Toolkit`	Abstract behavior for binding the abstract awt classes to a platform-specific toolkit implementation
`Window`	A top-level window, and the superclass of the Frame and Dialog classes

`java.awt.image`

The `java.awt.image` package is a subpackage of the awt that provides classes for managing bitmap images.

Interfaces

`ImageConsumer`	Methods for receiving an image created by an ImageProducer
`ImageObserver`	Methods to track the loading and construction of an image
`ImageProducer`	Methods for producing image data received by an ImageConsumer

Classes

`ColorModel`	An abstract class for managing color information for images
`CropImageFilter`	A filter for cropping images to a particular size
`DirectColorModel`	A specific color model for managing and translating pixel color values
`FilteredImageSource`	An ImageProducer that takes an image and an ImageFilter object and produces an image for an ImageConsumer
`ImageFilter`	A filter that takes image data from an ImageProducer, modifies it in some way, and hands it off to an ImageConsumer
`IndexColorModel`	A specific color model for managing and translating color values in a fixed-color map
`MemoryImageSource`	An image producer that gets its image from memory; used after constructing an image by hand
`PixelGrabber`	An ImageConsumer that retrieves a subset of the pixels in an image
`RGBImageFilter`	Abstract behavior for a filter that modifies the RGB values of pixels in RGB images

java.awt.peer

The `java.awt.peer` package is a subpackage of awt that provides the hidden platform-specific awt classes (for example, Motif, Macintosh, Windows 95) with platform-independent interfaces to implement. Thus, callers using these interfaces need not know which platform's window system these hidden awt classes are currently implementing.

Each class in the awt that inherits from either `Component` or `MenuComponent` has a corresponding peer class. Each of those classes is the name of the `Component` with `-Peer` added (for example, `ButtonPeer`, `DialogPeer`, and `WindowPeer`). Because each one provides similar behavior, they are not enumerated here.

java.applet

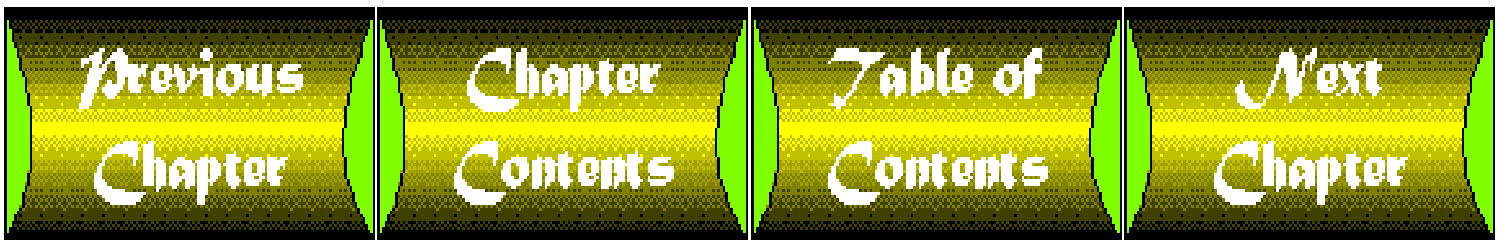
The `java.applet` package provides applet-specific behavior.

Interfaces

`AppletContext`	Methods to refer to the applet's context
`AppletStub`	Methods for implementing applet viewers
`AudioClip`	Methods for playing audio files

Classes

`Applet`	The base applet class



appendix E

Differences Between Java and C/C++

CONTENTS

- The Preprocessor
- Pointers
- Structures and Unions
- Functions
- Multiple Inheritance
- Strings
- The goto Statement
- Operator Overloading
- Automatic Coercions
- Variable Arguments
- Command-Line Arguments

It is no secret that the Java language is highly derived from the C and C++ languages. Because C++ is currently considered the language of choice for professional software developers, it's important to understand what aspects of C++ Java inherits. Of possibly even more importance are what aspects of C++ Java doesn't support. Because Java is an entirely new language, it was possible for the language architects to pick and choose which features from C++ to implement in Java and how.

The focus of this appendix is to point out the differences between Java and C++. If you are a C++ programmer, you will be able to appreciate the differences between Java and C++. Even if you don't have any C++ experience, you can gain some insight into the Java language by understanding what C++ discrepancies it clears up in its implementation. Because C++ backwardly supports C, many of the differences pointed out in this appendix refer to C++, but inherently apply to C as well.

The Preprocessor

All C/C++ compilers implement a stage of compilation known as the preprocessor. The C++ preprocessor basically performs an intelligent search and replace on identifiers that have been declared using the `#define` or `#typedef` directives. Although most advocates of C++ discourage the use of the preprocessor, which was inherited from C, it is still widely used by most C++ programmers. Most of the processor definitions in C++ are stored in header files, which complement the actual source code

files.

The problem with the preprocessor approach is that it provides an easy way for programmers to inadvertently add unnecessary complexity to a program. What happens is that many programmers using the `#define` and `#typedef` directives end up inventing their own sublanguage within the confines of a particular project. This results in other programmers having to go through the header files and sort out all the `#define` and `#typedef` information to understand a program, which makes code maintenance and reuse almost impossible. An additional problem with the preprocessor approach is that it is weak when it comes to type checking and validation.

Java does not have a preprocessor. It provides similar functionality (`#define`, `#typedef`, and so on) to that provided by the C++ preprocessor, but with far more control. Constant data members are used in place of the `#define` directive, and class definitions are used in lieu of the `#typedef` directive. The result is that Java source code is much more consistent and easier to read than C++ source code. Additionally, Java programs don't use header files; the Java compiler builds class definitions directly from the source code files, which contain both class definitions and method implementations.

Pointers

Most developers agree that the misuse of pointers causes the majority of bugs in C/C++ programming. Put simply, when you have pointers, you have the ability to trash memory. C++ programmers regularly use complex pointer arithmetic to create and maintain dynamic data structures. In return, C++ programmers spend a lot of time hunting down complex bugs caused by their complex pointer arithmetic.

The Java language does not support pointers. Java provides similar functionality by making heavy use of references. Java passes all arrays and objects by reference. This approach prevents common errors due to pointer mismanagement. It also makes programming easier in a lot of ways simply because the correct usage of pointers is easily misunderstood by all but the most seasoned programmers.

You may be thinking that the lack of pointers in Java will keep you from being able to implement many data structures, such as dynamic arrays. The reality is that any pointer task can be carried out just as easily and more reliably with objects and arrays of objects. You then benefit from the security provided by the Java runtime system; it performs boundary checking on all array indexing operations.

Structures and Unions

There are three types of complex data types in C++: classes, structures, and unions. Java only implements one of these data types: classes. Java forces programmers to use classes when the functionality of structures and unions is desired. Although this sounds like more work for the programmer, it actually ends up being more consistent, because classes can imitate structures and unions with ease. The Java designers really wanted to keep the language simple, so it only made sense to eliminate aspects of the language that overlapped.

Functions

In C, code is organized into functions, which are global subroutines accessible to a program. C++ added classes and in doing so provided class methods, which are functions that are connected to classes. C++ class methods are very similar to Java class methods. However, because C++ still supports C, there is nothing discouraging C++ programmers from using functions. This results in a mixture of function and method use that makes for confusing programs.

Java has no functions. Being a purer object-oriented language than C++, Java forces programmers to bundle all routines into class methods. There is no limitation imposed by forcing programmers to use methods instead of functions. As a matter of fact, implementing routines as methods encourages programmers to organize code better. Keep in mind that strictly speaking there is nothing wrong with the procedural approach of using functions; it just doesn't mix well with the object-oriented paradigm that defines the core of Java.

Multiple Inheritance

Multiple inheritance is a feature of C++ that allows you to derive a class from multiple parent classes. Although multiple inheritance is indeed powerful, it is complicated to use correctly and causes many problems otherwise. It is also very complicated to implement from the compiler perspective.

Java takes the high road and provides no direct support for multiple inheritance. You can implement functionality similar to multiple inheritance by using interfaces in Java. Java interfaces provide object method descriptions but contain no implementations.

Strings

C and C++ have no built-in support for text strings. The standard technique adopted among C and C++ programmers is that of using null-terminated arrays of characters to represent strings.

In Java, strings are implemented as first class objects (`String` and `StringBuffer`), meaning that they are at the core of the Java language. Java's implementation of strings as objects provides several advantages:

- The manner in which you create strings and access the elements of strings is consistent across all strings on all systems
- Because the Java string classes are defined as part of the Java language and not part of some extraneous extension, Java strings function predictably every time
- The Java string classes perform extensive runtime checking, which helps eliminate troublesome runtime errors

The `goto` Statement

The dreaded `goto` statement is pretty much a relic these days even in C and C++, but it is technically a legal part of the languages. The `goto` statement has historically been cited as the cause for messy, impossible to understand, and sometimes even impossible to predict code known as "spaghetti code." The primary usage of the `goto` statement has merely been as a convenience to substitute not thinking through an alternative, more structured branching technique.

For all these reasons and more, Java does not provide a `goto` statement. The Java language specifies `goto` as a keyword, but its usage is not supported. I suppose the Java designers wanted to eliminate the possibility of even using `goto` as an identifier! Not including `goto` in the Java language simplifies the language and helps eliminate the option of writing messy code.

Operator Overloading

Operator overloading, which is considered a prominent feature in C++, is not supported in Java. Although roughly the same functionality can be implemented by classes in Java, the convenience of operator overloading is still missing. However, in defense of Java, operator overloading can sometimes get very tricky. No doubt the Java developers decided not to support operator overloading to keep the Java language as simple as possible.

Automatic Coercions

Automatic coercion refers to the implicit casting of data types that sometimes occurs in C and C++. For example, in C++ you can assign a `float` value to an `int` variable, which can result in a loss of information. Java does not support C++ style automatic coercions. In Java, if a coercion will result in a loss of data, you must always explicitly cast the data element to the new type.

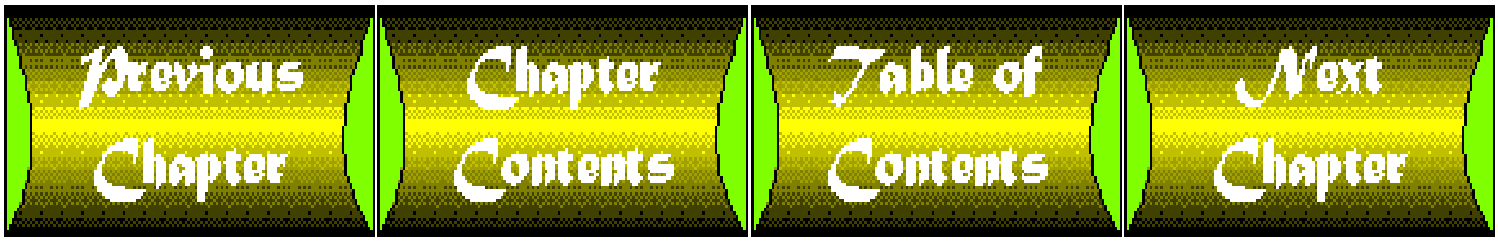
Variable Arguments

C and C++ let you declare functions, such as `printf`, that take a variable number of arguments. Although this is a convenient feature, it is impossible for the compiler to thoroughly type check the arguments, which means problems can arise at runtime without you knowing. Again Java takes the high road and doesn't support variable arguments at all.

Command-Line Arguments

The command-line arguments passed from the system into a Java program differ in a couple of ways from the command-line arguments passed into a C++ program. First, the number of parameters passed differs between the two languages. In C and C++, the system passes two arguments to a program: `argc` and `argv`. `argc` specifies the number of arguments stored in `argv`. `argv` is a pointer to an array of characters containing the actual arguments. In Java, the system passes a single value to a program: `args`. `args` is an array of `Strings` that contains the command-line arguments.

In C and C++, the command-line arguments passed into a program include the name used to invoke the program. This name always appears as the first argument and is rarely ever used. In Java, you already know the name of the program because it is the same name as the class, so there is no need to pass this information as a command-line argument. Therefore, the Java runtime system only passes the arguments following the name that invoked the program.



---

appendix F

Java Intranet Framework Reference

CONTENTS

- jif.awt
  - CalendarPanel
  - Effects
  - ImagePanel
  - JifCheckbox
  - JifDialog
  - JifLabel
  - JifPanel
  - JifPanePanel
  - JifTabPanel
  - JifTabSelector
  - JifTextArea
  - JifTextField
  - MessageBox
  - PickList
  - ResponseDialog
  - SimpleDBUI
  - StatusBar
- jif.jiflet
  - JifApplication
  - JifMessage
  - Jiflet
  - SimpleDBJiflet
- jif.log
  - DiskLog
  - Log

- ScreenLog
- jif.sql
  - CodeLookerUpper
  - Connector
  - DBConnector
  - DBRecord
  - MSQLConnector
  - MSSQLServerConnector
  - ODBcconnector
  - OracleConnector
  - OracleSequence
  - SequenceGenerator
  - SQLFactory
  - SybaseConnector
- jif.util
  - CallbackTimer
  - ConfigProperties
  - EventTimer
  - FileDate
  - TimeOut

This appendix is a reference guide for the Java Intranet Framework (JIF) classes that were developed in this book. Listed here are all the `public` and `protected` members of each JIF class.

In this appendix, you'll look at each package in JIF and then each class within each package. This appendix is meant as a reference for you to go back to instead of constantly looking at the source code.

jif.awt

The `jif.awt` package contains many classes that extend and enhance several of the stock Java awt classes. These include a calendar, a three-dimensional container, and a tabbed panel, just to mention a few.

CalendarPanel

This class is a `JifPanel` that adds calendar functionality. Each day in the calendar is a button. When pressed, the button sends an `ACTION_EVENT` event to its parent container. The selected date is passed

in the argument.

```
public class CalendarPanel extends JifPanel
{
//    Public constructors
    public CalendarPanel();
    public CalendarPanel( int style );
//    Public Methods
    public Date getSelectedDate();
    public void hilightButton( Button button, boolean onOff
);
}
```

Effects

This interface defines the constants that a class can use to provide three-dimensional effects. It contains no methods.

```
public interface Effects
{
//    Public Text Styles
    public final static int     TEXT_NORMAL = 0;
    public final static int     TEXT_LOWERED = 1;
    public final static int     TEXT_RAISED = 2;
    public final static int     TEXT_SHADOW = 3;
//    Public Text placements
    public final static int     CENTER = 0;
    public final static int     LEFT = 1;
    public final static int     RIGHT = 2;
//    Public 3D Looks
    public final static int     NONE = 0;
    public final static int     FLAT = 1;
    public final static int     GROOVED = 2;
    public final static int     LOWERED = 4;
    public final static int     ROUNDED = 8;
    public final static int     RAISED = 16;
    public final static int     RIDGED = 32;
    public final static int     CAPTION = 64;

}
```

ImagePanel

This class extends `JifPanel` to display a graphic image. You can specify the x and y offset with which the image is displayed within the rectangle.

```
public class ImagePanel extends JifPanel
{
```

```
    //    Protected class variables
        protected Image myImage;
        protected int offset;

    //    Public constructors
        public ImagePanel( String imageToUse, int offset )
            throws FileNotFoundException;
        public ImagePanel( String imageToUse )
            throws FileNotFoundException;

    //    Public methods
        public void addNotify();
        public void sizeSelf();
        public void paint( Graphics g );
    }
```

JifCheckbox

This class extends Java's `Checkbox` adding tab recognition and SQL code generation. The `JifCheckbox` produces a `Y` or `N` value in SQL. It is perfect for use with indicators.

```
    public class JifCheckbox extends Checkbox implements
    SQLFactory
    {
    //    Protected class variables
        protected String columnName = "";
        protected boolean dataChange = false;
        protected boolean initialized = false;
        protected boolean primaryKey = false;
        protected boolean isNumericData = false;

    //    Public constructors
        public JifCheckbox();
        public JifCheckbox( String columnName );
        public JifCheckbox( String columnName, boolean primaryKey
    );

    //    Public methods
        public void setColumnName( String columnName );
        public String getColumnName();
        public void setNumeric( boolean onOff );
        public void setPrimaryKey( boolean primaryKey );
        public boolean isPrimaryKey();
        public void setState( boolean state );
        public boolean didDataChange();
        public void reset();
```

```
        public String generateUpdateSQL( boolean addSet );
        public String generateInsertSQL( boolean addParen );
        public String getSQL( boolean forWhere );
    }
```

JifDialog

This class extends Java's `Dialog` class and adds the capability to center itself on the entire screen or within its parent's frame.

```
    public class JifDialog extends Dialog
    {
    //    Public constructors
        public JifDialog( Frame parent, boolean modal );
        public JifDialog( Frame parent, String title, boolean
    modal );

    //    Public methods
        public void center( boolean onScreen );
    }
```

JifLabel

This class extends `JifPanel` to implement a multiline label. The implementation is quite simple, thus the object has no added functionality.

```
    public class JifLabel extends JifPanel
    {
    //    Public constructor
        public JifLabel( String s );
    }
```

JifPanel

This class extends Java's `Panel` and adds many features-the coolest of which are some three-dimensional effects. These effects are defined by the `Effects` interface and implemented by this class. In addition, the `JifPanel` implements the `JifMessage` interface allowing it to send `JifMesssage` messages.

```
    public class JifPanel extends Panel implements Effects,
    JifMessage
    {
    //    Constants
        public final static int TAB_KEY;
        public final static int TEXT_OFFSET;

    //    Protected class variables
        protected int               textStyle = TEXT_NORMAL;
```

```
    protected int               textPlacement = CENTER;
    protected int               style = NONE;
    protected int               thickness = 2;
    protected int               myWidth;
    protected int               myHeight;
    protected String            text = null;
    protected boolean           skipLeft = false;
    protected boolean           skipTop = false;
    protected boolean           skipBottom = false;
    protected boolean           skipRight = false;
    protected int               currentPos = 0;

//    Public constructors
    public JifPanel();
    public JifPanel( int style );
    public JifPanel( String text );
    public JifPanel( int style, String text );
    public JifPanel( int style, int width, int height, String
text );
    public JifPanel( int style, int width, int height );

//    Public methods
    public void addNotify();
    public void addWithConstraints( Component comp, String
constraints );
    protected String buildComponentList( JifPanel me, Vector
valueList,
        Vector columnList, String whereClause );
    public void drawFrame( Graphics g );
    public void drawtext( Graphics g );
    protected Component findChild( JifPanel p, String name );
    protected int findNextComponent( int startPos );
    protected int findPreviousComponent( int startPos );
    public synchronized void focusForward();
    public synchronized void focusBackward();
    public String generateInsertSQL( String tableName );
    public String generateUpdateSQL( String tableName );
    public void getMetrics();
    public boolean isStyleSet( int check );
    public Dimension minimumSize();
    public void paint( Graphics g );
    public Dimension preferredSize();
    public void sendJifMessage( Event event, int msg );
    public void setChildValue( String name, String value );
    public void setChildValue( String name, int value );
    public void setFocus( Component target );
```

```
        public void setFont( Font f );
        public void setSkip( String which, boolean onOff );
        public void setStyle( int style );
        public void setText( String newText, int textStyle, int
    placement );
        public void setText( String newText );
        public void setTextPlacement( int placement );
        public void setTextStyle( int textStyle );
        public void setThickness( int thick );
    }
```

JifPanePanel

This class is used by the `JifTabPanel` to hold all of the panes. It contains a `CardLayout` to manage them.

```
    public class JifPanePanel extends JifPanel
    {
    //    Public constructors
        public JifPanePanel();

    //    Public methods
        public void addPane( String name, Component comp );
        public void selectPane( String paneName );
    }
```

JifTabPanel

This class implements a tabbed panel by extending the `JifPanel` and adding some pane management techniques. This class is the master class, and the two helper classes (`JifPanePanel` and `JifTabSelector`) implement the pane swapping and the tab selection.

```
    public class JifTabPanel extends JifPanel
    {
    //    Protected instance variables
        protected JifPanePanel panePanel = new JifPanePanel();
        protected JifTabSelector selector = new JifTabSelector(
    panePanel );

    //    Public constructor
        public JifTabPanel();

    //    Public methods
        public void addPane( String name, Component comp );
        public void selectPane( String name );
    }
```

JifTabSelector

This is a helper class for the `JifTabPanel` object implementing the actual tabs. It is responsible for drawing the tabs and communicating with the `JifTabPanel` about which tab has been clicked. This class extends `Canvas` so that it can draw custom tabs.

```
public class JifTabSelector extends Canvas
{
//    Public instance variables
    public boolean                fullWidthTabs = false;

//    Protected instance variables
    protected Vector              tabs = new Vector();
    protected int                 currentTab = 0;
    protected Font                normalFont = null;
    protected Font                selFont = null;
    protected int                 realWidth = 0, realHeight =
0;
    protected JifPanePanel        myPanePanel = null;

//    Public constructors
    public JifTabSelector( JifPanePanel jpp );
    public void addPane( String paneName );
    public Dimension minimumSize();
    public void paint( Graphics g );
    public Dimension preferredSize();
    public void selectPane( String paneName );
    public void selectPane( int which );
}
```

JifTextArea

This class extends Java's `TextArea`, adding tab recognition and SQL code generation. `JifTextArea`s produce a string value in SQL. In addition, you can restrict the data that is entered in a `JifTextArea` by using the `setStyle()` method. The types of input restrictions are as follows:

- `JifTextArea.ANY`-Any characters are allowed.
- `JifTextArea.LOWER`-All characters entered are lowercased.
- `JifTextArea.UPPER`-All characters entered are uppercased.
- `JifTextArea.NUMERIC`-Only numbers are allowed to be entered.

```
public class JifTextArea extends TextArea implements
SQLFactory
{

//    Constants
    public static final int        ANY = 0;
```

```
    public static final int        LOWER = 1;
    public static final int        UPPER = 2;
    public static final int        NUMERIC = 3;

//    Protected instance variables
    protected String            columnName = "";
    protected boolean            dataChange = false;
    protected boolean            initialized = false;
    protected boolean            primaryKey = false;
    protected boolean            isNumericData = false;
    protected boolean            isDateData = false;
    protected int                style = ANY;

//    Public constructors
    public JifTextArea( String s );
    public JifTextArea( String s, String columnName );
    public JifTextArea( String s, String columnName, boolean
primaryKey );
    public JifTextArea( int rows, int cols );
    public JifTextArea( int rows, int cols, String columnName
);
    public JifTextArea( int rows, int cols, String
columnName,
        boolean primaryKey );

//    Public methods
    public boolean didDataChange();
    public synchronized void disable();
    public synchronized void enable();
    public String generateInsertSQL( boolean addParen );
    public String generateUpdateSQL( boolean addSet );
    public String getColumnName();
    public String getSQL( boolean forWhere );
    public boolean isPrimaryKey();
    public void reset();
    public void setColumnName( String columnName );
    public void    setDate( boolean onOff );
    public void setNumeric( boolean onOff );
    public void setPrimaryKey( boolean primaryKey );
    public void setStyle( int style );
    public void setText( String text );
}
```

JifTextField

This class extends Java's `TextField` by adding tab recognition and SQL code generation. `JifTextField`s produce a string value in SQL. In addition, you can restrict the data that is entered in a `JifTextField` by using the `setStyle()` method. The types of input restrictions are as follows:

- `JifTextField.ANY`-Any characters are allowed.
- `JifTextField.LOWER`-All characters entered are lowercased.
- `JifTextField.UPPER`-All characters entered are uppercased.
- `JifTextField.NUMERIC`-Only numbers are allowed to be entered.

```
public class JifTextField extends TextField implements
SQLFactory
{

//    Constants
    public static final int        TAB_KEY = 9;
    public static final int        ANY = 0;
    public static final int        LOWER = 1;
    public static final int        UPPER = 2;
    public static final int        NUMERIC = 3;

//    Protected instance variables
    protected String            columnName = "";
    protected boolean           dataChange = false;
    protected boolean           initialized = false;
    protected boolean           primaryKey = false;
    protected boolean           isNumericData = false;
    protected boolean           isDateData = false;
    protected int               style = ANY;

//    Public Constructors
    public JifTextField()
    public JifTextField( String s )
    public JifTextField( String s, String columnName )
    public JifTextField( String s, String columnName, boolean
primaryKey )
    public JifTextField( int cols )
    public JifTextField( int cols, String columnName )
    public JifTextField( int cols, String columnName, boolean
primaryKey )

//    Public methods
    public void setStyle( int style )
    public void setColumnName( String columnName )
    public String getColumnName()
```

```
    public void setNumeric( boolean onOff )
    public void setDate( boolean onOff )
    public void setPrimaryKey( boolean primaryKey )
    public boolean isPrimaryKey()
    public boolean didDataChange()
    public void reset()
    public String generateUpdateSQL( boolean addSet )
    public String generateInsertSQL( boolean addParen )
    public String getSQL( boolean forWhere )
    public void setText( String text )
    public synchronized void enable()
    public synchronized void disable()
}
```

MessageBox

This class extends the `JifDialog` class to provide a popup message box. The class can be used to display information to the user, such as warnings, errors, or other informational items. You can optionally display an image in front of the text. There are four built-in images to choose from:

- `MessageBox.INFO`-Displays an informational mark.
- `MessageBox.EXCLAMATION`-Displays an exclamation signal.
- `MessageBox.STOP`-Displays a stop signal.
- `MessageBox.QUESTION`-Displays a question mark.

```
public class MessageBox extends JifDialog
{
//    Constants
    public static final String     INFO = "Information.gif";
    public static final String     EXCLAMATION =
"Exclamation.gif";
    public static final String     STOP = "Stop.gif";
    public static final String     QUESTION = "Question.gif";

//    Public constructors
    public MessageBox( Frame parent, String title, String
message,
        String iconToUse );
    public MessageBox( Frame parent, String title, String
message,
        String iconToUse, boolean addButtons );
    public MessageBox( Frame parent, String title, String
message );
}
```

PickList

This abstract class can be used to quickly build picklists of data. Simply supply an `init()` method that fills the `List myList` with data. You receive an `ACTION_EVENT` event notifying you of the user's selection.

```
public abstract class PickList extends JifDialog
{
//    Protected instance variables
    protected Button            okButton = new Button( "Ok"
);
    protected Button            cancelButton = new Button(
"Cancel" );
    protected List              myList = new List( 10, false
);

//    Public constructor
    public PickList( Frame daddy, String title )

//    Public method
    public abstract void init();
}
```

ResponseDialog

This class extends the `MessageBox` class to provide the same functionality with a configurable number of buttons. This allows you to produce "Yes, No, or Cancel?" type queries among others. Simply pass a string of button names separated by commas to the constructor.

```
public class ResponseDialog extends MessageBox
{
//    Protected instance variables
    protected Button[]          buttonList;
    protected int               buttonCount;

//    Public Constructor
    public ResponseDialog( Frame parent, String title, String
message,
        String buttons );
}
```

SimpleDBUI

This class implements functionality that can be used in simple database applications.

```
public abstract class SimpleDBUI extends JifPanel
{
//    Public instance variables
```

```
    public Button                saveButton = new Button(
"Save" );
    public Button                clearButton = new Button(
"Clear" );
    public Button                newButton = new Button(
"New" );
    public Button                deleteButton = new Button(
"Delete" );
    public Button                chooseButton = new Button(
"Choose" );
    public Button                closeButton = new Button(
"Close" );

//    Public constructor
    public SimpleDBUI( SimpleDBJiflet jiflet );

//    Public methods
    public SimpleDBJiflet getJiflet();
    public void setJiflet( SimpleDBJiflet jiflet );
    public abstract void moveToScreen();
    public abstract void clearScreen();
    public abstract void moveFromScreen();
}
```

StatusBar

This class implements a single-line, sunken status bar that can be found in almost every microcomputer software package.

```
public class StatusBar extends JifPanel
{
//    Public constructor
    public StatusBar();
//    Public Methods
    public void clear();
    public Dimension preferredSize();
}
```

jif.jiflet

The `jif.jiflet` package contains classes that pull together many of the other packages into one unit. This unit, the Jiflet, can be used as a base for almost any type of application. In addition, it provides many of the niceties of Java applets without the security restrictions.

JifApplication

This interface defines the methods that a Jiflet must provide.

```
public interface JifApplication
{
//    Public methods
    public void init();
    public void run();
    public void destroy();
}
```

JifMessage

This interface provides the standard `JifMessage` constants and the method needed to send them. Any class can implement this interface.

```
public interface JifMessage
{
//    Constants
    public static final int     NEW = 0;
    public static final int     CLEAR = 1;
    public static final int     SAVE = 2;
    public static final int     DELETE = 3;
    public static final int     CUT = 4;
    public static final int     COPY = 5;
    public static final int     PASTE = 6;
    public static final int     HELP_WINDOW = 7;
    public static final int     HELP_CONTEXT = 8;
    public static final int     HELP_ABOUT = 9;
    public static final int     HELP_HELP = 10;
    public static final int     DATA_chANGE = 11;
    public static final int     chOOSE = 12;
    public static final int     CLOSE = 13;

//    Public method
    public void sendJifMessage( Event event, int msg );
}
```

Jiflet

This class is the applet-like class that implements the model intranet application shell as described in Chapter 7, "A Model Intranet Application."

```
public abstract class Jiflet extends Frame implements
JifApplication, Log
{
```

```
//    Protected instance variables
    protected ConfigProperties       configProperties;
    protected String                 appName;
    protected boolean                appVerbosity = false;
    protected DiskLog                appLogFile;
    protected ScreenLog              defaultLog;
    protected StatusBar              myStatusBar = null;
    protected boolean                activeFlag = false;

//    Public constructors
    public Jiflet();
    public Jiflet( String title );
    public Jiflet( String title, String name, String args[]
);
    public Jiflet( String title, String name, String args[],
boolean verbosity );

//    Public methods
    public void run();
    public void destroy();
    public void setVerboseMode( boolean whichWay );
    public void verboseLog( char logLevel, String logEntry );
    public void verboseLog( String logEntry );
    public void errorLog( String logEntry );
    public void log( char logLevel, String logEntry );
    public void log( String logEntry );
    protected boolean handleMenuEvent( Event event, Object
arg );
    public boolean shutDown( int level );
    public boolean shutDown();
    public void suicide( Exception e, String logLine, int
level );
    public void suicide( String logLine );
    public void suicide( String logLine, int level );
    public void suicide( Exception e );
    public void suicide( Exception e, String logLine );
    public void center();
    public void enableStatusBar( String text );
    public void enableStatusBar();
    public void showStatus( String text );
    public void clearStatus();
    protected void setConnector( DBConnector aConnector );
    public DBConnector getConnector();
    public void startWait();
    public void endWait();
    public String getParameter( String key );
```

```
    public String getParameter( String key, String
defaultValue );
    public boolean canClose();
    public String getJifletInfo();
    public boolean isActive();
}
```

SimpleDBJiflet

This class, when used in conjunction with the `SimpleDBUI` class, provides a simple shell for database intranet applications.

```
public abstract class SimpleDBJiflet extends Jiflet
{
//    Protected instance variables
    protected Menu                fileMenu;
    protected Menu                helpMenu;

//    Public constructor
    public SimpleDBJiflet( String title, String name, String
args[] );

//    Public methods
    public void init();
    public void initializeMenus();
    public abstract void initializeUI();
    public void connectToDatabase();
    public void disconnectFromDatabase();
    protected boolean handleMenuEvent( Event event, Object
arg );
    public boolean saveRecord();
    public boolean deleteRecord();
    protected void setCopyright( String cpr );
    public boolean canClose();
    public void setDBRecord( DBRecord theRecord );
    public DBRecord getDBRecord();
    public void setUIPanel( SimpleDBUI panel );
    public SimpleDBUI getUIPanel();
}
```

jif.log

The `jif.log` package contains classes that write entries in common format to either the screen or a disk file.

DiskLog

This class writes commonly formatted entries into a disk file.

```
public class DiskLog extends RandomAccessFile implements Log
{
//    Protected instance variable
    protected Log               backupLog = null;

//    Public constructor
    public DiskLog( String name ) throws IOException;
    public DiskLog( String logName, String name ) throws
IOException;
    public DiskLog( String logDir, String logName, String
name )
       throws IOException;

//    Public methods
    public void setBackupLog( Log sl );
    static public String createLogFileName();
    static public String constructFileName( String path,
String name );
    public void log( char logLevel, String logEntry );
}
```

Log

This interface defines the constants and methods needed to implement a standard logging mechanism. Any class can implement this interface.

```
public interface Log
{
//    Constants
    public static final char     DEBUG = 'D';
    public static final char     INFO = 'I';
    public static final char     WARNING = 'W';
    public static final char     ERROR = 'E';
    public static final char     FATAL = 'F';

//    Public method
    public void log( char logLevel, String logEntry ) throws
IOException;
}
```

ScreenLog

This class provides a common log output to the screen. You can optionally send this information to a window instead of to the standard output (`stdout`).

```
public class ScreenLog extends Frame implements Log
{
//    Public constructors
    public ScreenLog();
    public ScreenLog( String name );
    public ScreenLog( String windowTitle, boolean popup );

//    Public method
    public void log( char logLevel, String logEntry );
}
```

jif.sql

The `jif.sql` package contains classes that implement SQL or JDBC specific functionality. These classes are either new or extend Java's `java.sql` packaged classes.

CodeLookerUpper

This class allows you to quickly look up a code value from a table. A code value can be any column from a row that has a single key value as its primary key. For instance, if you have an employee table with the employee number as the primary key, you can use this class to retrieve the employee's name. Look in the Employee Benefits sample application for a usage example.

```
public class CodeLookerUpper
{
//    Protected instance variables
    protected DBConnector       myConnection = null;
    protected String            mySQL = null;

//    Public constructor
    public CodeLookerUpper( DBConnector connector, String
tableName,
        String keyColumnName, String valueColumnName );

//    Public method
    public String lookupCode( int code_id );
}
```

Connector

This interface defines the methods required to become a JDBC/JIF database connector.

```
public interface Connector
{
//    Public methods
    public boolean connect( String user, String password,
String server );
    public boolean disconnect();
    public boolean connected();
    public String getConnectionURL();
}
```

DBConnector

This class implements the `Connector` interface and provides several of the methods. However, some remain abstract so that you can derive your own database connectors from them.

```
public abstract class DBConnector implements Connector
{
//    Protected instance variables
    protected Jiflet            myJiflet;
    protected Connection         myConnection;
    protected boolean            isConnected = false;
    protected Statement         myStatement;
    protected String            lastError;

//    Public constructors
    public DBConnector( Jiflet jiflet );
    public DBConnector();

//    Public methods
    public boolean connect( String user, String password,
String server );
    public boolean disconnect();
    public boolean connected();
    public Statement getStatement();
    public void errorLog( String logEntry );
    public void log( String logEntry );
    public boolean commit();
    public boolean rollback();
    public boolean close();
    public String getLastError();
}
```

DBRecord

This abstract class encapsulates a row of data from a table. It can be used as a base class for your own table data.

```
public abstract class DBRecord
{
//    Protected instance variables
    protected boolean            dataChange = false;
    protected boolean            isNewRecord = false;

//    Public constructors
    public DBRecord();
    public DBRecord( ResultSet rs );

//    Public methods
    public boolean parseResultSet( ResultSet rs );
    public abstract boolean update( DBConnector theConnector,
JifPanel ap );;
    public abstract boolean deleteRow( DBConnector
theConnector );;
    public boolean setDataChange( boolean onOff );
    public void clear();
    public boolean canSave();
    public boolean didDataChange();
    public void setNewStatus( boolean how );
    public boolean getNewStatus();
}
```

MSQLConnector

This class implements the remaining methods from `DBConnector` to provide access to an `MSQL` data source using JDBC.

```
public class MSQLConnector extends DBConnector
{
//    Protected instance variable
    protected String            dbUrl;

//    Public constructors
    public MSQLConnector( Jiflet jiflet, String host, int
port, String instance );
    public MSQLConnector( String host, int port, String
instance );

//    Public method
    public String getConnectionURL();
```

```
}
```

MSSQLServerConnector

This class implements the remaining methods from `DBConnector` to provide access to a Microsoft SQL Server data source using JDBC.

```
public class MSSQLServerConnector extends DBConnector
{
//    Public constructors
    public MSSQLServerConnector( Jiflet jiflet );
    public MSSQLServerConnector();

//    Public method
    public String getConnectionURL();
}
```

ODBcconnector

This class implements the remaining methods from `DBConnector` to provide access to an ODBC data source using JDBC.

```
public class ODBcconnector extends DBConnector
{
//    Public constructors
    public ODBcconnector( Jiflet jiflet, String dsInfo );
    public ODBcconnector( String dsInfo );

//    Public method
    public String getConnectionURL();
}
```

OracleConnector

This class implements the remaining methods from `DBConnector` to provide access to an Oracle data source using JDBC.

```
public class OracleConnector extends DBConnector
{
//    Public constructors
    public OracleConnector( Jiflet jiflet );
    public OracleConnector();

//    Public method
    public String getConnectionURL();
}
```

OracleSequence

This class encapsulates access to an Oracle sequence. A sequence is an automatic counter that is maintained by the database server. It can be used to generate employee IDs for instance.

```
public class OracleSequence
{
//    Public constructor
    public OracleSequence( OracleConnector connector, String
sequenceName );

//    Public method
    public int getNextValue();
    public int getCurrentValue();
    protected synchronized int getDBSequenceValue( boolean
currentVal );
}
```

SequenceGenerator

This class implements a mock database counter for databases that do not provide native sequences.

```
public class SequenceGenerator
{
//    Public constructors
    public SequenceGenerator( DBConnector connector, String
table,
        String column );
    public SequenceGenerator( DBConnector connector );

//    Public method
    public int getNextValue();
    public int getCurrentValue();
    protected synchronized void getDBSequenceValue();
}
```

SQLFactory

This interface defines the methods required for a class to provide SQL generation capabilities.

```
public interface SQLFactory
{
//    Public methods
    public String generateUpdateSQL( boolean addSet );
    public String generateInsertSQL( boolean addParen );
    public String getColumnName();
    public void setColumnName( String colName );
    public boolean isPrimaryKey();
```

```
        public String getSQL( boolean forWhere );
    }
```

SybaseConnector

This class implements the remaining methods from `DBConnector` to provide access to a Sybase data source using JDBC.

```
    public class SybaseConnector extends DBConnector
    {
    //    Public constructors
        public SybaseConnector( Jiflet jiflet );
        public SybaseConnector();

    //    Public method
        public String getConnectionURL();
    }
```

jif.util

The `jif.util` package contains classes that are simply utilitarian in nature. They provide functionality that can be used by almost any software project.

CallbackTimer

This class implements a timer that calls back the owner when the alarm goes off.

```
    public class CallbackTimer extends Thread
    {
    //    Public constructor
        public CallbackTimer( TimeOut target, int interval )

    //    Public method
        public void run()
    }
```

ConfigProperties

This class extends the Java `Properties` class to do two things. First it reads the application configuration file. Second, it merges any properties passed in at construction with the read in properties.

```
    public class ConfigProperties extends Properties
    {
    //    Protected instance variable
        protected Properties            argProperties = new
    Properties();
```

```
    //    Public constructor
        public ConfigProperties( String args[], String fileName
    );

    //    Public methods
        public int parseArguments( String args[] );
        public int parseConfigFile( String appdefaultConfigFile
    );
        protected void setProperties( Properties prop );
    }
```

EventTimer

This class implements a timer that generates an event to the owner when the alarm goes off.

```
    public class EventTimer extends Thread
    {
    //    Public constructor
        public EventTimer( Component target, int msInterval )

    //    Public method
        public void run()
    }
```

FileDate

This class is an all-purpose date formatting class. It allows you to convert the date into a variety of formats.

```
    public class FileDate extends Date
    {
    //    Constants
        public final static int        MDY = 0;
        public final static int        DMY = 1;
        public final static int        YMD = 2;
        public final static int        MMMDY = 3;
        public final static int        DMMMY = 4;
        public final static int        YMMMD = 5;
        public final static int        MDYYYY = 6;
        public final static int        DMYYYY = 7;
        public final static int        YYYYMD = 8;
        public final static int        MMMDYYYY = 9;
        public final static int        DMMMYYYY = 10;
        public final static int        YYYYMMMD = 11;

    //    Public constructors
        public FileDate();
```

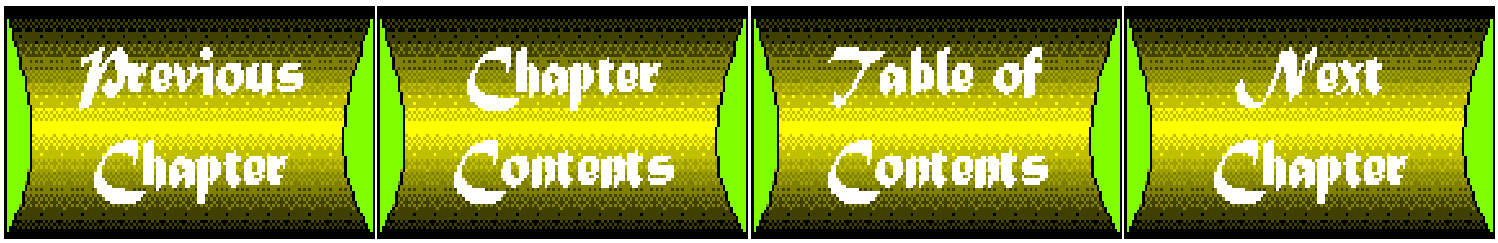
```
        public FileDate( java.util.Date date );
        public FileDate( java.sql.Date date );
        public FileDate( int y, int m, int d );

    //    Public methods
        public String toFileString();
        public static String toFileString( String s );
        public static String toFileString( java.util.Date d );
        public static String toFileString( java.sql.Date d );
        public String toOracleString();
        public static String toOracleString( String s );
        public static String toOracleString( java.util.Date d );
        public static String toOracleString( java.sql.Date d );
        public String toNormalString();
        public static String toNormalString( String s );
        public static String toNormalString( java.util.Date d );
        public static String toNormalString( java.sql.Date d );
        public static FileDate valueOf( String s ) throws
    IllegalArgumentException;
        public String formatDateToString();
        public String formatDateToString( String delimiter, int
    fmtOpt );
        public static String formatDateToString( String s );
        public static String formatDateToString( String s, String
    delimiter );
        public static String formatDateToString( String s, String
    delimiter,
            int fmtOpt );
        public static String formatDateToString( int y, int m,
    int d );
        public static String formatDateToString( int y, int m,
    int d,
            String delimiter );
        public static String formatDateToString( int y, int m,
    int d,
            String delimiter, int fmtOpt );
    }
```

TimeOut

This interface defines the method that a class must implement to receive callback notification of events when using the `CallbackTimer` class.

```
    public interface TimeOut
    {
    //    Public method
```

```
        public void timeOut( CallbackTimer timer );
    }
```



---

appendix G

What's on the CD-ROM

CONTENTS

- Windows Software
  - Java
  - mSQL
  - Servers
  - HTML Tools
  - Graphics, Video, and Sound Applications
  - Explorer
  - Utilities
- About Shareware

On the *Developing Intranet Applications with Java* CD-ROM, you will find all the sample files that have been presented in this book, along with a wealth of other applications and utilities.

Note
Please refer to the `readme.wri` file on the CD-ROM (Windows) for the latest listing of software.

Windows Software

Java

- Sun's Java Developer's Kit for Windows 95/NT, Version 1.0.2
- Sun's JDBC Java Database API v1.01
- Sun's JDBC-ODBC Bridge Driver
- Sample Java applets and scripts
- JFactory Java IDE
- JPad Java IDE
- JPad Pro Java IDE
- Javelin Java IDE

- Kawa Java IDE
- JDesigner Pro Java IDE
- Studeio J++ Java IDE

mSQL

- mSQL v1.0 by Hughes Technologies

Note
Mini SQL is copyright Hughes Technologies of Australia and is included iwth their permission. mSQL is not free software, any commercial use of the software requires the purchase of a commercial use license. More information on mSQL can be found on the Hughes Technologies web site at http://www.Hughes.com.au/

- mSQL kfor Java v 1.1.3
- Java Class Library JDBC Driver for mSQL v 0.94

Servers

- Apache HTTP Server
- CESRN HTTP Server (W3C httpd)

HTML Tools

- Microsoft Internet Assistants for Access, Excel, PowerPoint, Schedule+, and Word
- W3e HTML editor
- CSE 3310 HTML validator
- HotDog 32-bit HTML editor
- HoTMeTaL HTML editor
- HTMLed HTML editor
- HTML Assistant for Windows
- WebEdit Pro HTML editor
- Web Weaver HTML editor
- ImageGen
- InContext Spider HTML editor
- InContext WebAnalyzer Website analyzer

Graphics, Video, and Sound Applications

- Goldwave sound editor, player, and recorder
- MapThis imagemap utility
- Paint Shop Pro 3.12 graphics editor and graphic file format converter for Windows
- SnagIt screen capture utility
- ThumbsPlus image viewer and browser

Explorer

- Microsoft Internet Explorer Version 3.0

Utilities

- Microsoft Viewers for Excel, PowerPoint, and Word
- Adobe Acrobat viewer
- Microsoft PowerPoint Animation Player & Publisher
- WinZip for Windows NT/95
- WinZip Self-Extractor-a utility program that creates native Windows self-extracting ZIP files

About Shareware

Shareware is not free. Please read all documentation associated with a third-party product (usually contained within files named `readme.txt` or `license.txt`) and follow all guidelines.