Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com, etc.

An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.

Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

IP Address

- An IP address helps to identify a specific resource on the network using a numerical representation.
- Most networks combine IP with TCP (Transmission Control Protocol). It builds a virtual bridge among the destination and the source.

Java InetAddress Class Methods

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	It returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	It returns the instance of InetAdddress containing local host name and address.
public String getHostName()	It returns the host name of the IP address.
public String getHostAddress()	It returns the IP address in string format.

Example of Java InetAddress Class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

InetDemo.java

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");
System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

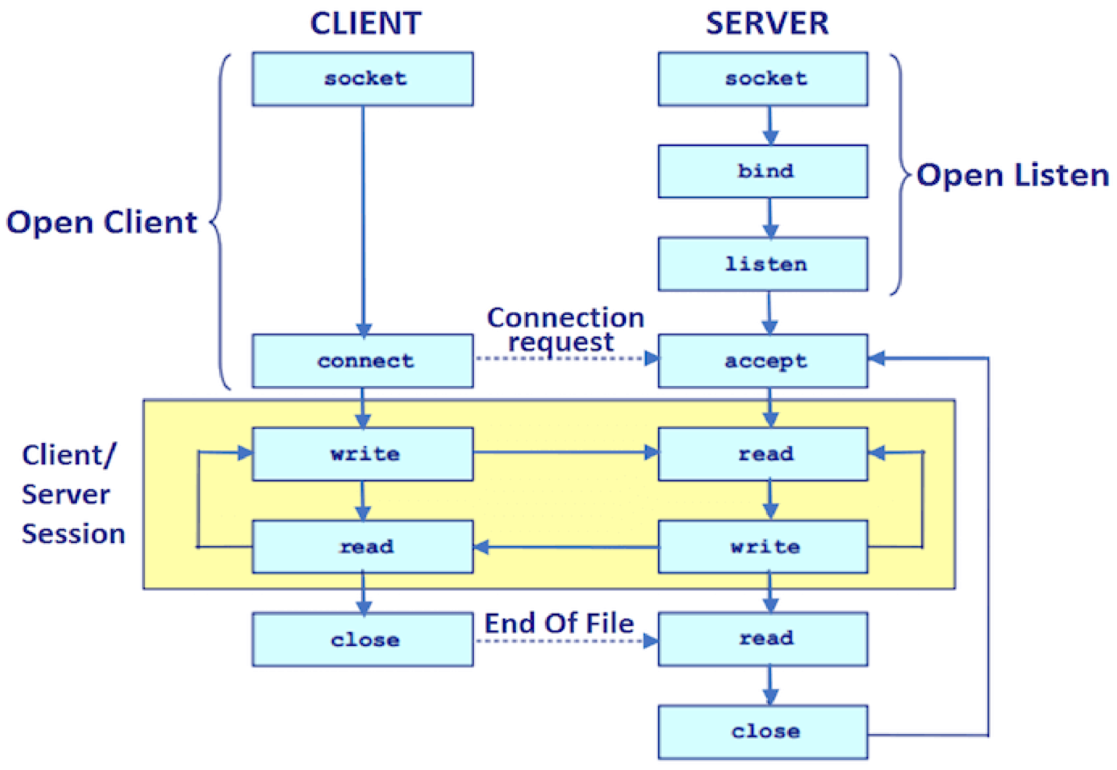
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



SOCKET API

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Method	**Description**
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Method	**Description**
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
1. ServerSocket ss=new ServerSocket(6666);
2. Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

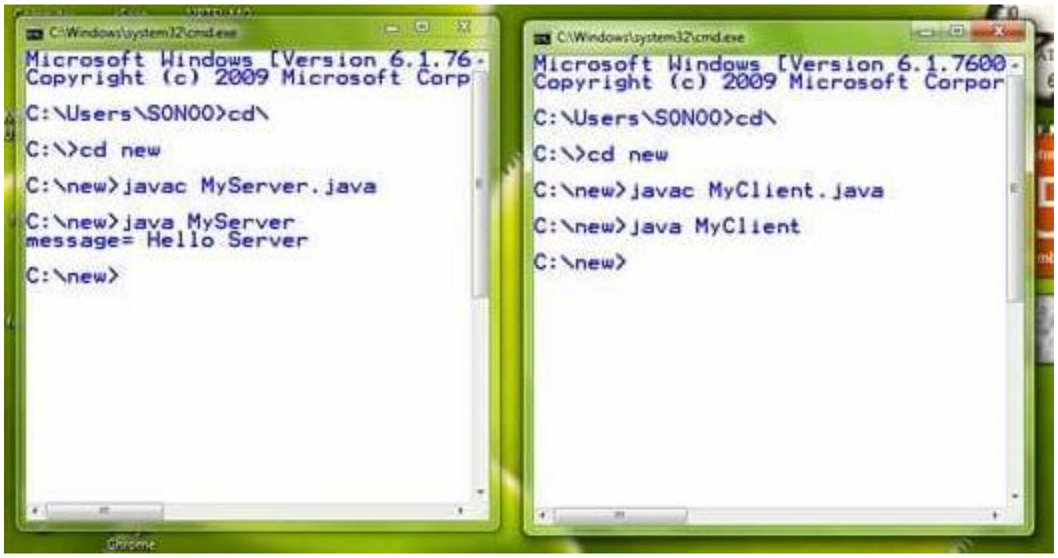
```
// Server Program
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());

//representation of a Unicode character string encoded in modified UTF-8 format;
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

```
// Client Program
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

In the above program a single message from client (i.e. Hello Server) is displayed on server side.

Example of two communication between client server (Connection Oriented)

Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text.

// Below is server side program

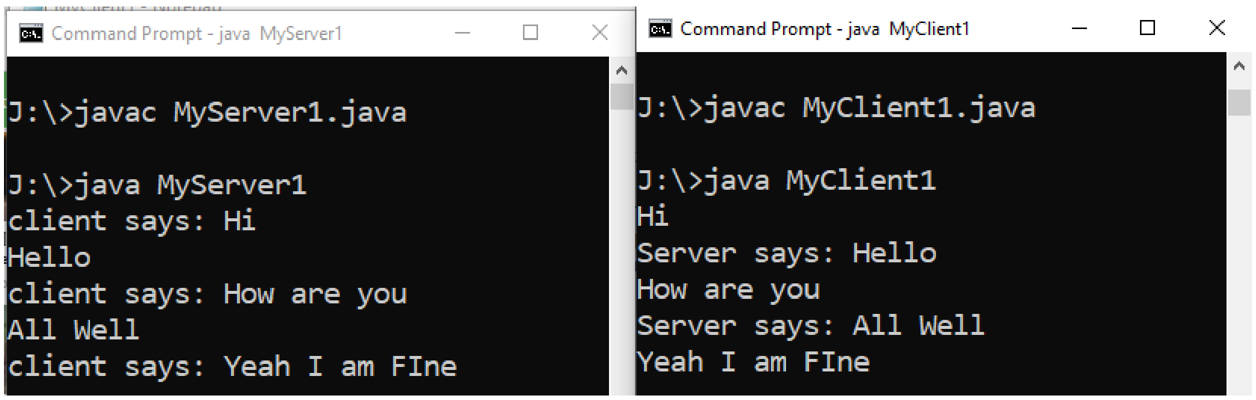
```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

// Below is Client side program

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Output of Client Server Communication



Command Prompt - java MyServer1

```
J:\>javac MyServer1.java

J:\>java MyServer1
client says: Hi
Hello
client says: How are you
All Well
client says: Yeah I am FIne
```

Command Prompt - java MyClient1

```
J:\>javac MyClient1.java

J:\>java MyClient1
Hi
Server says: Hello
How are you
Server says: All Well
Yeah I am FIne
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

Datagram

Datagrams are collection of information sent from one device to another device via the established network. When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if the datagram received is damaged or not. The UDP protocol is used to implement the datagrams in Java.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. It is a mechanism used for transmitting datagram packets over network.`
A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramSocket Class

Method	Description
void bind(SocketAddress addr)	It binds the DatagramSocket to a specific address and port.
void close()	It closes the datagram socket.
void connect(InetAddress address, int port)	It connects the socket to a remote address for the socket.
void disconnect()	It disconnects the socket.

InetAddress getInetAddress()	It returns the address to where the socket is connected.
InetAddress getLocalAddress()	It gets the local address to which the socket is connected.
int getLocalPort()	It returns the port number on the local host to which the socket is bound.
int getPort()	It returns the port number to which the socket is connected.
void send(DatagramPacket p)	It sends the datagram packet from the socket.
void receive(DatagramPacket p)	It receives the datagram packet from the socket.

Java DatagramPacket Class

Java DatagramPacket is a message that can be sent or received. It is a data container. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] b, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] b, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Java DatagramPacket Class Methods

Method	**Description**
InetAddress getAddress()	Returns the IP address of the machine to which the datagram is being sent or from which the datagram was received.
byte[] getData()	Returns the data buffer.
int getLength()	Returns length of the data to be sent or the length of the data received.
int getOffset()	Returns offset of the data to be sent or the offset of the data received.
int getPort()	It returns the port number on the remote host to which the datagram is being sent or from which the datagram was received.
void setAddress(InetAddress iaddr)	Sets the IP address of machine to which the datagram is being sent.
void setData(byte[] buff)	It sets the data buffer for the packet.

void setLength(int length)	It sets the length of the packet.
void setPort(int iport)	It sets the port number on the remote host to which the datagram is being sent.

The programs shown below implement client server communication where a clients the message sent from server.

// Program to set up a server in connectionless networking

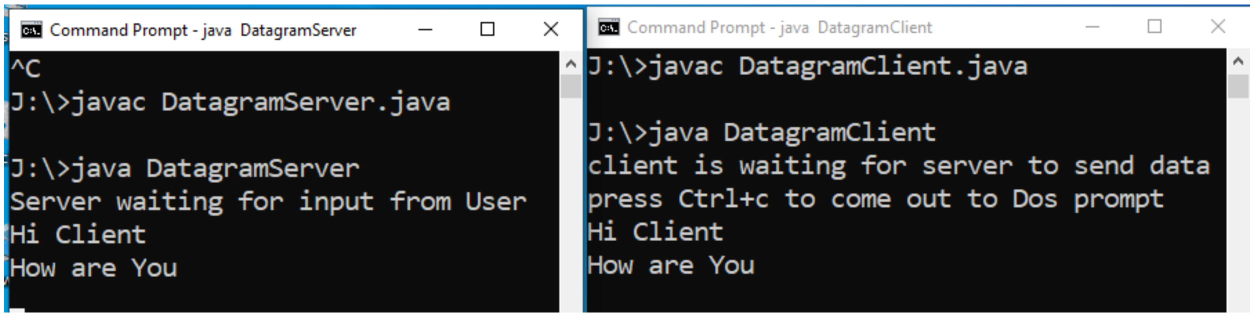
Below is the Server Program

```
import java.net.*;
import java.io.*;
class DatagramServer
{
public static DatagramSocket ds;
public static int clientport=789,serverport=790;
public static void main(String args[]) throws Exception
{
byte buffer[]=new byte[1024];
ds=new DatagramSocket(serverport);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Server waiting for input from User");
InetAddress ia=InetAddress.getByName("localhost");
while(true)
{
String str1=br.readLine();
if(str1==null || str1.equals("end"))
break;
buffer=str1.getBytes();
ds.send(new DatagramPacket(buffer,str1.length(),ia,clientport));
}
}
}
```

Below is Client Program

```
import java.net.*;
import java.io.*;
class DatagramClient
{
public static DatagramSocket ds;
public static byte buffer[]=new byte[1024];
public static int clientport=789,serverport=790;
public static void main(String args[]) throws Exception
{
ds=new DatagramSocket(clientport);
System.out.println("client is waiting for server to send data");
System.out.println("press Ctrl+c to come out to Dos prompt");
while(true)
{
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);
String st=new String(p.getData(),0,p.getLength());
System.out.println(st);
}
}
}
```

Output on Two Command Prompt instances


Command Prompt - java DatagramServer
Command Prompt - java DatagramClient

```
^C
J:\>javac DatagramServer.java

J:\>java DatagramServer
Server waiting for input from User
Hi Client
How are You
```

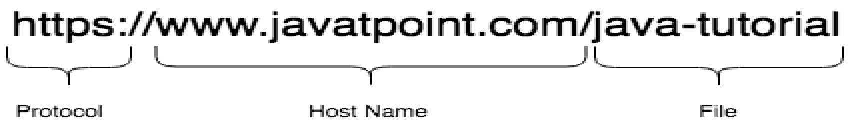
```
J:\>javac DatagramClient.java

J:\>java DatagramClient
client is waiting for server to send data
press Ctrl+c to come out to Dos prompt
Hi Client
How are You
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

1. https://www.javatpoint.com/java-tutorial



A URL contains many information:

1. **Protocol:** In this case, https is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/java-tutorial/, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, java-tutorial is the directory name.

Constructors of Java URL class

URL(String spec)

Creates an instance of a URL from the String representation.

URL(String protocol, String host, int port, String file)

Creates an instance of a URL from the given protocol, host, port number, and file.

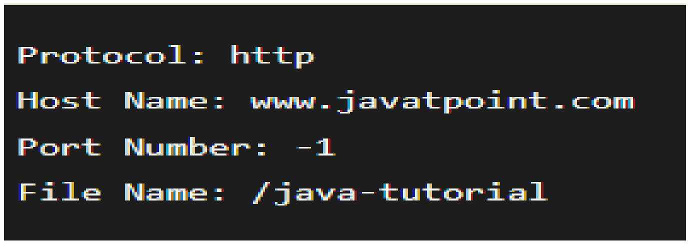
Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	**Description**
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public int getDefaultPort()	public int getDefaultPort()
public String getAuthority()	**public String getAuthority()**

Example of Java URL class

```
//URLDemo.java
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}
```



```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

```
//URLDemo.java
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("https://www.google.com/search?q=javatpoint&oq=javatpoint&sourceid
=chrome&ie=UTF-8");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("Default Port Number: "+url.getDefaultPort());
System.out.println("Query String: "+url.getQuery());
System.out.println("Path: "+url.getPath());
System.out.println("File: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}
```

```
Protocol: https
Host Name: www.google.com
Port Number: -1
Default Port Number: 443
Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
Path: /search
File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
```

URLConnection Class

The **Java URLConnection** class represents a communication link between the URL and the application. It can be used to read and write data to the specified resource referred by the URL.

Features of URLConnection class

1. URLConnection is an abstract class. The two subclasses HttpURLConnection and JarURLConnection makes the connetion between the client Java program and URL resource on the internet.
2. With the help of URLConnection class, a user can read and write to and from any resource referenced by an URL object.
3. Once a connection is established and the Java program has an URLConnection object, we can use it to read or write or get further information like content length, etc.

Constructors

Constructor	**Description**
1) protected URLConnection(URL url)	It constructs a URL connection to the specified URL.

URLConnection Class Methods

Method	**Description**
void addRequestProperty(String key, String value)	It adds a general request property specified by a key-value pair
void connect()	It opens a communications link to the resource referenced by this URL, if such a connection has not already been established.
int getConnectionTimeout()	It returns setting for connect timeout.
Object getContent()	It retrieves the contents of the URL connection.
Object getContent(Class[] classes)	It retrieves the contents of the URL connection.