Steal This Book!

Yes, you read that right. Steal this book. For free.

Nothing. Zero. Zilch. Nadda. Zip.

Undoubtedly you're asking yourself, "Why would he give away a book he probably spent six grueling months writing? Is he crazy?"

The answer…is yes. Sort of. I know that every day you're faced with hundreds of computer titles, and a lot of them don't deliver the value or the information you need. So here's the deal: I'll give you this book (or this chapter, if you've only downloaded part of the book) for free provided you do me a couple favors:

1. **Send this book to your friends:** No, not your manager. Your "with it" computer friends who are looking for the next Big Thing. JXTA is it. Trust me. They want to know about it.

2. **Send a link to the book's web site:** Maybe the book is too big to send. After all, not everyone can have a fibre optic Internet connection installed in their bedroom. The site, at **www.brendonwilson.com/projects/jxta**, provides chapter-sized PDFs for easy downloading by the bandwidth-challenged.

3. **Visit the book's web site:** Being a professional developer, you probably have Carpal Tunnel Syndrome and shudder at the idea of typing in example source code. Save yourself the trouble. Go to **www.brendonwilson.com/projects/jxta** and download the source code. And while you're there, why not download some of the chapters you're missing?

4. **Buy the book:** You knew there had to be a catch. Sure, the book's PDFs are free, but I'm hoping that enough of you like the book so much that you have to buy a copy. Either that, or none of you can stand to read the book from a screen (or, worse yet, print it all out <shudder>) and resort to paper to save what's left of your eyesight. The book is available at your local bookstore or from **Amazon.com** (at a **handsome discount**, I might add).

I now return to your regularly scheduled program: enjoy the book!

3

Introducing JXTA P2P Solutions

Now that you've gotten a basic introduction to the terminology, components, and issues of P2P networking, it's time to begin exploring the JXTA platform. This chapter introduces the logical structure and building blocks of JXTA, and demonstrates the capabilities of JXTA using the JXTA Shell application to provide interactive experimentation with the JXTA platform.

As outlined in Chapter 2, "P2P Concepts," a complete P2P solution provides mechanisms for a peer to do the following:

- Discover other peers and their services
- Publish its available services
- Exchange data with another peer
- Route messages to other peers
- Query peers for status information
- Group peers into peer groups

The JXTA platform defines a set of protocols designed to address the common functionality required to allow peers on a network to form robust pervasive networks, independent of the operating system, development language, and network transport employed by each peer.

Core JXTA Design Principles

While designing the protocol suite, the Project JXTA team made a conscious decision to design JXTA in a manner that would address the needs of the widest possible set of P2P applications. The design team stripped the protocols of any application-specific assumptions, focusing on the core P2P functionality that forms the foundation of all types of P2P applications.

One of the most important design choices was not to make assumptions about the type of operating system or development language employed by a peer. By making this choice, the Project JXTA team hoped to enable the largest number of potential participants in any JXTA-enabled P2P networking application. The JXTA Protocols Specification expressly states that network peers should be assumed to be any type of device, from the smallest embedded device to the largest supercomputer cluster.

In addition to eliminating barriers to participation based on operating system, computing platform, or programming language, JXTA makes no assumptions about the network transport mechanism, except for a requirement that JXTA must not require broadcast or multicast transport capabilities. JXTA assumes that peers and their resources might appear and disappear spontaneously from the network and that a peer's network location might change spontaneously or be masked by Network Address Translation (NAT) or firewall equipment.

Apart from the requirements specified by the JXTA Protocols Specification, the specification makes several important recommendations. In particular, the specification recommends that peers cache information to reduce network traffic and provide message routing to peers that are not directly connected to the network.

The JXTA Protocol Suite

Based on these design criteria and others documented in the Protocols Specification, the Project JXTA team designed a set of six protocols based on XML messages, shown in Figure 3.1.

Each of the JXTA protocols addresses exactly one fundamental aspect of P2P networking. Each protocol conversation is divided into a portion conducted by the local peer and another portion conducted by the remote peer. The local peer's half of the protocol is responsible for generating messages and sending them to the remote peer. The remote peer's half of the protocol is responsible for handling the incoming message and processing the message to perform a task.

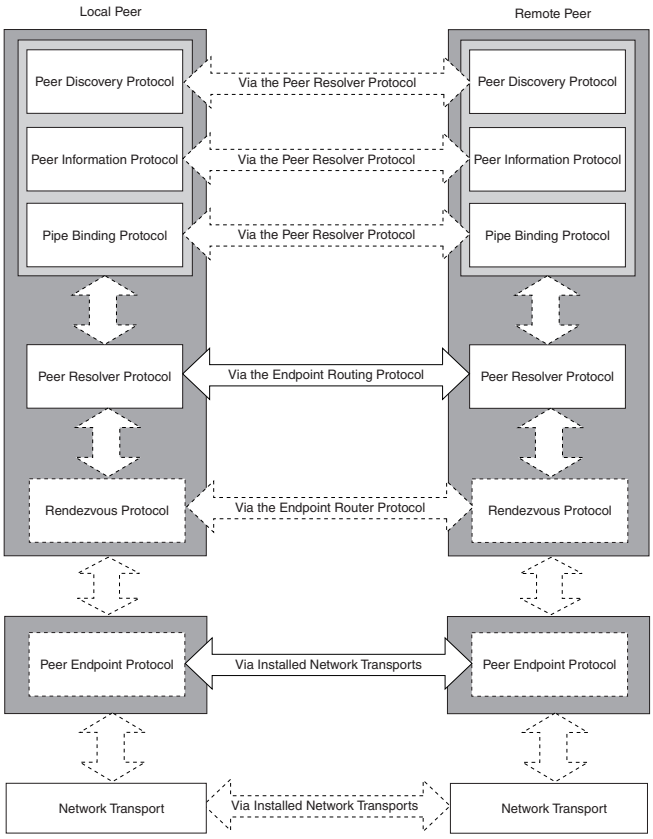

Figure 3.1 The JXTA protocol stack.

Each protocol is semi-independent of the others. A peer can elect to implement only a subset of the protocols to provide functionality, while relying on prespecified behavior to eliminate the need for a protocol. For example, a peer could rely on a preconfigured set of router peers and, therefore, would not require an implementation of the Endpoint Routing Protocol. However, the protocols aren't entirely independent of each other because each layer in the JXTA protocol stack depends on the layer below to provide connectivity to other peers. Although it would be possible to build an independent implementation of the Peer Discovery Protocol, it wouldn't be useful without an implementation of the Peer Resolver and Endpoint Routing Protocols to handle transporting its messages to remote peers.

Peers can even elect to implement only one half of a protocol to provide a peer optimized for one specific task. However, despite the allowance for partial implementations, the JXTA specification recommends that peers completely implement all the protocols.

The Logical Layers of JXTA

The JXTA platform can be broken into three layers, as shown in Figure 3.2.

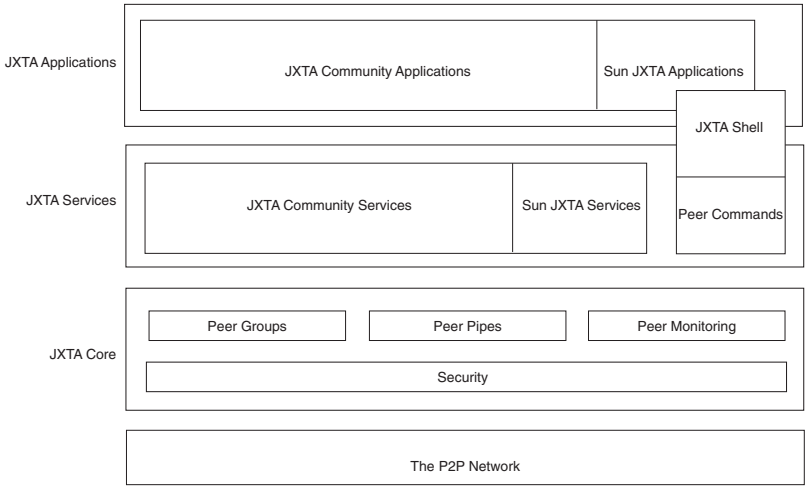

Figure 3.2 The JXTA three-layer architecture.

Each layer builds on the capabilities of the layer below, adding functionality and behavioral complexity.

The Core Layer

The core layer provides the elements that are absolutely essential to every P2P solution. Ideally, the elements of this layer are shared by all P2P solutions. These concepts were discussed in Chapter 2. The elements of the core layer are listed here:

- Peers
- Peer groups
- Network transport (pipes, endpoints, messages)
- Advertisements

- Entity naming (identifiers)
- Protocols (discovery, communication, monitoring)
- Security and authentication primitives

The core layer includes the six main protocols provided by JXTA. Although these protocols are implemented as services, they are located in the platform layer and are designated as core services to distinguish them from the service solutions of the services layer.

The core layer, as its name suggests, is the fundamental core of the JXTA solution. All other aspects of a JXTA P2P solution in the services or applications layers build on this layer to provide functionality.

The Services Layer

The services layer provides network services that are desirable but not necessarily a part of every P2P solution. These services implement functionality that might be incorporated into several different P2P applications, such as the following:

- Searching for resources on a peer
- Sharing documents from a peer
- Performing peer authentication

The services layer encompasses additional functionality that is being built by the JXTA community (open-source developers working with Project JXTA) in addition to services built by the Project JXTA team. Services built on top of the JXTA platform provide specific capabilities that are required by a variety of P2P applications and can be combined to form a complete P2P solution.

The Applications Layer

The applications layer builds on the capabilities of the services layer to provide the common P2P applications that we know, such as instant messaging. Because an application might encompass only a single service or aggregate several services, it's difficult sometimes to determine what constitutes an application and what constitutes a service.

Usually, the presence of some form of user interface indicates an application rather than a service. In the case of the JXTA Shell, most of the functionality is built on peer commands, simple services that accept command-line arguments from the JXTA Shell. The JXTA Shell itself is a service, providing only a minimal user interface, so the Shell is spread across the application/service boundary.

Applications include those P2P applications being built by the JXTA Community, as well as demonstration applications such as the JXTA Shell being built by the Project JXTA team.

XML: A Brief Introduction

All aspects of JXTA build on the eXtensible Markup Language (XML) to structure data as advertisements, messages, and protocols. XML is good choice for representing data for five reasons:

- **XML is language-neutral.** Any programming language capable of manipulating text strings is capable of parsing and formatting XML data.
- **XML is simple.** XML uses text markup to structure data in much the same way that HTML structures text documents for display in web browsers. The simplicity of XML makes it easier for developers to understand and debug.
- **XML is self-describing.** An XML document consists of data structured using metadata tags and attributes that describe the format of the data. Although XML supports the use of Document Type Definitions (DTDs) to provide a schema definition of a valid document, this is not a requirement for a well-formed XML document.
- **XML is extensible.** Unlike HTML, XML allows authors to define their own set of markup tags to structure data.
- **XML is a standard.** The World Wide Web Consortium (`www.w3.org`) is responsible for maintaining the XML standard, with industry and community input, and has been widely adopted in all areas of the computer industry.

To learn all you'll need to know about XML to understand JXTA, consider the simple example given in Listing 3.1.

Listing 3.1 **A Simple XML Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
  <Name>Erwin van der Koogh</Name>
  <Address>12 Lower Hatch Street</Address>
  <City>Dublin</City>
  <Country>Ireland</Country>
  <Phone>555-5555</Phone>
</Person>
```

Even if you've never seen XML, you probably recognize the example XML document as the contact information for a person named Erwin van der Koogh. From the example, you might guess at some of the rules of XML as follows:

- Each piece of information is encapsulated between a beginning and an end tag (such as `<Name></Name>`).
- The name of a tag specifies the type of content contained by the tags.
- Tags can be nested to form hierarchies that further structure the data in a meaningful way.

The only piece of information that might be puzzling is the first line. The first line specifies that the document is formatted using the rules set out by the XML 1.0 standard and that the document is encoded using UTF-8.

This example is straightforward. However, you might ask yourself, "What if Erwin has more than one phone number?" To further structure the data, an XML document can contain any number of the same type of element and can augment the elements with attributes that distinguish the elements, as shown in Listing 3.2.

Listing 3.2 **An Expanded XML Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
  <Name>Erwin van der Koogh</Name>
  <Address>12 Lower Hatch Street</Address>
  <City>Dublin</City>
  <Country>Ireland</Country>
  <Phone Type="Home">555-5555</Phone>
  <Phone Type="Work">555-1234</Phone>
</Person>F
```

The addition of the `Type` attribute to the `Phone` element tells you that `555-5555` corresponds to Erwin's home phone number and that `555-1234` corresponds to his work phone number.

More formal XML documents might use DTDs to define the following:

- Which tags are valid for a document
- How many times a specific tag might occur
- The order of the tags
- Required and optional attributes
- Default attribute values

When an XML document implements the rules specified by a DTD, the XML document is said to be valid. When an XML document doesn't use a DTD but otherwise follows the rules of XML, it is said to be well formed. For simple applications of XML, it is usually enough that documents are well formed, eliminating the overhead required to check that a document complies with a DTD.

That, in a nutshell, is about all the XML you need to know or understand to comprehend the XML used by JXTA. Although XML supports many other wonderful capabilities, understanding these capabilities isn't necessary to understand JXTA's use of XML. For more information on XML, see Appendix B, "Online Resources," for the location of the XML standard and other XML resources.

JXTA Advantages and Disadvantages

JXTA provides a far more abstract language for peer communication than previous P2P protocols, enabling a wider variety of services, devices, and network transports to be used in P2P networks. The employment of XML provides a standards-based format for structured data that is well understood, well supported, and easily adapted to a variety of transports. XML also has the advantage that it's a human-readable format, making it easy for developers to debug and comprehend. So far, JXTA seems to have done everything right. Well, maybe not.

One important element that JXTA does not attempt to address is how services (other than the core services) are invoked. Several standards exist for defining service invocation, such as the Web Services Description Language (WSDL), but none has been specifically chosen by the JXTA Protocols Specification. JXTA provides a generic framework for exchanging information between peers, so any mechanism, such as WSDL, could potentially be used via JXTA to exchange the information required to invoke services.

Several other arguments arise against the flexibility that the designers of JXTA infused throughout the JXTA Protocols Specification. Although JXTA's use of XML specifies all aspects of P2P communication for any generic P2P application, JXTA might not be suited to a specific standalone P2P application. In an individual application, the network overhead of XML messaging might be more trouble than it's worth, especially if the application developer has no intention of taking advantage of JXTA's capabilities to incorporate other P2P services into the application.

Critics of JXTA point out that the platform's abstraction of the network transport is another potential area of excess. If most P2P applications today rely on the Transport Control Protocol (TCP) to provide a network transport, why does JXTA go to such lengths to avoid tying the protocols to a specific network transport? Why not specify TCP as the assumed network transport and eliminate the overhead?

All these points highlight the need for developers to balance flexibility with performance when implementing their P2P applications. JXTA might not be the best or most efficient solution for implementing a particular P2P application. However, JXTA provides the most well-rounded platform for producing P2P applications that have the flexibility required to grow in the future. The capability to leverage other P2P services and enable widespread development of P2P communities is the core value of the JXTA platform.

How Is JXTA Different from Jini or .NET?

The promise of interconnecting any type of device over any type of network might sound familiar to followers of Sun's Jini technology. Although there are some similar goals, Jini relies exclusively on the Java platform for its functionality, whereas JXTA has no dependence on a particular programming language. Unlike JXTA, Jini uses a centralized server to locate services on the network and relies on Remote Method Invocation (RMI) and object serialization for communication with remote devices. JXTA relies on XML rather than object serialization to exchange structured data and discovers services across all peers on the P2P network.

The Web Services aspects of Microsoft's .NET platform are heavily infused with XML, but the use of XML alone doesn't make them comparable. Fundamentally, JXTA and .NET have completely different purposes, with .NET focusing more on the traditional client/server architecture of service delivery. Although .NET technology could form the foundation of a P2P application, creating a full P2P solution with .NET would require extra work on the part of the developer. Developing a P2P solution using .NET would require a developer to specify all the core P2P interactions, such as peer discovery. This solution would essentially involve recreating all the mechanisms that are already defined by the JXTA protocols.

Introducing the JXTA Shell

Rather than try to explain JXTA in the abstract, what better way to start to understand JXTA than seeing the technology in action? To do this, the remainder of this chapter guides you through using the JXTA Shell.

The JXTA Shell is a demo application built on top of the JXTA platform that allows users to experiment with the functionality made available through the Java reference implementation of the JXTA protocols. The JXTA Shell provides a UNIX-like command-line interface that allows the user to perform P2P operations by manipulating peers, peer groups, and pipes using simple commands.

Before You Install the JXTA Shell

To make the installation easier, you should already have a Java Run-Time Environment (JRE), version 1.3 or later, on your computer. To test whether you have a JRE already installed, go to the command prompt and type

```
java -version
```

If you have an existing JRE, you will see version information from the run-time of this form:

```
java version "1.3.1_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1_01)
Java HotSpot(TM) Client VM (build 1.3.1_01, mixed mode)
```

If you don't see this type of output, or if your version is lower that 1.3, you need to install a version 1.3 or higher JRE.

You can download a version of the JXTA Shell installer for most platforms with a standalone JRE included. However, if you intend to try the example code in this book's later chapters, you should install the Java 2 SDK (which includes a JRE) instead of a standalone JRE. The Java 2 SDK for most major platforms, including Solaris, Linux, and Windows, is available from `www.javasoft.com/j2se/`.

For developers using the Mac platform, the latest Java environment can be downloaded from `www.apple.com/java/` but is available only for the Mac OS X platform.

Obtaining and Installing the JXTA Shell

The JXTA Shell application can be obtained from either the Project JXTA web site as a set of prebuilt binaries or from the Project JXTA source control system as a set of source files.

To avoid the extra work required to build the JXTA Shell from source code, these experiments use the prebuilt JXTA Shell binaries that come with the JXTA demo applications. To download the JXTA demo installer that includes the JXTA Shell binaries, go to `download.jxta.org/easyinstall/install.html`.

Installing the JXTA demo applications also installs the latest stable build of the JXTA platform, packaged as a set of Java Archive (JAR) files. Unless you're interested in working with the latest experimental (and potentially unstable) version available from the Project JXTA CVS repository, these archives are all that's required to build new JXTA solutions in Java. The latest JXTA build at the time of writing was build 47b, built on January 25, 2002.

The installation procedure is slightly different for each operating system. The following sections describe the installation procedure for various operating systems.

Installing the JXTA Shell for Windows

To install the JXTA demo applications for the Windows platform, follow these steps:

1. Open `download.jxta.org/easyinstall/install.html` in a web browser.
2. If you already have a version 1.3 or later JRE installed on your machine, download the Windows Without Java VM installer; otherwise, download the Windows Includes Java VM installer.
3. When prompted by your web browser, specify a directory to store the downloaded installer.
4. After the download is complete, open Windows Explorer and go to the folder where you stored the downloaded installer.
5. Run the installer. It should be called either `JXTAInst.exe` or `JXTAInst_VM.exe`, depending on whether you chose the installer that includes the JVM.
6. Click Next to dismiss the Introduction dialog box.
7. The installer displays the License Agreement dialog box. Select the radio button titled I Accept the Terms of License Agreement, and click Next.
8. The installer prompts you to specify where the JXTA demo applications should be installed. Unless you have reason to install them elsewhere, use the default installation directory provided (`C:\Program Files\JXTA_Demo`). Click Install.

9. The installer installs the JXTA demo applications and then displays instructions on how to run the demo applications. Note these instructions before clicking Next to dismiss the launch instructions.
10. Click Done to close the installer.

The demo applications are now installed, and you can safely delete the installer that you downloaded.

Installing the JXTA Shell for Solaris, Linux, and UNIX

To install the JXTA demo applications for the Windows platform, follow these steps:

1. Open `download.jxta.org/easyinstall/install.html` in a web browser.
2. If you already have a version 1.3 or later JRE installed on your machine, download the Without Java VM installer for your platform; otherwise, download the Includes Java VM installer for your platform. The UNIX platform install does not have a version that includes a standalone JRE, so if you don't already have a JRE, you must download and install one first.
3. When prompted by your web browser, specify a directory to store the downloaded installer.
4. After the download is complete, open a console and go to the folder where you stored the downloaded installer.
5. Run the installer using `sh ./JXTAInst.bin`, replacing `JXTAInst.bin` with the name of the file that you downloaded. It should be called `JXTAInst.bin`, `JXTAInst_Sol_VM.bin`, or `JXTAInst_LNX_VM.bin`, depending on which version you chose to download.
6. Click Next to dismiss the Introduction dialog box.
7. The installer displays the License Agreement dialog box. Select the radio button titled I Accept the Terms of License Agreement, and click Next.
8. The installer prompts you to specify where the JXTA demo applications should be installed. Unless you have reason to install them elsewhere, use the default installation directory provided (`~/JXTA_Demo`). Click Install.
9. The installer installs the JXTA demo applications and then displays instructions on how to run the demo applications. Note these instructions before clicking Next to dismiss the launch instructions.
10. Click Done to close the installer.

The demo applications are now installed, and you can safely delete the installer that you downloaded.

Installing the JXTA Shell for Other Java-Supported Platforms

To install the JXTA demo applications for any other platform that supports Java, you can download a Java-based installer. However, you must already have a JRE installed on your machine. To install the JXTA demo applications for a Java-enabled platforms, follow these steps:

1. Open `download.jxta.org/easyinstall/install.html` in a web browser.
2. Download the Other Java-Enabled Platforms version of the JXTA Shell installer.
3. When prompted by your web browser, specify a directory to store the downloaded installer.
4. After the download is complete, open a console and go to the folder where you stored the downloaded installer.
5. Run the installer using `java -classpath JXTA_Demo.zip install`.
6. Click Next to dismiss the Introduction dialog box.
7. The installer display the License Agreement dialog box. Select the radio button titled I Accept the Terms of License Agreement, and click Next.
8. The installer prompts you to specify where the JXTA demo applications should be installed. Unless you have reason to install them elsewhere, use the default installation directory provided (usually `C:\Program Files\JXTA_Demo` or `~/JXTA_Demo`). Click Install.
9. The installer installs the JXTA demo applications and then displays instructions on how to run the demo applications. Note these instructions before clicking Next to dismiss the launch instructions.
10. Click Done to close the installer.

The demo applications are now installed, and you can safely delete the installer that you downloaded.

The Installation Directory Structure

When the installation is complete, the directory structure that's shown in Figure 3.3 appears.

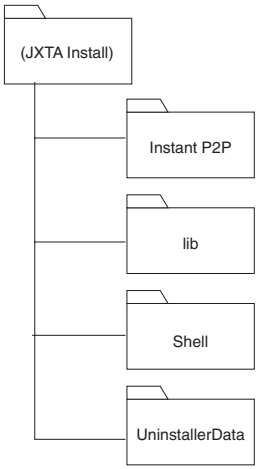

Figure 3.3 The installation directory structure.

(`JXTA Install`) is the installation directory that you specified to the installer. The `lib` subdirectory contains the JARs for the JXTA platform and the demo applications, and the `Shell` subdirectory contains the executable to start the Shell application. After the Shell is executed, the `Shell` subdirectory also holds a cache of configuration information and discovered peers and resources.

The `InstantP2P` directory contains another demo application that you do not use here. The `UninstallerData` directory contains the executable required to uninstall the JXTA demo applications.

Running the JXTA Shell

To start the JXTA Shell, follow the instructions provided at the end of the installation process.

On Windows, start the application by clicking Start, Programs, JXTA, JXTA Shell.

On other platforms, execute the script provided by the installer to start the application:

1. Open a command shell.
2. Go to the directory location that you specified for the JXTA Shell during the installation.
3. Go to the `Shell` subdirectory.
4. Execute the `shell.exe` or the `shell.sh script`.

Alternatively, you can invoke the Shell application directly using this command from the `Shell` subdirectory of the JXTA installation:

```
C:\Program Files\JXTA_Demo\Shell>java -classpath ..\lib\jxta.jar;
..\lib\jxtashell.jar;..\lib\log4j.jar;..\lib\jxtasecurity.jar;
..\lib\cryptix-asn1.jar;..\lib\cryptix32.jar;..\lib\minimalBC.jar;
..\lib\jxtaptls.jar net.jxta.impl.peergroup.Boot
```

On non-Windows platforms, you need to change the command given to match the directory and environment variable separator characters used by your platform. On Solaris, Linux, and UNIX, use `/` instead of `\`, and `:` instead of `;`.

Configuring the Shell

The first time you execute the application, you are presented with a screen requesting configuration information, as shown in Figure 3.4.

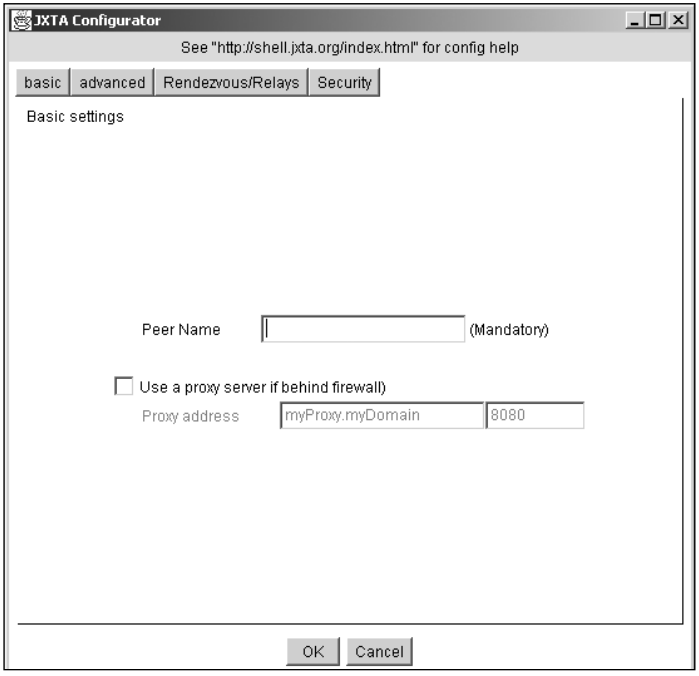

Figure 3.4 The basic configuration screen.

The user interface that appears, called the Configurator, is used by the reference implementation to configure the JXTA platform before starting the JXTA platform. To configure the JXTA platform using the Configurator, follow these steps:

1. Enter a name for your peer in the Peer Name text field.
2. Go to the Security tab.
3. Enter a username in the Secure Username text field.
4. Enter a password in the Password text field, and enter the same password in the Verify Password text field. Be sure to note the username and password that you enter because they will be required each time you start the JXTA platform in the future.
5. Click OK.

After you click OK, the JXTA platform starts and connects to the network. The time that it takes to start the JXTA platform varies with speed of your network connection, but it should take less than 30 seconds, at the most. Assuming that you have a simple network configuration, the Shell should start up and display the screen that's shown in Figure 3.5.

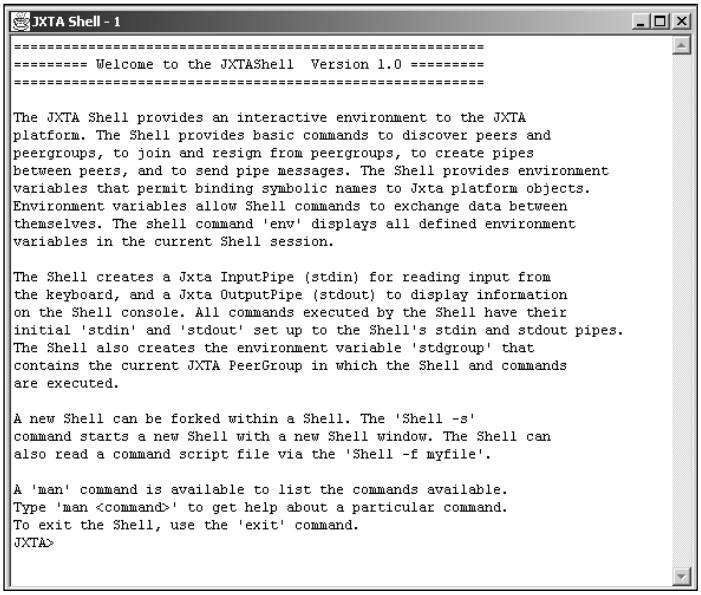

Figure 3.5 The JXTA Shell user interface.

To confirm that your client is correctly connected to the network, enter the `rdvstatus` command at the JXTA prompt:

```
JXTA>rdvstatus
```

If the Shell is correctly configured and managed to locate a rendezvous server, the `rdvstatus` command returns a similar result to the one given in Listing 3.3.

Listing 3.3 **Results of the *rdvstatus* Command**

```
Rendezvous Connection Status:

————————————————————

Is Rendezvous : [false]

Rendezvous Connections :

  Rendezvous name: JXTA.ORG 237
  Rendezvous name: JXTA.ORG 235
  Rendezvous name: ensd_1

Rendezvous Disconnections :

  [None]
```

This output shows that the Shell has correctly connected to three rendezvous peers, named `JXTA.ORG 237`, `JXTA.ORG 235`, and `ensd_1`. If you receive this response, your Shell peer is correctly configured and connected to the network; if you don't receive this response, see the next section to troubleshoot your configuration.

Troubleshooting Your Peer's Configuration

Listing 3.4 shows the output of the `rdvstatus` command when the client has failed to locate any rendezvous peers and cannot locate other peers.

Listing 3.4 **No Visible Rendezvous Peers**

```
Rendezvous Connection Status:

————————————————————

Is Rendezvous : [False]

Rendezvous Connections :
```

continues

Listing 3.4 **Continued**

```
[None]

Rendezvous Disconnections :

[None]
```

In some cases, it might take a few moments to see the rendezvous peers due to network latency. Wait a few moments before running `rdvstatus` again to see if the problem is simply high network latency. If the `rdvstatus` still shows no rendezvous peers, try using this command:

```
JXTA>peers -r
```

Wait a few moments and try the `rdvstatus` command again. If `rdvstatus` still fails to show any rendezvous peers, several possible reasons exist:

- No rendezvous peers are available.
- Your firewall configuration is preventing you from communicating with a rendezvous peer.
- You're not connected to a network.

If you aren't connected to a network, you can still use the Shell to experiment with the JXTA platform by following the instructions in the later section, "Using the JXTA Shell Without a Network Connection."

Finding Available Rendezvous Peers

First, confirm that rendezvous peers are available on the network:

1. Force the Shell to display the configuration screen the next time you start the Shell by typing the following from within the Shell:

   ```
   JXTA>peerconfig
   ```

   If you don't invoke this command before exiting the Shell, the Shell simply uses cached configuration information the next time it starts, with the same results. The `peerconfig` command will return this:

   ```
   peerconfig: Please exit and restart the jxta shell to
   reconfigure !!!!!
   ```

2. Follow the instructions and exit the shell by using the following command:

   ```
   JXTA>exit
   ```

3. Restart the Shell application the same way you started it the first time. This time you are prompted to enter only the username and password that you entered the first time in the Configurator. Enter the username and password, and hit Enter.
4. When the configuration screen appears this time, go to the Rendezvous/Relays tab and click Download Relay and Rendezvous Lists. The Load list from URL dialog box appears. (See Figure 3.6.)

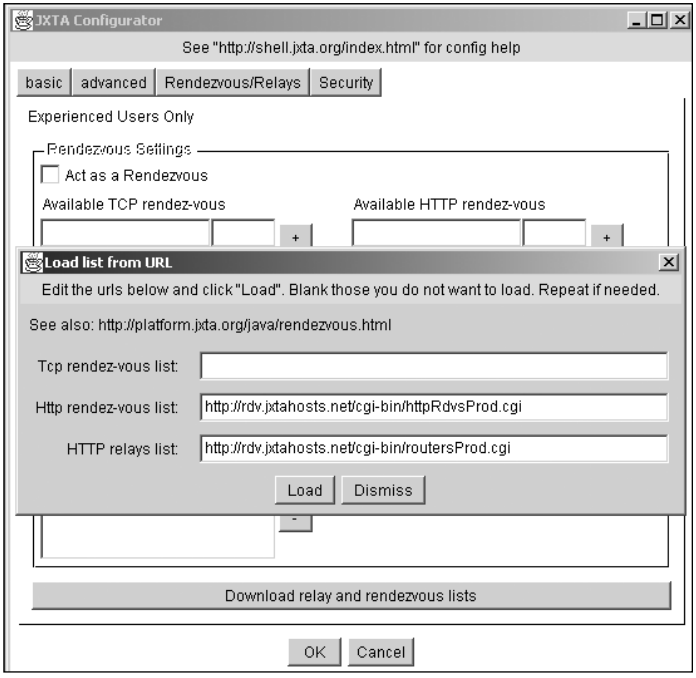

Figure 3.6 The Download Rendezvous/Router List dialog box.

To find rendezvous peers to use for peer discovery, the JXTA Shell attempts to download a list of available rendezvous peer IP addresses. This is a convenient mechanism for finding rendezvous peers, although you could just as easily enter the IP address and port of a rendezvous peer manually in the Rendezvous Settings section of Rendezvous/Router tab.

Using a web browser, go to the location shown in the Http rendez-vous list text field—by default, this value is as follows:

```
http://rdv.jxtahosts.net/cgi-bin/httpRdvsProd.cgi
```

This site returns a list of the production rendezvous peers run by Project JXTA. These peers are running the latest stable release of the JXTA platform, which should be the same as the platform version provided with the JXTA demo application installer. If the page returned is empty, no known production rendezvous peers are available from Project JXTA. Most likely, this is only a temporary situation occurring during an update to the rendezvous peer software. Try again later, or see the next section, "Using the JXTA Shell Without a Network Connection" for further instructions.

If the URL returns a list of rendezvous peers, you should test to make sure that at least one rendezvous peer in the list is operating at the specified IP address and port. To do this, you can use a web browser to request an acknowledgement from the rendezvous peer. For example, if the rendezvous peer is located at IP address 63.81.220.34 and is listening on port 9700, pointing a web browser to `http://63.81.220.34:9700/ping/` should return a blank web page. You can view the source of the web page to confirm that the result is a web page, not an error page.

If you have found a working rendezvous peer, the problem is mostly likely due to the configuration of a firewall between your machine and the outside network. Go to the Basic tab, check the Use a Proxy Server option, and enter the location of a HTTP proxy on your local network. You can obtain the location of your network's HTTP proxy by copying the proxy settings from your web browser or talking to your network administrator. The Shell should now show rendezvous peers when you start the application and run the `rdvstatus` command.

Using the JXTA Shell Without a Network Connection

If, for some reason, you don't have network access, you can still explore JXTA using the Shell and the experiments in the rest of this book. The Shell application is a peer like any other on a JXTA P2P network, so all the standard commands to manipulate peers, peer groups, and pipes will work exactly the same, independent of the location of the peer. However, you will be able to see, manipulate, and communicate only with your own peer.

If you want to experiment with the JXTA Shell in a more realistic environment, you can run two instances of the Shell on the same machine, using one of the Shell instances as a rendezvous peer. Due to the way the Java reference implementation of the JXTA platform implements its cache of configuration information, you need to make a copy of the Shell directory to prevent clashes between the instances of the Shell:

1. Force the Shell to display the configuration dialog box the next time it starts using the `peerconfig` command from within the Shell.
2. Exit the Shell using `exit`.

3. Make a copy of the `Shell` subdirectory (located underneath the JXTA installation directory) called `Shell2` at the same level as the `Shell` subdirectory.

Before attempting to configure each Shell, you should know your machine's local IP address.

On Windows, follow these steps:

1. Open a command prompt.
2. Invoke the `ipconfig` command.
3. Note the IP address specified in the output from `ipconfig`.

You should also ensure that you can ping your own IP address because some internal networks might not allow you to see your own IP address. To check if you can see your own IP address, follow these steps:

1. Open a command prompt.
2. Invoke the `ping` command.
3. Ensure that the response doesn't indicate that the destination host is unreachable.

If you cannot ping the IP address returned by `ipconfig`, you should use the localhost IP address 127.0.0.1 instead of the IP address returned by `ipconfig`. On other operating systems, consult your operating system's help system to learn how to determine your machine's IP address and ping an IP address.

To start one Shell as a rendezvous peer, open a command prompt and follow these steps:

1. Go to the `Shell` subdirectory.
2. Start the Shell using the executable or script in the directory directly (`shell.exe` or `shell.sh`) or manually using the `java` command.
3. Enter a name for the peer.
4. Go to the Rendezvous/Relays tab.
5. Remove each TCP and HTTP rendezvous server and each HTTP relay server.
6. Deselect Use a Relay in the HTTP Relay Settings section.
7. Select Act as a Rendezvous.
8. Go to the Advanced tab.
9. Deselect Enabled from the HTTP Settings section.
10. Select Enabled and Manual from the TCP Settings section.

11. Select Always Manual, and note the IP address and port number (default 9701) that has been automatically set. If no IP address has been set, enter the IP address that you obtained from your operating system.
12. Click OK.
13. Enter your username and password when prompted, and hit Enter.

To start a second Shell to act as a simple peer using the rendezvous peer that you just created, open a second command prompt and do the following:

1. Go the `Shell2` subdirectory that you created.
2. Remove the `pse` subdirectory. This directory contains the personal security settings protected by the password entered in the Configurator.
3. Remove the `PlatformConfig` file. This file contains configuration information for your peer, and it must be removed to prevent the second instance from reusing the peer's unique ID.
4. Start the Shell using the executable or script in the directory directly (`shell.exe` or `shell.sh`) or manually using the `java` command.
5. Enter a name for the peer, preferably one that is different from the one you used for the rendezvous peer.
6. Go to the Rendezvous/Relays tab.
7. Remove each TCP and HTTP rendezvous server and each HTTP relay server.
8. Deselect Use a Relay in the HTTP Relay Settings section.
9. Enter the IP address and port that you noted in the first shell as a TCP Rendezvous, and add it to the list using the + button.
10. Go to the Advanced tab.
11. Select Enabled and Manual from the TCP Settings section.
12. Select Always Manual, and enter your IP address and a different port number (say, 9702).
13. Deselect Enabled from the HTTP Settings section.
14. Go the Security tab.
15. Enter a username and password.
16. Click OK.

You now have a simple peer configured to use the first instance of the Shell as a rendezvous peer, and you can conduct P2P communication between the two peers as normal.

Navigating the JXTA Shell

The JXTA Shell presents a simple command-line user interface similar to UNIX's interface. Simple text commands are entered at the JXTA prompt:

```
JXTA>
```

Like most UNIX shells, the Shell is case-sensitive and maintains a history of previously issued commands. At any time, you can see a complete list of the previously issued commands by using the `history` command:

```
JXTA>history
  0 man
  1 history
```

At any time, you can scroll through the commands using the up and down arrow keys, invoking previous commands without retyping the command.

Learning About Shell Commands

The JXTA Shell resembles a UNIX shell in many ways, and several of the commands are available within the Shell. To learn what commands are available from the Shell, you can use the `man` command by itself to print a list of all available commands, shown in Table 3.1:

```
JXTA>man
```

Table 3.1 **Built-In Shell Commands**

Command	Description
`cat`	Concatenates and displays a Shell object
`chpgrp`	Changes the current peer group
`clear`	Clears the shell's screen
`env`	Displays environment variable
`exit`	Exits the Shell
`exportfile`	Exports to an external file
`get`	Gets data from a pipe message
`grep`	Searches for matching patterns
`groups`	Discovers peer groups
`help`	Gives instructions on where to find help
`history`	Shows the history of Shell commands executed
`importfile`	Imports an external file
`instjar`	Installs JAR files containing additional Shell commands

continues

Table 3.1 **Continued**

Command	Description
`join`	Joins a peer group
`leave`	Leaves a peer group
`man`	Online help command that displays information about a specific Shell command
`mkadv`	Makes an advertisement
`mkmsg`	Makes a pipe message
`mkpgrp`	Creates a new peer group
`mkpipe`	Creates a pipe
`more`	Pages through a Shell object
`peerconfig`	Forces reconfiguration the next time the Shell is started
`peerinfo`	Gets information about peers
`peers`	Discovers peers
`put`	Puts data into a pipe message
`rdvserver`	Runs the peer as a standalone rendezvous server
`rdvstatus`	Displays information about rendezvous
`recv`	Receives a message from a pipe
`search`	Discovers JXTA advertisements
`send`	Sends a message into a pipe
`set`	Sets an environment variable
`setenv`	Sets an environment variable
`sftp`	Sends a file to another peer
`share`	Shares an advertisement
`Shell`	Forks a new JXTA Shell command interpreter
`Sql`	Issues an SQL command (not implemented)
`Sqlshell`	Acts as the JXTA SQL Shell command interpreter
`Talk`	Talks to another peer
`Uninstjar`	Uninstalls JAR files previously installed with `instjar`
`Version`	Returns the Shell version information
`wc`	Counts the number of lines, words, and characters in an object
`who`	Displays credential information
`whoami`	Displays information about a peer or a peer group

The `man` command also enables you to learn about the purpose and options for various commands available within the Shell. For example, to find out more about the `rdvstatus` command, use this command:

```
JXTA>man rdvstatus
```

This pulls up the usage information for the `rdvstatus` command, as shown in Listing 3.5.

Listing 3.5 **Usage Information for *rdvstatus***

```
NAME
     rdvstatus - display information about rendezvous

SYNOPSIS
     rdvstatus [-v]
       [-v]    print verbose information

DESCRIPTION

rdvstatus displays information about the peer
rendezvous. The command shows how many rendezvous peers
the peer is connected to.

OPTIONS
       -v    print verbose information

EXAMPLE

    JXTA>rdvstatus


SEE ALSO
    whoami peers
```

Environment Variables

The Shell provides environment variables to store pieces of information in the Shell for later use. You can see the defined environment variables using the `env` command, as shown in Listing 3.6.

Listing 3.6 **The Shell Environment Variables**

```
JXTA>env
stdin = Default InputPipe (class net.jxta.impl.shell.ShellInputPipe)
SHELL = Root Shell (class net.jxta.impl.shell.bin.Shell.Shell)
History = History (class net.jxta.impl.shell.bin.history.HistoryQueue)
parentShell = Root Shell (class net.jxta.impl.shell.bin.Shell.Shell)
Shell = Root Shell (class net.jxta.impl.shell.bin.Shell.Shell)
stdout = Default OutputPipe (class net.jxta.impl.pipe.NonBlockingOutputPipe)
consout = Default Console OutputPipe (class
net.jxta.impl.shell.ShellOutputPipe)
consin = Default Console InputPipe (class
net.jxta.impl.shell.ShellInputPipe)
stdgroup = Default Peer Group (class net.jxta.impl.peergroup.StdPeerGroup)
```

These environment variables are set by default to handle the input, output, and basic functionality of the Shell. More variables can be defined by the output of commands, each corresponding to an object, data, or cached advertisement accessible within the Shell's environment.

Importing and Exporting Environment Variables

In addition to working with environment variables within the Shell, environment variables can be imported and exported using the `importfile` and `exportfile` commands. The commands enable you to import XML or plain text files into Shell environment variables. By default, the working directory for these commands is set to the directory where you executed the Shell, usually the Shell subdirectory of the JXTA installation directory.

To demonstrate the `importfile` and `exportfile` commands, follow these steps:

1. Create a text file called `input.txt` containing some text in the Shell subdirectory under the JXTA installation directory.
2. Import the text of the file into an environment variable called `test` using `importfile -f input.txt test`.

You should see a new environment variable named `test` in the list of variables returned by the `env` command. Rather than trying to find a variable in the output of `env`, you can use the `cat` command to view the contents of the `test` variable, as shown in Listing 3.7:

```
JXTA>cat test
```

Listing 3.7 **The Imported Environment Variable**

```
<?xml version="1.0"?>

<ShellDoc>
      <Item>
            This is some test text.
      </Item>
</ShellDoc>
```

The `cat` command knows how to render several types of environment variables to the standard output of the Shell, including the XML document produced by importing `input.txt` with the `importfile` command. The `test` environment variable can now be exported to a file called `output.txt` using this command:

```
JXTA>exportfile -f output.txt test
```

A file called `output.txt` containing the contents of the `test` variable appears in the Shell subdirectory of the JXTA installation.

Although the usefulness of this functionality might seem trivial now, remember that all functionality in JXTA is expressed in terms of XML-based advertisements. As you'll see, having the capability to manipulate environment variables is central to the power of the JXTA Shell as a tool for experimenting with the JXTA platform.

Combining Commands

In a manner similar to UNIX, the JXTA Shell allows users to string together simple commands to perform complex functionality. The `|` operator allows the output of a command on the left of the operator to be used as input into the command on the right.

Consider a simple example: The output of the `man` command is a bit too long to read without scrolling. The `more` command breaks a piece of input text into screen-size chunks. You can combine these two commands by typing the following:

```
JXTA>man | more
```

This pipes the output of the `man` command as input into the `more` command, allowing you to view the `man` output in screen-size chunks that you can move between by hitting the Enter key. Similarly, you could count the number of characters in the `man` output using this command:

```
JXTA>man | wc -c
```

This pipes the output of the `man` command as input into the `wc` command, which counts the number of characters in the input when the `-c` option is set.

Manipulating Peers

The JXTA Shell provides basic capabilities to discover peers and obtain peer information. Working with a peer involves working with a Peer Advertisement that describes the peer and its services.

Learning About the Local Peer

Before learning about other peers, you need to know a bit about your own local peer:

```
JXTA>whoami
```

The `whoami` command returns the peer information for the local peer run by the JXTA Shell given in Listing 3.8.

Listing 3.8 **Results of the *whoami* Shell Command**

```
<Peer>MyPeer</Peer>
<Keywords>NetPeerGroup by default</Keywords>
<PeerId>urn:jxta:uuid-59616261646162614A78746150325033855A703D4E614D
B7B54A9BE583FFCD4C03</PeerId>
<TransportAddress>tcp://asterix:9701/</TransportAddress>
<TransportAddress>http://JxtaHttpClientuuid-59616261646162614A787461
50325033855A703D4E614DB7B54A9BE583FFCD4C03/</TransportAddress>
```

This short version of the local peer information shows only the basic peer information. A longer version that displays the whole Peer Advertisement stored in environment variable `peerX` can be viewed using this command:

```
JXTA>cat peerX
```

You can find the environment variable holding your Peer Advertisement by looking for your peer's name in the results of the `peers` command.

I won't go into the details of the Peer Advertisement at this point. I will provide a complete description of the Peer Advertisement when we explore the Peer Discovery Protocol in the next chapter. For now, it's enough to notice some of the information provided by the advertisement:

- A name for the peer
- A unique identifier for the peer

- Services provided by the peer
- Transport endpoint details

The services provided by the peer are called peer services; these are services offered only by the peer. If the peer disconnects from the network, these services are unavailable to other peers.

Finding Peers

Before your peer can request services from a peer, it needs to know the existence of the peer, what services the peer offers, and how to contact the peer on the network. To find peers that your local peer is already aware of, execute the `peers` command given in Listing 3.9.

Listing 3.9 **Results of the *peers* Shell Command**

```
JXTA>peers
peer0: name = rdv-235
peer1: name = rdv-237
peer2: name = dI_lab1
peer3: name = dI_lab_Tokyo
peer4: name = MyPeer
```

Each Peer Advertisement is made available in the Shell environment via a variable with a name of the form `peerX`, where `X` is an integer. At this point, your peer is aware of only local or cached Peer Advertisements for peers that have already been discovered; no discovery of remote peers has yet been performed. Caching Peer Advertisements reduces the amount of discovery that a peer might have to perform and can be used by simple peers as well as rendezvous peers to reduce network traffic.

Each entry returned by the `peers` command shows the simple peer name for a peer and the name of an environment variable storing the Peer Advertisement for that peer. In the previous example, the `peer4` environment variable stores the Peer Advertisement for the local peer. You can view the Peer Advertisement using the `cat` command:

```
JXTA>cat peer4
```

To discover other peers on the network, you need to send a peer discovery message using the following:

```
JXTA>peers -r
peer discovery message sent
```

This sends a discovery message immediately to all the rendezvous peers that your peer is aware of on the network. The rendezvous peers forward the request to other rendezvous and simple peers that it is aware of on the network. The rendezvous peers might potentially reply using cached Peer Advertisements to improve the response time and reduce network traffic across the P2P network. The `peers` command returns to the JXTA prompt immediately, and the discovered peers can be viewed using the `peers` command, as shown in Listing 3.10.

Listing 3.10 **The Updated List of Discovered Peers**

```
JXTA>peers
peer0: name = cajunboy
peer1: name = fds
peer2: name = Rdv-235
peer3: name = domehuhu
peer4: name = MyPeer
peer5: name = Rdv-236
...
```

The results of the peer discovery might not be immediately viewable with the `peers` command. JXTA provides no guarantees about the time required to receive a response to a discovery message; it is possible that responses might never return. The delay depends on a variety of factors, including the speed of your connection to other peers and the network configuration (firewall, NAT).

Flushing Cached Peer Advertisements

At some point, it might be appropriate to remove the Peer Advertisements from the local cache, eliminating the local peer's knowledge of other peers on the network. To flush the local cache of Peer Advertisements, use this command:

```
JXTA>peers -f
```

The only remaining Peer Advertisement will be that of your own local peer:

```
JXTA>peers
peer0: name = MyPeer
```

To find peers on the network, you need to send another peer discovery message to the network using the `peers -r` command to populate the local cache of Peer Advertisements.

Manipulating Peer Groups

In the same manner that you just managed to discover and manipulate peer information, you can discover and manipulate peer groups. Working with a peer group involves working with a Peer Group Advertisement that describes the peer group and its services.

Learning About the Current Peer Group

The `whoami` command permits you to examine the peer group information for the local peer's current peer group. In the Shell, the peer can manipulate only one peer group at a time. For convenience, this peer group is set as the current peer group in an environment variable called `stdgroup`. To retrieve information about the current peer group, use `whoami -g` to obtain the peer group information in a form similar to this:

```
<PeerGroup>NetPeerGroup</PeerGroup>
<Description>NetPeerGroup by default</Description>
<PeerGroupId>urn:jxta:jxta-NetGroup</PeerGroupId>
```

This peer group information shows that the peer is currently a part of the Net Peer Group. By default, all peers are members of the Net Peer Group, thereby allowing all peers on the network to see and communicate with each other.

The peer group information returned by `whoami -g` is a condensed version of the information provided by the peer group's advertisement. A Peer Group Advertisement also contains information on the set of services that the peer group makes available to its members. These services are called peer group services to distinguish them from peer services. Peer group services can be implemented by several members of a peer group, enabling redundancy. Unlike a peer service, a peer group service remains available as long as one member of the peer group is connected to the P2P network and is providing the service.

Finding Peer Groups

In a similar manner to viewing the known peers on the network, you can view the known peer groups using this command:

```
JXTA>groups
```

As with the `peers` command, only those peer groups that have been discovered in the past and have had their Peer Group Advertisement cached appear in the list when this command is executed in an instance of the Shell. Although all peers belong to the Net Peer Group and this group is always present, the Net Peer Group does not show up in the results from the `groups` command.

To find peer groups available on the P2P network, a peer group discovery request must be made to the network:

```
JXTA>groups -r
group discovery message sent
```

Using the `groups` command again returns a list of groups discovered on the network:

```
JXTA>groups
group0: name = SomeGroup
group1: name = AnotherGroup
...
```

As with peer discovery, the response to a group discovery message might not be immediate, if a response is obtained at all. Each of the cached Peer Group Advertisements is available in the environment as a variable with a name of the form `groupX`, where `X` is an integer. The contents of the environment variable can be viewed using the `cat` command:

```
JXTA>cat group0
```

This command displays the full Peer Group Advertisement instead of the condensed version returned by `whoami -g`.

Creating a Peer Group

A new peer group can be created from within the JXTA Shell in two ways: by cloning the Net Peer Group Peer Group Advertisement or by creating a new Peer Group Advertisement from scratch.

Cloning The Net Peer Group

To create a new peer group, use the `mkpgrp` command and provide a name for your peer group:

```
JXTA>mkpgrp MyGroup
```

Used this way, the `mkpgrp` command makes a new peer group by cloning the existing Net Peer Group peer group.

Creating a New Peer Group Advertisement

Instead of cloning the existing Net Peer Group, you can create a new Peer Group Advertisement with a given name using this command:

```
JXTA>MyGroupAdvertisement = mkadv -g <name>
```

This form of the `mkadv` command creates a new Peer Group Advertisement by cloning the current peer group. If you haven't yet joined any groups, the current peer group is the Net Peer Group, and the result is identical to using the

mkpgrp command. Alternatively, you can import a saved Peer Group Advertisement from a text file and use it to create the advertisement:

```
JXTA>importfile -f advertisement.txt MyDocument
JXTA>MyAdvertisement = mkadv -g -d MyDocument
JXTA>mkpgrp -d MyAdvertisement MyGroup
```

This set of commands imports a file called advertisement.txt, creates a Peer Group Advertisement out of its contents, and uses them to create a new peer group called MyGroup.

> **Note**
> Currently, the Shell ignores the MyGroup name for the peer group and uses the name from the Peer Group Advertisement; this is a known bug with the current Shell implementation.

Joining a Peer Group

When your peer is aware of a peer group, either by creating one or by performing peer group discovery, you must join the group before any communication as a part of that peer group can occur. To join a group whose Peer Group Advertisement is stored in an environment variable called group1, use the join -d command:

```
JXTA>join -d group1
```

The join command prompts you for an identity that you want to use on this group:

```
Enter the identity you want to use when joining this peergroup (nobody)
Identity:
```

Identities assign credentials to users for accessing peer resources. The peer group's Membership service is responsible for defining accepted identities and authenticating peers that want to join a group.

The join -d command sets the current peer group in the environment to the most recently joined peer group. Issuing the join command again lists the current known groups and their status:

```
JXTA>join
 Unjoined Group : AnotherGroup
 Joined Group   : MyGroup       (current)
 Unjoined Group : SomeGroup
```

If you make another group called MyGroup2 and join it, the current peer group changes to reflect MyGroup2 as the current peer group:

```
JXTA>join
 Unjoined Group : AnotherGroup
```

```
 Joined Group   : MyGroup
 Joined Group   : MyGroup2       (current)
 Unjoined Group : SomeGroup
```

To move between peer groups, change the current shell peer group by issuing the `chpgrp` command, as shown in Listing 3.11.

Listing 3.11 **Changing the Current Peer Group**

```
JXTA>chpgrp MyGroup
JXTA>join
 Unjoined Group : AnotherGroup
 Joined Group   : MyGroup        (current)
 Joined Group   : MyGroup2
 Unjoined Group : SomeGroup
```

If you decide to leave a peer group, issue the `leave` command; your peer will leave the current peer group, as shown in Listing 3.12.

Listing 3.12 **Result of Leaving a Group**

```
JXTA>leave
JXTA>join
 Unjoined Group : AnotherGroup
 Unjoined Group : MyGroup
 Joined Group   : MyGroup2
 Unjoined Group : SomeGroup
```

After you leave a peer group, the current peer group is set to the Net Peer Group. You must issue a `chpgrp` command to set the current peer group again.

Flushing Cached Peer Group Advertisements

Just as it might be appropriate to remove the Peer Group Advertisements from the local cache, it might also be appropriate to remove peer group advertisements from the local cache. To flush the local cache of Peer Group Advertise-ments, thereby eliminating the local peer's knowledge of peer groups on the network, use this command:

```
JXTA>groups -f
```

To join a peer group on the network, you need to send another peer group discovery message to the network using the `groups -r` command to populate the local cache of Peer Group Advertisements.

Manipulating Pipes

Pipes provide the basic mechanism for peers to share information with each other. Pipes and pipe endpoints are abstractions of the underlying network-transport mechanism responsible for providing network connectivity. Communicating with other peers involves discovering pipes and endpoints, binding to a pipe, and sending and receiving messages through the pipe.

Creating Pipes

To create a pipe, you must first create a Pipe Advertisement:

```
JXTA>MyPipeAdvertisement = mkadv -p
```

Using the `cat` command, you can view the newly created Pipe Advertisement, as shown in Listing 3.13.

Listing 3.13 **Viewing the New Pipe Advertisement**

```
JXTA>cat MyPipeAdvertisement
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
      <Id>
            urn:jxta:uuid-59616261646162614E50472050325033 9C0C74ADD7094CEC90EC9D4471DFED5304
      </Id>
      <Type>JxtaUnicast</Type>
</jxta:PipeAdvertisement>
```

When a peer has a Pipe Advertisement, defining a pipe from the Pipe Advertisement is as simple as using these commands:

```
JXTA>MyInputPipe = mkpipe -i MyPipeAdvertisement
JXTA>MyOutputPipe = mkpipe -o MyPipeAdvertisement
```

This defines an input and an output pipe from the advertisement stored in the `MyPipeAdvertisement` environment variable.

Creating a Message

Communication between an input and an output pipe relies on the capability to form a message object to exchange. If you import a text file into the Shell, you can package it inside a message:

```
JXTA>importfile -f test.txt SomeData
JXTA>MyMessage = mkmsg
JXTA>put MyMessage MyData SomeData
```

The last line places the contents of the `SomeData` variable inside an element called `MyData`, as shown in Listing 3.14.

Listing 3.14 **The Newly Created Message**

```
JXTA>cat MyMessage
Tag: MyData
Body:
<?xml version="1.0"?>

<ShellDoc>
     <Item>
          This is some test text.
     </Item>
</ShellDoc>
```

Sending and Receiving Messages

To demonstrate how simple it is to send a message using a pipe, you're going to send a message from the peer to itself. To send the message from the peer, first define the input and output pipes:

```
JXTA>MyPipeAdvertisement = mkadv -p
JXTA>MyInputPipe = mkpipe -i MyPipeAdvertisement
JXTA>MyOutputPipe = mkpipe -o MyPipeAdvertisement
```

Next, import a file that will form the body of the message:

```
JXTA>importfile -f test.txt SomeData
JXTA>MyMessage = mkmsg
JXTA>put MyMessage MyData SomeData
```

Now send the message:

```
JXTA>send MyOutputPipe MyMessage
```

To receive the message from the pipe, use this command:

```
JXTA>ReceivedMessage = recv -t 5000 MyInputPipe
```

This command attempts to receive a message from the `MyInputPipe` input pipe and store it in the `ReceivedMessage` variable. The command attempts to receive a message for only five seconds before timing out.

If the attempt to receive a message is successful, the command returns the following:

```
recv has received a message
```

The data can be extracted from the received message, as shown in Listing 3.15.

Listing 3.15 **Viewing the Received Message Data**

```
JXTA>NewData = get ReceivedMessage MyData
JXTA>cat NewData
<?xml version="1.0"?>

<ShellDoc>
      <Item>
            This is some test text.
      </Item>
</ShellDoc>
```

If no message is available to be received, the Shell reports the following:

```
recv has not received any message
```

The Shell recognizes whether a pipe is not the appropriate type required to send or receive a message. Attempting to send using an input pipe instead of an output pipe results in an error:

```
JXTA>send MyInputPipe MyMessage
send: MyInputPipe is not an OutputPipe
java.lang.ClassCastException: net.jxta.impl.pipe.InputPipeImpl
```

Similarly, attempting to receive using an output pipe instead of an input pipe results in an error:

```
JXTA>inputMessage = recv -t 5000 outputPipe
wait: outputPipe is not an InputPipe
java.lang.ClassCastException: net.jxta.impl.pipe.NonBlockingOutputPipe
```

Talking to Other Peers

The `talk` command is a simple application written on top of the JXTA Shell that allows you to talk to other peers. To do this, first create a talk advertisement for a specific username:

```
JXTA>talk -register myusername
```

This has to be done only once as the platform caches the advertisement. Next, start a talk listener daemon using this command:

```
JXTA>talk -login myusername
```

After this, you can talk to another user:

```
JXTA>talk -u myusername myfriendsusername
```

This allows you to enter text messages that will be sent to the other talk user `myfriendsusername`, as shown in Figure 3.7. You can even send a text message to yourself using this command:

```
JXTA>talk -u myusername myusername
```

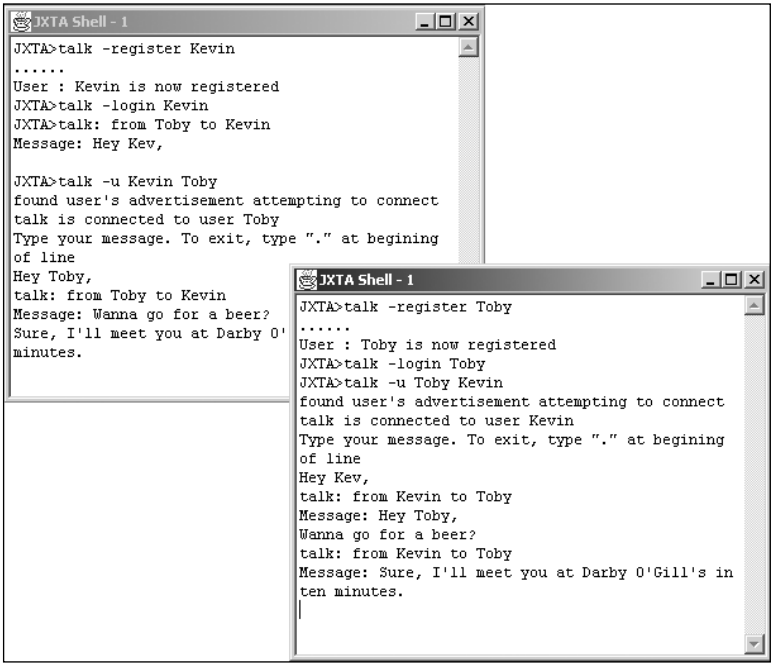

Figure 3.7 Using `talk` between two shell instances.

When you're done talking for the session, use this command to shut down the `talk` daemon:

```
JXTA>talk -logout myusername
```

Extending the Shell Functionality

The JXTA Shell is designed to be more than just a toy to explore the basic building blocks of P2P technology. The Shell is designed to allow developers to extend its functionality easily and incorporate new commands. All the core commands that you've used so far are invoked dynamically, and any new commands that a developer creates will be invoked the same way.

A developer needs to follow a few simple rules to create a new command for the Shell. To work in the Shell properly, a new command must do the following:

- Extend the `net.jxta.impl.shell.ShellApp` class
- Implement the `startApp` and `stopApp` methods
- Be part of a subpackage of `net.jxta.impl.shell.bin`
- Exist in a subpackage of the same name as the command
- Be in a class of the same name as the command

A Simple Shell Command

Following these simple rules, you'll now write a simple command to print the name of the peer. Listing 3.16 creates a command called `helloworld`.

Listing 3.16 **The *helloworld* Shell Command (*helloworld.java*)**

```
package net.jxta.impl.shell.bin.helloworld;

import net.jxta.impl.shell.ShellApp;
import net.jxta.impl.shell.ShellEnv;
import net.jxta.impl.shell.ShellObject;

import net.jxta.peergroup.PeerGroup;

/**
 * A simple example command for the JXTA Shell.
 */
public class helloworld extends ShellApp
{
    /**
     * The shell environment.
     */
    private ShellEnv theEnvironment;

    /**
     * Invoked by the Shell to starts the command.
     *
     * @param   args a set of arguments passed to the command.
     * @return  a status code indicating the success or failure
     *          of the command.
     */
```

continues

Listing 3.16 **Continued**

```
    public int startApp(String[] args)
    {
        println("Starting command...");

        // Get the shell's environment.
        theEnvironment = getEnv();

        // Use the environment to obtain the current peer group.
        ShellObject theShellObject = theEnvironment.get("stdgroup");
        PeerGroup aPeerGroup = (PeerGroup) theShellObject.getObject();

        // Check to see if there were any command arguments.
        if ((args == null) || (args.length == 0))
        {
            // Print the peer name to the console.
            println("My peer name is " + aPeerGroup.getPeerName());
        }
        else
        {
            println("This command doesn't support arguments.");

            // Return the 'parameter error' status code.
            return ShellApp.appParamError;
        }

        // Return the 'no error' status code.
        return ShellApp.appNoError;
    }

    /**
     * Invoked by the Shell to stop the command.
     */
    public void stopApp()
    {
        // Do nothing.
    }
}
```

As demanded by the rules of the Shell, the `helloworld` class is a part of the `net.jxta.impl.shell.bin.helloworld` package and implements the `startApp` and `stopApp` methods. In this simple example, the command retrieves an object representing the current peer group using the `stdgroup` environment variable:

```
ShellObject theShellObject =
    theEnvironment.get("stdgroup");
```

The `ShellEnv` object is the same store of environment objects that you've been working with from inside the Shell throughout this chapter. The `PeerGroup` object is retrieved from the wrapper `ShellObject` returned by `ShellEnv`:

```
PeerGroup aPeerGroup =
    (PeerGroup) theShellObject.getObject();
```

Finally, the name of the peer in the peer group is printed to the console using the Shell's standard output:

```
println("My peer name is " +
    aPeerGroup.getPeerName());
```

To make this command work with the Shell, compile the `helloworld.java` source from the command line. To make life easier, place the source code in the `Shell` subdirectory of the JXTA demo installation and compile it using the following:

```
javac -d . -classpath ..\lib\jxta.jar;..\lib\jxtashell.jar helloworld.java
```

Now execute the Shell, making sure to include the current directory in the classpath:

```
java -classpath .;..\lib\jxta.jar;..\lib\jxtashell.jar;..\lib\cms.jar;
..\lib\cmsshell.jar;..\lib\log4j.jar;..\lib\beepcore.jar;
..\lib\cryptix32.jar;..\lib\cryptix-asn1.jar;..\lib\jxtaptls.jar;
..\lib\jxtasecurity.jar;. net.jxta.impl.peergroup.Boot
```

The Shell starts up as usual, and you can now try your new command:

```
JXTA>helloworld
Starting command...
My peer name is MyPeer
```

Congratulations, you just created your first solution using JXTA! Although this example doesn't do much, it demonstrates how simple it is to build on the JXTA platform to incorporate new functionality.

Summary

This chapter provided a crash course on using the JXTA Shell. Most of the details of JXTA, its protocols, and the Java reference implementation are revealed in the following chapters. In the next chapter, you start examining the JXTA platform in detail by looking at the Peer Discovery Protocol and its components. Your familiarity with the JXTA Shell will come in handy by providing a framework for the examples, thereby reducing the amount of coding required and allowing the examples to focus on the particulars of peer discovery.