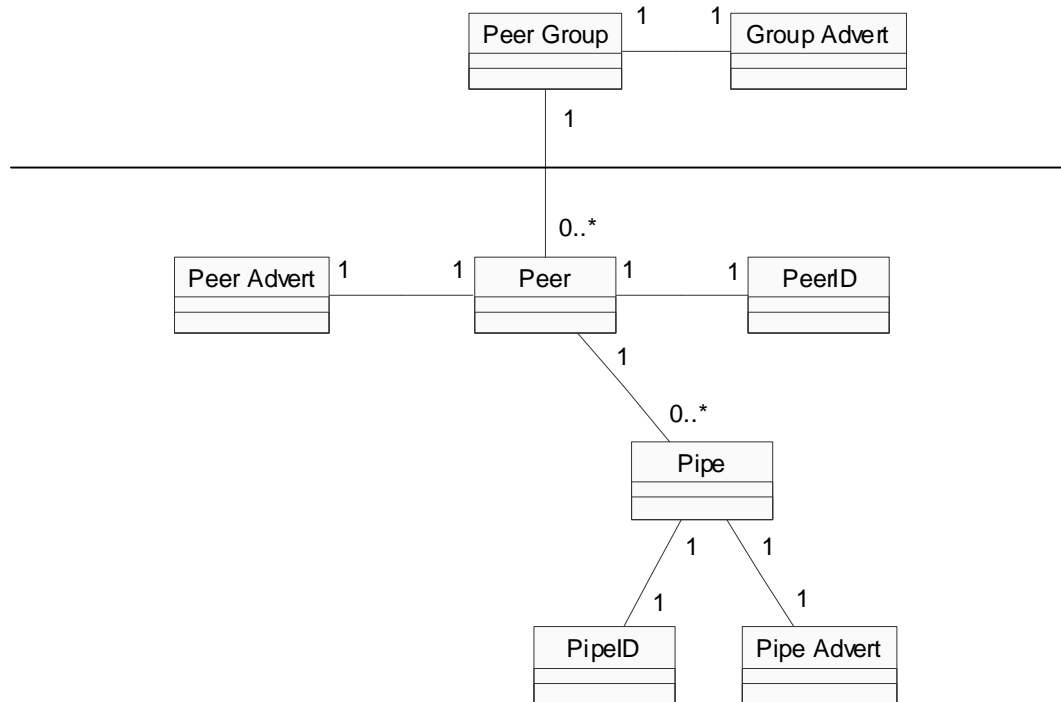


JXTA Basics

The following diagram represents a basic understanding of a peer on the JXTA P2P network. I shall now briefly explain what each of these entities are for and also how to implement a peer using the JXTA API.



Peer groups are logical groups of like-minded peers, napster and its file sharing capabilities would typically represent one such group. A peer group advertises its existence with a group advertisement.

Peers can apply for membership and join peer groups; each peer has a unique PeerID and can advertise itself with a Peer Advertisement.

Pipes are used to join two peer endpoints together; pipes can be one-to-one (unicast) or one-to-many (multicast).

To create a peer that is capable of giving some form of service the following steps can be used, later in the document we shall connect to this service with another peer application;-

1. Make an attempt to join the default peer group *netPeerGroup*.

```
static PeerGroup group = null;
```

```
group = PeerGroupFactory.newNetPeerGroup();
```

2. Now attempt to interface with the discovery and pipe services.

```
private DiscoveryService discoSvc;  
private PipeService pipeSvc;  
  
discoSvc = group.getDiscoveryService();  
pipeSvc = group.getPipeService();
```

3. Instantiate a module advertisement for the class and initialise it with some general descriptors and of course a unique ID, this will simply publicise the service so others can know of its existence.

```
ModuleClassAdvertisement mcadv = (ModuleClassAdvertisement)  
    AdvertisementFactory.newAdvertisement(  
        ModuleClassAdvertisement.getAdvertisementType());  
  
mcadv.setName("JXTAMOD:SOMENAME");  
mcadv.setDescription("A Program");  
  
ModuleClassID mcID = IDFactory.newModuleClassID();  
mcadv.setModuleClassID(mcID);
```

4. And now publish this advert locally (in a cache that JXTA creates and maintains), and remotely (to other peers that have been found and registered in the cache).

```
discoSvc.publish(mcadv, DiscoveryService.ADV);  
discoSvc.remotePublish(mcadv, DiscoveryService.ADV);
```

5. We now have to do something useful with the service we have created, things like how others can actually access it. This is achieved by creating a module spec advertisement, contained within is the pipe advert (using XML – see end of doc) for the service; we are loading this from a file so it remains static upon each execution of the service. An ID is created for this also.

```
PipeAdvertisement pipeadv = null;  
ModuleSpecAdvertisement mdadv = (ModuleSpecAdvertisement)  
    AdvertisementFactory.newAdvertisement(  
        ModuleSpecAdvertisement.getAdvertisementType());  
  
mdadv.setName("SOMESERVICE");  
mdadv.setVersion("Version 1.0");  
mdadv.setCreator("somewhere.com");  
mdadv.setModuleSpecID(IDFactory.newModuleSpecID(mcID));  
mdadv.setSpecURI("http://www.somewhere.com");  
  
try {  
    FileInputStream is = new FileInputStream("pipeadvert.adv");
```

```

        pipeadv = (PipeAdvertisement)
            AdvertisementFactory.newAdvertisement(
                new MimeMediaType("text/xml"), is);
        is.close();
    }
    catch (java.io.IOException e) {
        System.out.println("failed to read/parse pipe advertisement");
        e.printStackTrace();
        System.exit(-1);
    }

    mdadv.setPipeAdvertisement(pipeadv);

```

6. Now we can publish the advertisement both locally and remotely and create an input pipe from the advertisement so we can begin to accept incoming connections.

```

private InputPipe myPipe;

discoSvc.publish(mdadv, DiscoveryService.ADV);
discoSvc.remotePublish(mdadv, DiscoveryService.ADV);
myPipe = pipeSvc.createInputPipe(pipeadv);

```

7. And finally we have to catch incoming connections, typically within a loop.

```

private Message msg;

msg = myPipe.waitForMessage();

```

Now we have seen how to create a simple service peer on the JXTA network we can now connect to this service using the following steps.

1. Create a different application and follow steps 1 and 2 from the above instructions (i.e. join the *netPeerGroup* and get handles on the *discovery* and *pipe* services).
2. The next step is actually finding out where the other peer is and getting hold of its advertisement, this is done by searching for the service name in our local cache ("JXTAMOD:SOMENAME"), if it is not found then we send a discovery message out into the JXTA network, hopefully at some point it will be found and cached locally so we can retrieve it. Discovery messages are asynchronous so we send the message out and try looking in the cache again at some point in the future.

```

Enumeration enum = null;
while (true) {

```

```

        enum = discoSvc.getLocalAdvertisements(DiscoveryService.ADV,
            "Name", "JXTAMOD:SOMENAME");
        if ((enum != null) && enum.hasMoreElements()) break;
        discoSvc.getRemoteAdvertisements(null, DiscoveryService.ADV,
            "Name", "JXTAMOD:SOMENAME", 1, null);
    }

```

```

ModuleSpecAdvertisement mdsadv =
    (ModuleSpecAdvertisement) enum.nextElement();

```

3. Now we can actually connect and send some data to the other peer that's providing some service. We do this by creating a message to hold out data and then an output pipe; this output pipe is then bound to the input pipe of the other peer. The 'createOutputPipe' call is best implemented within a loop, as the connection may not occur upon the first call.

```

private PipeAdvertisement serverpipeadv;
private OutputPipe ServerPipe;
private Message msg;

serverpipeadv = mdsadv.getPipeAdvertisement();

msg = pipeSvc.createMessage();
msg.setString("DATA", "Hi there");

ServerPipe = pipeSvc.createOutputPipe(serverpipeadv, 20000);

ServerPipe.send(msg);

ServerPipe.close();

```

And that's it, we have created a peer that can offer a potential service to another peer and we have then created another peer and sent some data to the first one. There remains one or two questions left unanswered though, things like;-

1. How does the server peer extract the data that has been sent to it?
2. How does the sever peer send anything back to the client?

In step 7 of creating the server peer any incoming data is dumped into 'msg'. To extract this data we first need to know what tag to look out for. The client peer sent a message to the server peer in step 3 above, it packaged the data "Hi there" and tagged it as "DATA", so extraction becomes simple;-

```

String str;

if(msg.hasElement("DATA")){
    str = msg.getString("DATA");
}

```

now to make the client accessible to the server all we have to do is create an *'InputPipe'* on the client and send the pipe advertisement (converted to a string) in the message to the server peer.

```
msg = pipeSvc.createMessage();
msg.setString("DATA", "Hi there");
msg.setString("PIPEADVERT", "Put the advert in here");
```

the server can extract this from the other end, reform it (see code at end of doc) into a pipe advert and use it to connect to the client peer.

Example Pipe Advert – File 'pipeadvert.adv'

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
  <Id>
    urn:jxta:uuid-
    9CCCDF5AD8154D3D87A391210404E59BE4B888209A2241A4A162A10916074A9504
  </Id>
  <Type>
    JxtaUnicast
  </Type>
  SOMENAME
  </Name>
</jxta:PipeAdvertisement>
```

Converts a string to an advertisement

```
private Advertisement convertStringToAd(String AdString){
  Advertisement ad = null;
  ByteArrayInputStream AdStream = new
    ByteArrayInputStream(AdString.getBytes());
  try {
    ad = AdvertisementFactory.newAdvertisement(new
      MimeMediaType("text/xml"), AdStream);
  }
  catch (IOException ex) {
    // Recasting exceptions is not really a good idea, but we do it.
  }
  return (Advertisement)ad;
} // end of convert string to ad
```