This chapter presents several JXTA programming examples that perform common tasks such as peer and peer group discovery, creating and publishing advertisements, creating and joining a peer group, and using pipes.

Note – These examples were developed and tested using the 03-02-2003 JXTA Stable Build (JXTA_2_0_Stable_20030301 03-02-2003).

Peer Discovery

This programming example illustrates how to discover other JXTA peers on the network. The application instantiates the JXTA platform, and then sends out Discovery Query messages to the default netPeerGroup looking for any JXTA peer. For each Discovery Response message received, the application prints the name of the peer sending the response (if it is known) as well as the name of each peer that was discovered.

Figure 7-1 shows example output when this application is run:

```
Sending a Discovery Message
Sending a Discovery Message
Got a Discovery Response [5 elements] from peer : unknown
  Peer name = suz
  Peer name = jsoto-2K
  Peer name = peertopeer
  Peer name = JXTA.ORG 237
  Peer name = Frog@SF05
Sending a Discovery message
Got a Discovery Response [5 elements] from peer : unknown
  Peer name = Mr Magoo
  Peer name = mypc
  Peer name = yaro-work
  Peer name = johnboy2
  Peer name = Lomax@DIOXINE.NET
```

Figure 7-1 Example output: Peer discovery example.
Note — Because Discovery Responses are sent asynchronously, you may need to wait while several Discovery Requests are sent before receiving any responses. If you don’t receive any Discovery Responses when you run this application, you most likely haven’t configured your JXTA environment correctly. You will typically want to specify at least one rendezvous peer. If your peer is located behind a firewall or NAT, you will also need to specify a relay peer. Remove the PlatformConfig file that was created in the current directory and re-run the application. When the JXTA Configurator appears, enter the correct configuration information. See http://platform.jxta.org/java/confighelp.html for more details on using the JXTA Configurator tool.

Discovery Service

The JXTA DiscoveryService provides an asynchronous mechanism for discovering peer, peer group, pipe, and service advertisements. Advertisements are stored in a persistent local cache (the ./.jxta/cm directory). When a peer boots up, the same cache is referenced. Within the ./.jxta/cm directory, subdirectories are created for each peer group that is discovered.

- ./.jxta/cm/jxta-NetGroup — contains advertisements for the net peer group
- ./.jxta/cm/group-ID — contains advertisements for this group

These directories will contain files of the following types:
- * idx — index files
- record-offsets.tbl — entry list store
- advertisements.tbl — advertisement store

A JXTA peer can use the getLocalAdvertisements() method to retrieve advertisements that are in its local cache. If it wants to discover other advertisements, it uses getRemoteAdvertisements() to send a Discovery Query message to other peers. Discovery Query messages can be sent to a specific peer or propagated to the JXTA network. In the J2SE platform binding, Discovery Query messages not intended for a specific peer are propagated on the local subnet utilizing IP multicast and also sent to the peer’s rendezvous. Connection to the rendezvous peer occurs asynchronously. If this peer has not yet connected to a rendezvous, the Discovery Query message will only be sent to the local subnet via multicast. Once the peer has connected to a rendezvous, the Discovery Query message will also be propagated to the rendezvous peer. A peer includes its own advertisement in the Discovery Query message, performing an announcement or automatic discovery mechanism.

There are two ways to receive DiscoveryResponse messages. You can wait for one or more peers to respond with DiscoveryResponse messages, and then make a call to getLocalAdvertisements() to retrieve any results that have been found and have been added to the local cache. Alternately, asynchronous notification of discovered peers can be accomplished by adding a Discovery Listener whose callback method, discoveryEvent(), is called when discovery events are received. If you choose to add a Discovery Listener, you have two options. You can call addDiscoveryListener() to register a listener. Or, you can pass the listener as an argument to the getRemoteAdvertisements() method.

The DiscoveryService is also used to publish advertisements. This is discussed in more detail in the example “Creating Peer Groups and Publishing Advertisements” on page 44.

The following classes are used in this example:
- net.jxta.discovery.DiscoveryService — asynchronous mechanism for discovering peer, peer group, pipe and service advertisements and publishing advertisements.
- `net.jxta.discovery.DiscoveryListener` — the listener interface for receiving DiscoveryService events.
- `net.jxta.DiscoveryEvent` — contains Discovery Response messages.
- `net.jxta.protocol.DiscoveryResponseMsg` — defines the Discovery Service "response"

**DiscoveryDemo**

This example uses the DiscoveryListener interface to receive asynchronous notification of discovery events. [The code for this example begins on page 36.] We define a single class, DiscoveryDemo, which implements the DiscoveryListener interface. We also define a class variable:

- `PeerGroup netPeerGroup` — our peer group (the default net peer group) [line 14]

and four methods:

- `public void startJxta()` — initialize the JXTA platform [line 18]
- `public void run()` — thread to send DiscoveryRequest messages [line 39]
- `public void discoveryEvent(DiscoveryEvent ev)` — handle DiscoveryResponse messages that are received [line 67]
- `static public void main()` — main routine [line 96]

**startJxta() method**

The `startJxta()` method instantiates the JXTA platform (the JXTA world group) and creates the default net peer group [line 31]:

```java
netPeerGroup = PeerGroupFactory.newNetPeerGroup();
```

Next, our discovery service is retrieved from our peer group, the `netPeerGroup` [line 31]:

```java
discovery = netPeerGroup.getDiscoveryService();
```

This discovery service will be used later to add ourselves as a DiscoveryListener for DiscoveryResponse events and to send DiscoveryRequest messages.

**run() method**

The `run()` method first adds the calling object as a DiscoveryListener for DiscoveryResponse events [line 42]:

```java
discovery.AddDiscoveryListener(this);
```

Now, whenever a Discovery Response message is received, the `discoveryEvent()` method for this object will be called. This enables our application to asynchronously be notified every time this JXTA peer receives a Discovery Response message.

Next, the `run()` method loops forever sending out DiscoveryRequest messages via the `getRemoteAdvertisements()` method. The `getRemoteAdvertisements()` method takes 5 arguments:

- `java.lang.string peerid` — ID of a peer to send query to; if null, propagate query request
- `int type` — `DiscoveryService.PEER`, `DiscoveryService.GROUP`, `DiscoveryService.ADV`
- `java.lang.string attribute` — attribute name to narrow discovery to
- `java.lang.string value` — value of attribute to narrow discovery to
- `int threshold` — the upper limit of responses from one peer

There are two main ways to send DiscoveryRequests via the Discovery Service. If a peer ID is specified in the `getRemoteAdvertisement()` call, the message is sent to only that one peer. In this case, the Endpoint Router attempts to resolve the destination peer’s endpoints locally; if necessary, it routes the message to other relays in an attempt to reach the specified peer. If a null peer ID is specified in the `getRemoteAdvertisements()` call, the discovery message

---

Programming with JXTA  33
is propagated on the local subnet utilizing IP multicast, and the message is also propagated to the rendezvous peer. Only peers in the same peer group will respond to a DiscoveryRequest message.

The type parameter specifies which type of advertisements to look for. The DiscoveryService class defines three constants: DiscoveryService.PEER (looks for peer advertisements), DiscoveryService.GROUP (looks for peer group advertisements), and DiscoveryService.ADV (looks for all other advertisement types, such as pipe advertisements or module class advertisements).

The discovery scope can be narrowed down by specifying an Attribute and Value pair; only advertisements that match will be returned. The Attribute must exactly match an element name in the associated XML document. The Value string can use wildcards (e.g., *) to determine the match. For example, the following call would limit the search to peers whose name contained the exact string "test1":

```java
discovery.getRemoteAdvertisements(null, DiscoveryService.PEER,
       "Name", "test1", 5);
```

while this example, using wildcards, would return any peer whose name contained the string "test":

```java
discovery.getRemoteAdvertisements(null, DiscoveryService.PEER,
       "Name", "*test*", 5);
```

The search can also be limited by specifying a threshold value, indicating the upper limit of responses from one peer.

In our example [line 47], we send Discovery Request messages to the local subnet and the rendezvous peers, looking for any peer. By specifying a threshold value of 5, we will get a maximum of 5 responses (peer advertisements) in each Discovery Response message. If the peer has more than the specified number of matches, it will select the elements to return at random.

```java
discovery.getRemoteAdvertisements(null, DiscoveryService.PEER,
       null, null, 5);
```

There is no guarantee that there will be a response to a DiscoveryRequest message. A peer may receive zero, one, or more responses.

**discoveryEvent() method**

Because our class implements the DiscoveryListener interface, we must have a discoveryEvent() method [line 67]:

```java
public void discoveryEvent(DiscoveryEvent ev)
```

The Discovery Service calls this method whenever a DiscoveryResponse message is received. Peers that have been discovered are automatically added to the local cache (.jxta/cm/group_name) by the Discovery Service.

The first part of this method prints out a message reporting which peer sent the response.

The discoveryEvent method is passed a single argument of type DiscoveryEvent. The getResponse() method returns the response associated with this event. In our example, this method returns a DiscoveryResponseMsg [line 69]:

```java
DiscoveryResponseMsg res = ev.getResponse();
```

Each DiscoveryResponseMsg object contains the responding peer’s peer advertisement, a count of the number of responses returned, and an enumeration of peer advertisements (one for each discovered peer). Our example retrieves the responding peer’s advertisement from the message [line 73]:

```java
PeerAdvertisement peerAdv = res.getPeerAdvertisement();
```

Because some peers may not respond with their peer advertisement, the code checks if the peer advertisement is null. If it is not null, it extracts the responding peer’s name [line 77]:

```java
name = peerAdv.getName();
```

Now we print a message stating we received a response and include the name of the responding peer (or unknown, if the peer did not include its peer advertisement in its response) [line 80]:
System.out.println("Got a Discovery Response [" +
 res.getResponseCount()+ " elements] from peer : " +
 name);

The second part of this method prints out the names of each discovered peer. The responses are returned as an enumeration, and can be retrieved from the DiscoveryResponseMsg [line 86]:

    Enumeration enum = res.getAdvertisements();

Each element in the enumeration is a PeerAdvertisement, and for each element we print the peer’s name [line 90]:

    adv = (PeerAdvertisement) enum.nextElement();
    System.out.println(" Peer name = " + adv.getName());

main()

The main() method [starting at line 96] first creates a new object of class DiscoveryDemo. It then calls the startJxta() method, which instantiates the JXTA platform. Finally, it calls the run() method, which loops continuously sending out discovery requests.