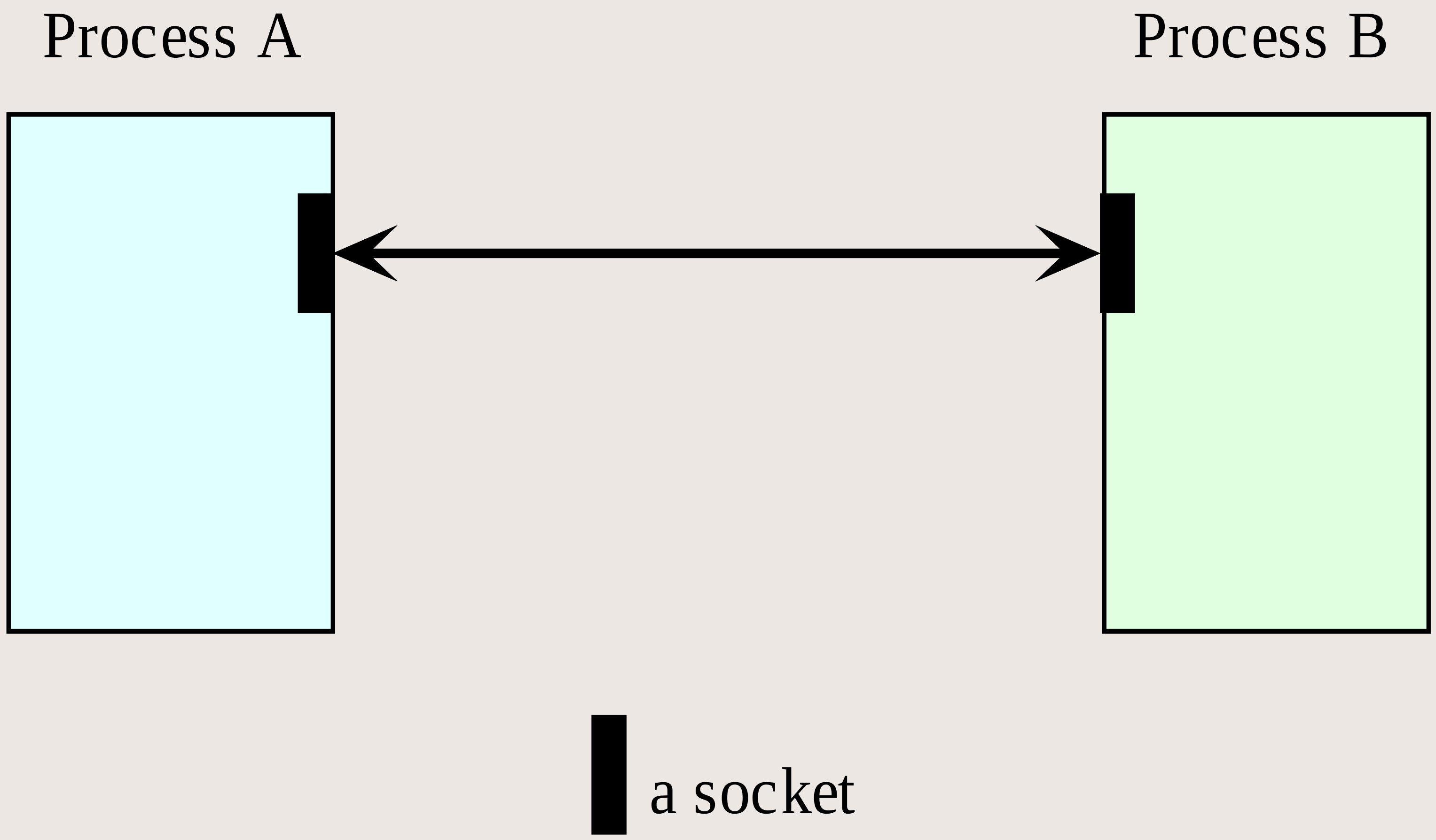
The Socket API

Mei-Ling Liu

Introduction

- The socket API is an Interprocessing Communication (IPC) programming interface originally provided as part of the Berkeley UNIX operating system.
- It has been ported to all modern operating systems, including Sun Solaris and Windows systems.
- It is a *de facto* standard for programming IPC, and is the basis of more sophisticated IPC interface such as remote procedure call and remote method invocation.

The conceptual model of the socket API



The socket API

- A socket API provides a programming construct termed a socket. A process wishing to communicate with another process must create an instance, or instantiate, such a construct
- The two processes then issues operations provided by the API to send and receive data.

Connection-oriented & connectionless datagram socket

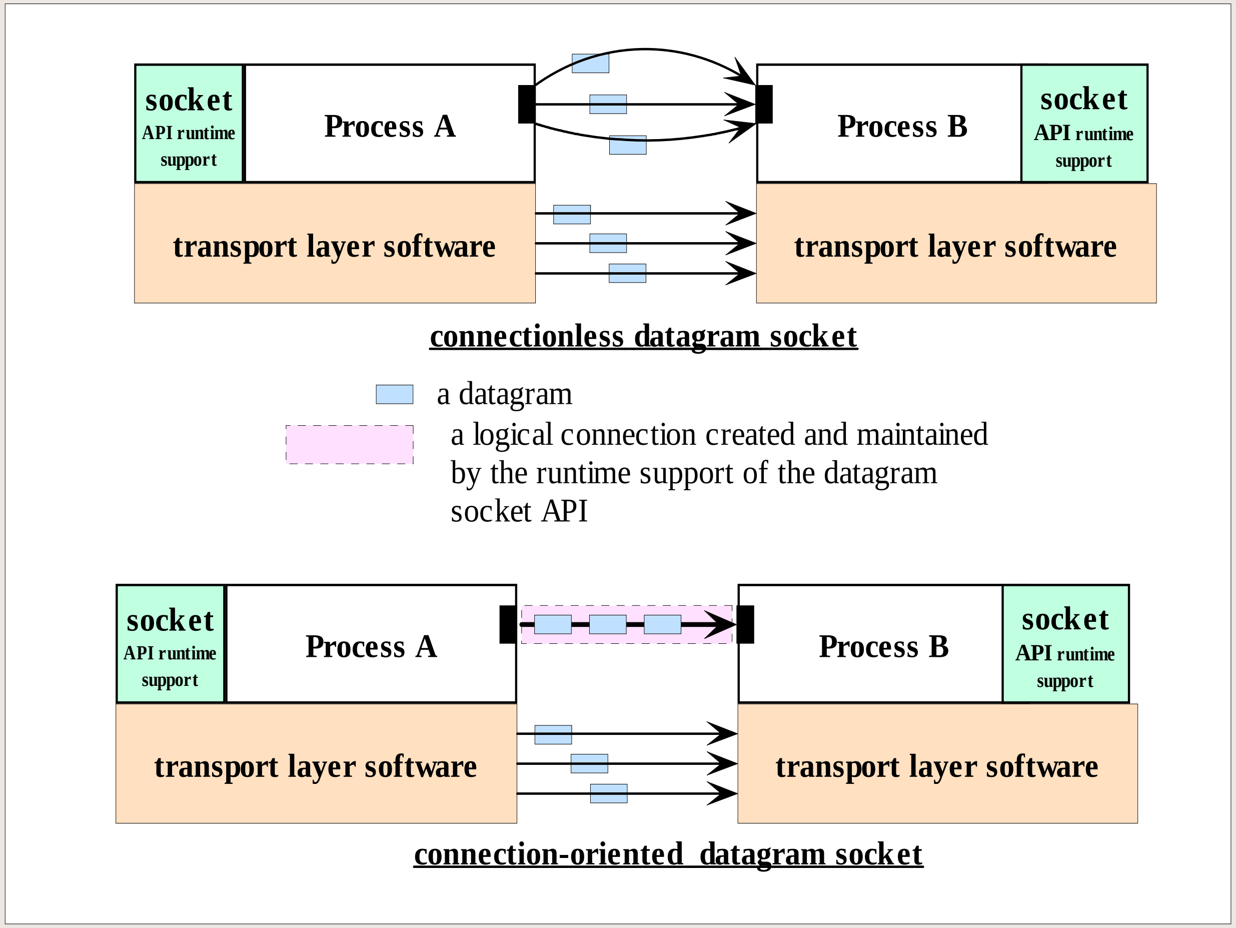
- A socket programming construct can make use of either the UDP or TCP protocol.
- Sockets that use UDP for transport are known as *datagram sockets*, while sockets that use TCP are termed *stream sockets*.
- Because of its relative simplicity, we will first look at datagram sockets, returning to stream sockets after we have introduced the client-server model in Chapter 5.

Connection-oriented & connectionless datagram socket

Datagram sockets can support both **connectionless** and **connection-oriented** communication at the application layer. This is so because even though datagrams are sent or received without the notion of connections at the transport layer, the **runtime support of the socket API** can create and maintain logical connections for datagrams exchanged between two processes, as you will see in the next section.

(The runtime support of an API is a set of software that is bound to the program during execution in support of the API.)

Connection-oriented & connectionless datagram socket



The Java Datagram Socket API

In Java, two classes are provided for the datagram socket API:

1. the ***DatagramSocket*** class for the sockets.
2. the *DatagramPacket* class for the datagram exchanged.

A process wishing to send or receive data using this API must instantiate a DatagramSocket object, or a socket in short. Each socket is said to be *bound* to a UDP port of the machine local to the process

The Java Datagram Socket API

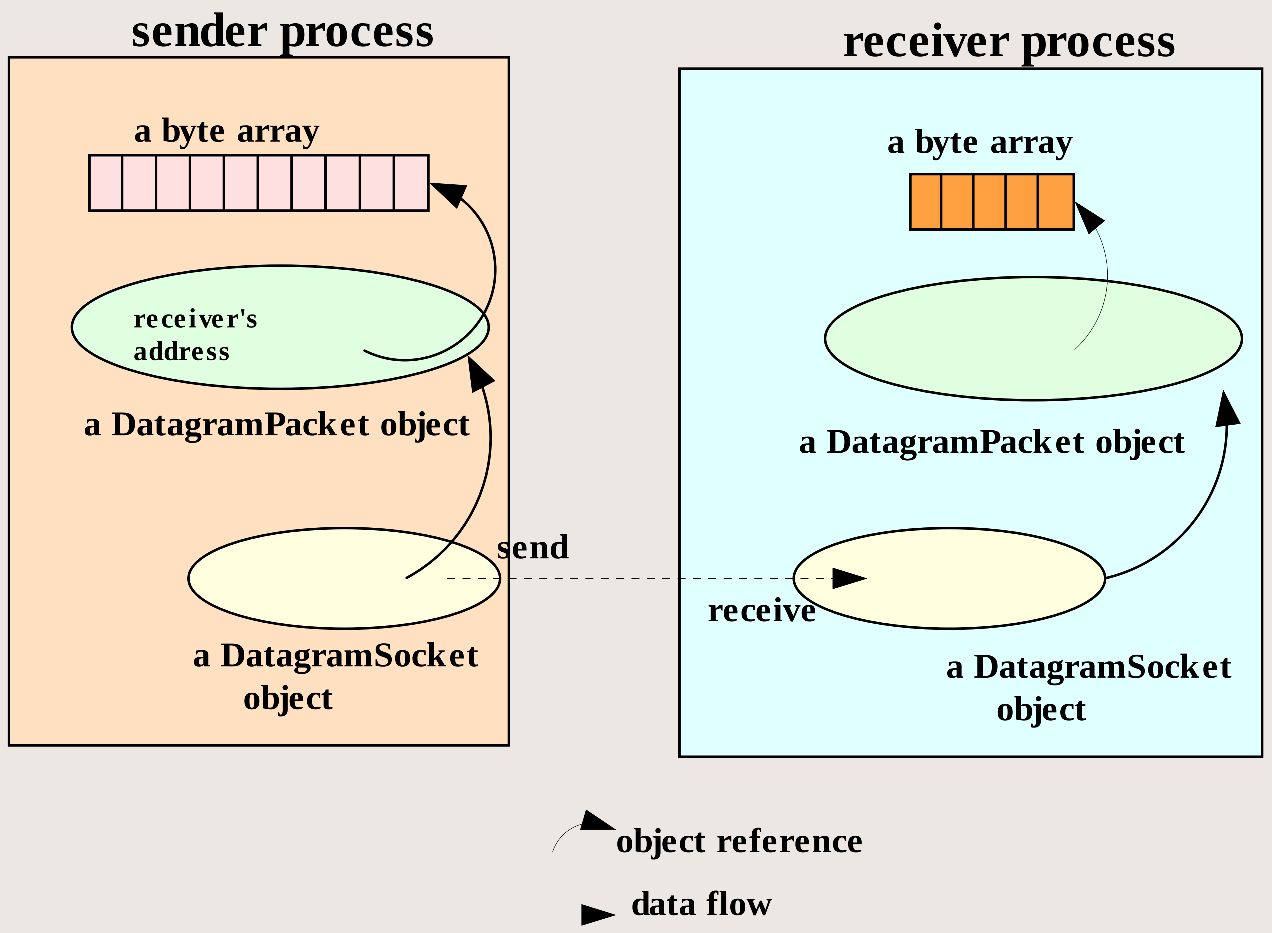
To send a datagram to another process, a process:

- creates an object that represents the datagram itself. This object can be created by instantiating a ***DatagramPacket*** object which carries
  1. (i) the payload data as a reference to a byte array, and
  2. (ii) the destination address (the host ID and port number to which the receiver’s socket is bound.
- issues a call to a *send* method in the *DatagramSocket* object, specifying a reference to the *DatagramPacket* object as an argument

The Java Datagram Socket API

- In the receiving process, a *DatagramSocket* object must also be instantiated and bound to a local port, the port number must agree with that specified in the datagram packet of the sender.
- To receive datagrams sent to the socket, the process creates a *datagramPacket* object which references a byte array and calls a *receive* method in its *DatagramSocket* object, specifying as argument a reference to the *DatagramPacket* object.

The Data Structures in the sender and receiver programs



The program flow in the sender and receiver programs

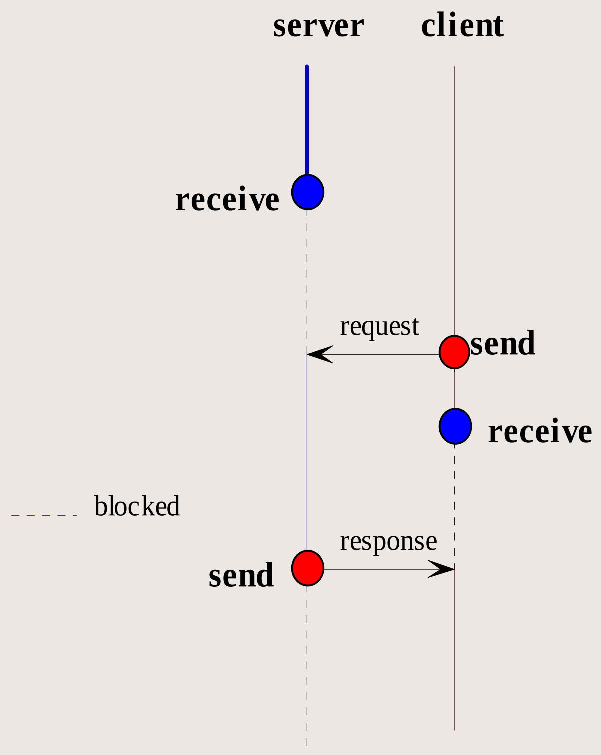
sender program

```
create a datagram socket and
  bind it to any local port;
place data in a byte array;
create a datagram packet, specifying
 the data array and the receiver's
 address;
invoke the send method of the
 socket with a reference to the
 datagram packet;
```

receiver program

```
create a datagram socket and
  bind it to a specific local port;
create a byte array for receiving the data;
create a datagram packet, specifying
 the data array;
invoke the receive method of the
 socket with a reference to the
 datagram packet;
```

Event synchronization with the connectionlss datagram socketsAPI



blocking receive,
nonblocking send
in a request-response
protocol

if data is sent before a corresponding receive operation is issued, the data will be discarded by the runtime support and will not be received.

Setting timeout

To avoid indefinite blocking, a timeout can be set with a socket object:

void **setSoTimeout**(int timeout)

Set a timeout for the blocking receive from this socket, in milliseconds.

Once set, the timeout will be in effect for all blocking operations.

Key Methods and Constructors

Method/Constructor	Description
DatagramPacket(byte[ ] buf, int length)	Construct a datagram packet for receiving packets of length *length;* data received will be stored in the byte array reference by *buf*.
DatagramPacket (byte[ ] buf, int length, InetAddress address, int port)	Construct a datagram packet for sending packets of length *length* to the socket bound to the specified port number on the specified host *;* data received will be stored in the byte array reference by *buf*.
DatagramSocket ( )	Construct a datagram socket and binds it to any available port on the local host machine; this constructor can be used for a process that sends data and does not need to receive data.
DatagramSocket (int port)	Construct a datagram socket and binds it to the specified port on the local host machine; the port number can then be specified in a datagram packet sent by a sender.
void close( )	Close this datagramSocket object
void receive(DatagramPacket p)	Receive a datagram packet using this socket.
void send (DatagramPacket p)	Send a datagram packet using this socket.
void setSoTimeout(int timeout)	Set a timeout for the blocking receive from this socket, in milliseconds.

The coding

```
//Excerpt from a receiver program
DatagramSocket ds = new DatagramSocket(2345);
DatagramPacket dp =
      new DatagramPacket(buffer, MAXLEN);
ds.receive(dp);
len = dp.getLength( );
System.out.Println(len + " bytes received.\n");
String s = new String(dp.getData( ), 0, len);
System.out.println(dp.getAddress( ) + " at port "
   + dp.getPort( ) + " says " + s);
```

```
// Excerpt from the sending process
InetAddress receiverHost=
    InetAddress.getByName("localHost");
DatagramSocket theSocket = new DatagramSocket( );
String message = "Hello world!";
byte[ ] data = message.getBytes( );
data = theLine.getBytes( );
DatagramPacket thePacket
    = new DatagramPacket(data, data.length,
                              receiverHost, 2345);
theSocket.send(theOutput);
```

Connectionless sockets

With connectionless sockets, it is possible for multiple processes to simultaneously send datagrams to the same socket established by a receiving process, in which case the order of the arrival of these messages will be unpredictable, in accordance with the UDP protocol

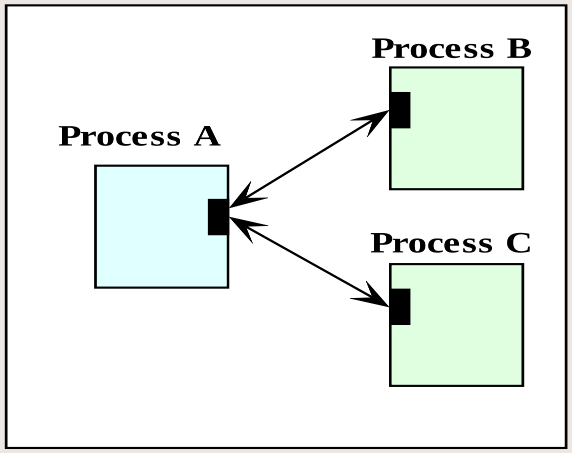
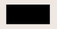


Figure 3a



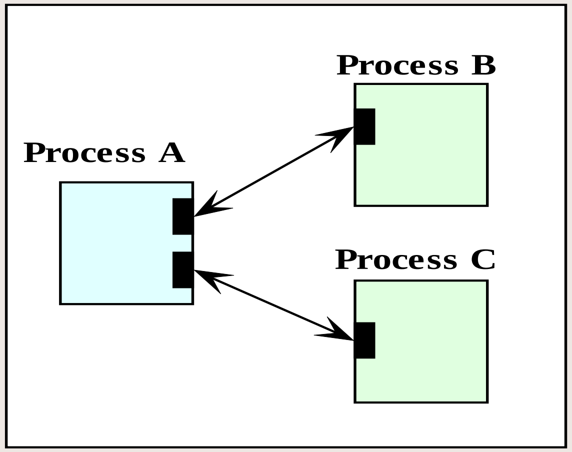


Figure 3b

Code samples

- Example1Sender.java, ExampleReceiver.java
- MyDatagramSocket.java, Example2SenderReceiver.java , Example2ReceiverSender.java

Connection-oriented datagram socket API

It is uncommon to employ datagram sockets for connection-oriented communication; the connection provided by this API is rudimentary and typically insufficient for applications that require a true connection. Stream-mode sockets, which will be introduced later, are more typical and more appropriate for connection-oriented communication.

Methods calls for connection-oriented datagram socket

Method/Constructor	**Description**
public void **connect**(InetAddress address, int port)	Create a logical connection between this socket and a socket at the rem ote address and port.
public void **disconnect**( )	Cancel the current connection, if any, from this socket.

A connection is made for a socket with a remote socket. Once a socket is connected, it can only exchange data with the remote socket.
If a datagram specifying another address is sent using the socket, an IllegalArgumentException will occur.
If a datagram from another socket is sent to this socket, The data will be ignored.

Connection-oriented Datagram Socket

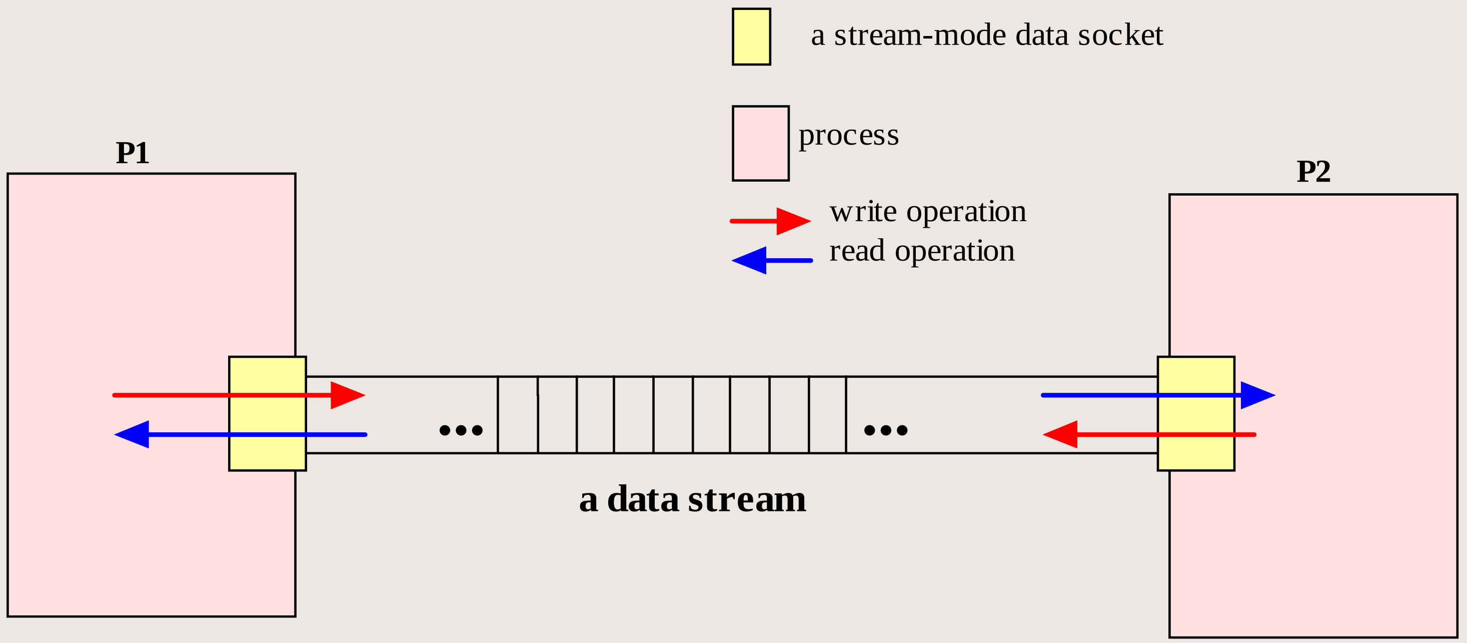
The connection is **unilateral**, that is, it is enforced only on one side. The socket on the other side is free to send and receive data to and from other sockets, unless it too commits to a connection to the other socket.

See Example3Sender, Example3Receiver.

The Stream-mode Socket API

- The **datagram socket** API supports the exchange of discrete units of data (that is, datagrams).
- the **stream socket** API provides a model of data transfer based on the stream-mode I/O of the Unix operating systems.
- By definition, a stream-mode socket supports connection-oriented communication only.

Stream-mode Socket API (connection-oriented socket API)



Stream-mode Socket API

- A stream-mode socket is established for data exchange between two specific processes.
- Data stream is written to the socket at one end, and read from the other end.
- A stream socket cannot be used to communicate with more than one process.

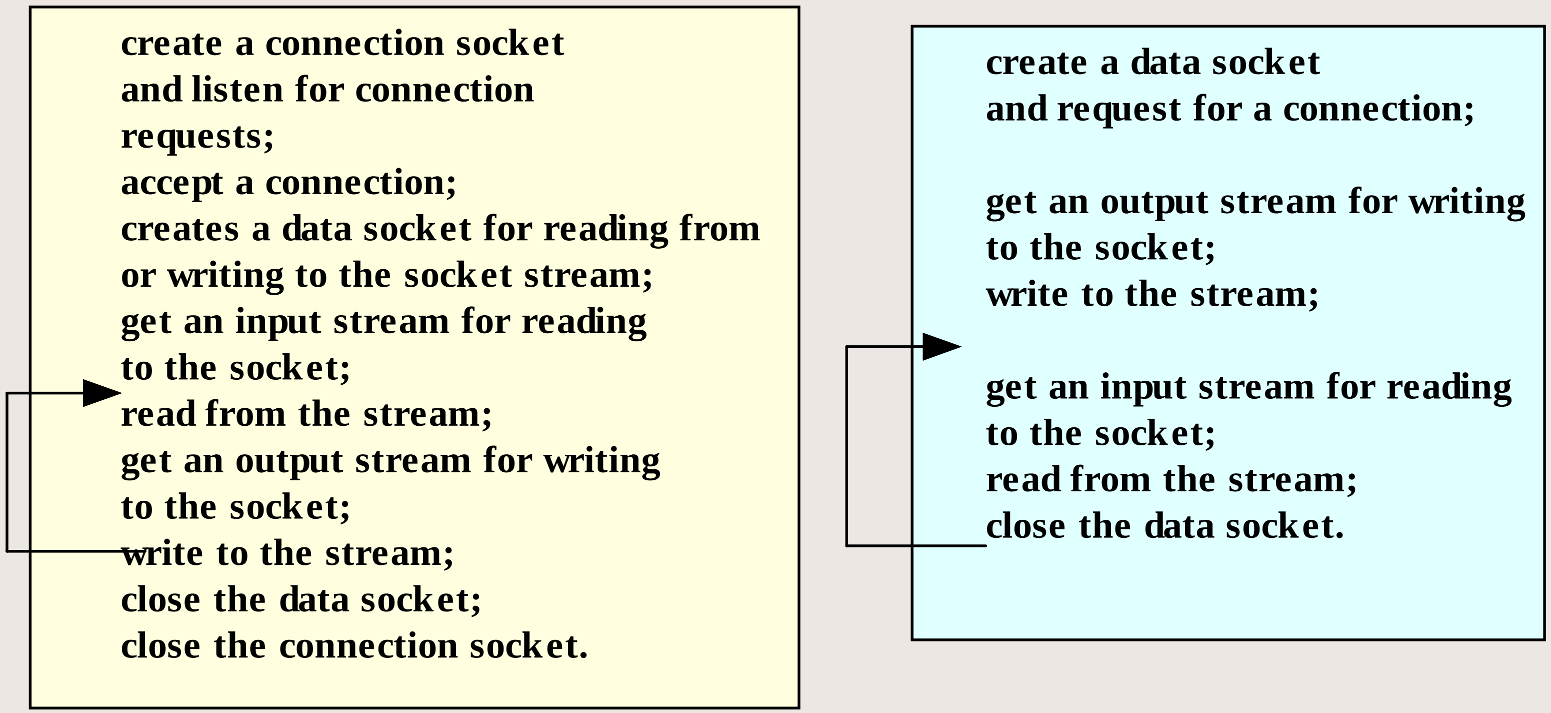
Stream-mode Socket API

In Java, the stream-mode socket API is provided with two classes:

- Server socket: for accepting connections; we will call an object of this class a connection socket.
- Socket: for data exchange; we will call an object of this class a data socket.

Stream-mode Socket API program flow

connection listener (server)



create a connection socket
and listen for connection
requests;
accept a connection;
creates a data socket for reading from
or writing to the socket stream;
get an input stream for reading
to the socket;
read from the stream;
get an output stream for writing
to the socket;
write to the stream;
close the data socket;
close the connection socket.

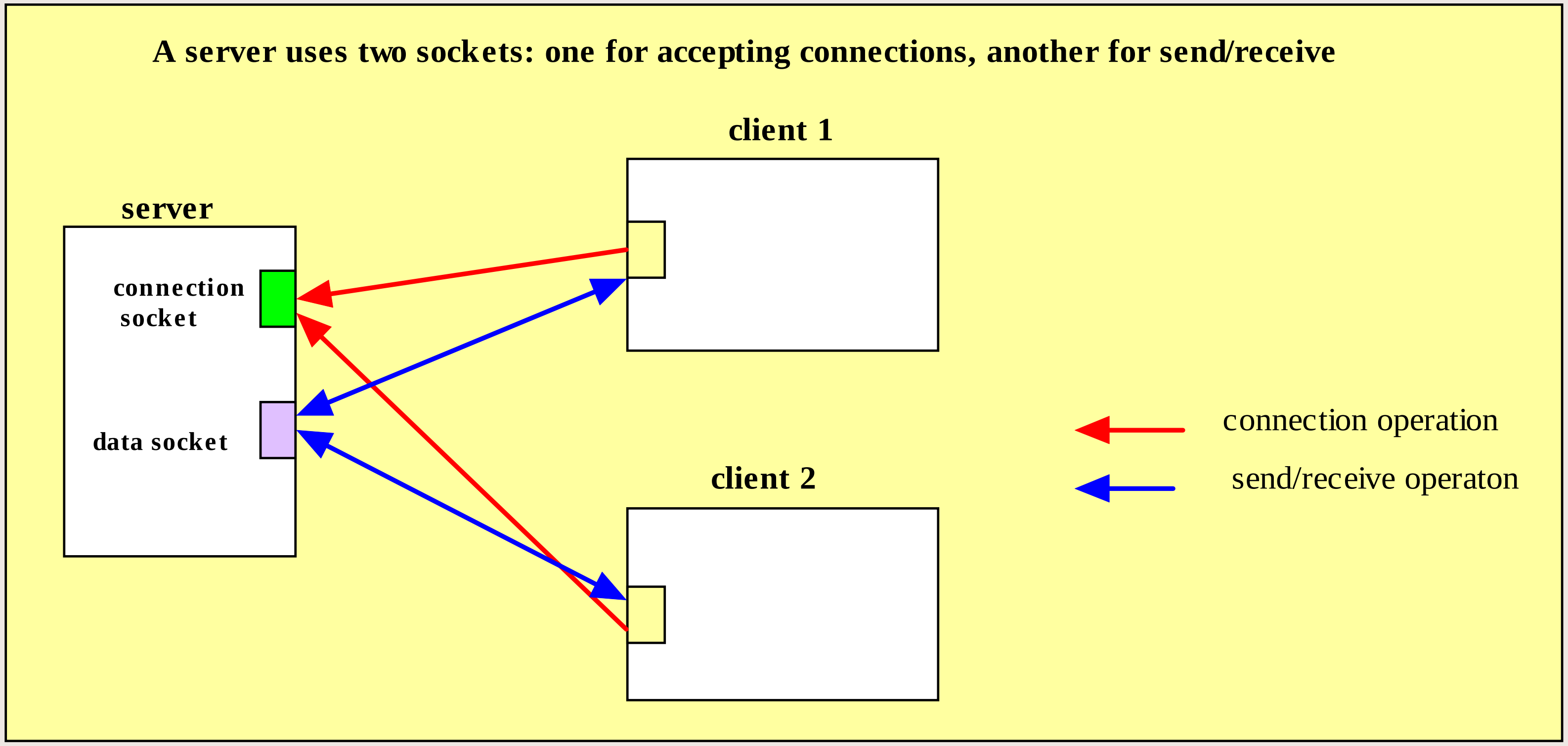
connection requester (server)

create a data socket
and request for a connection;

get an output stream for writing
to the socket;
write to the stream;

get an input stream for reading
to the socket;
read from the stream;
close the data socket.

The server (the connection listener)



Key methods in the ServerSocket class

Method/constructor	Description
ServerSocket(int port)	Creates a server socket on a specified port.
Socket accept() throws IOException	Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
public void close() throws IOException	Closes this socket.
void setSoTimeout(int timeout) throws SocketException	Set a timeout period (in milliseconds) so that a call to accept( ) for this socket will block for only this amount of time. If the timeout expires, a java.io.InterruptedIOException is raised

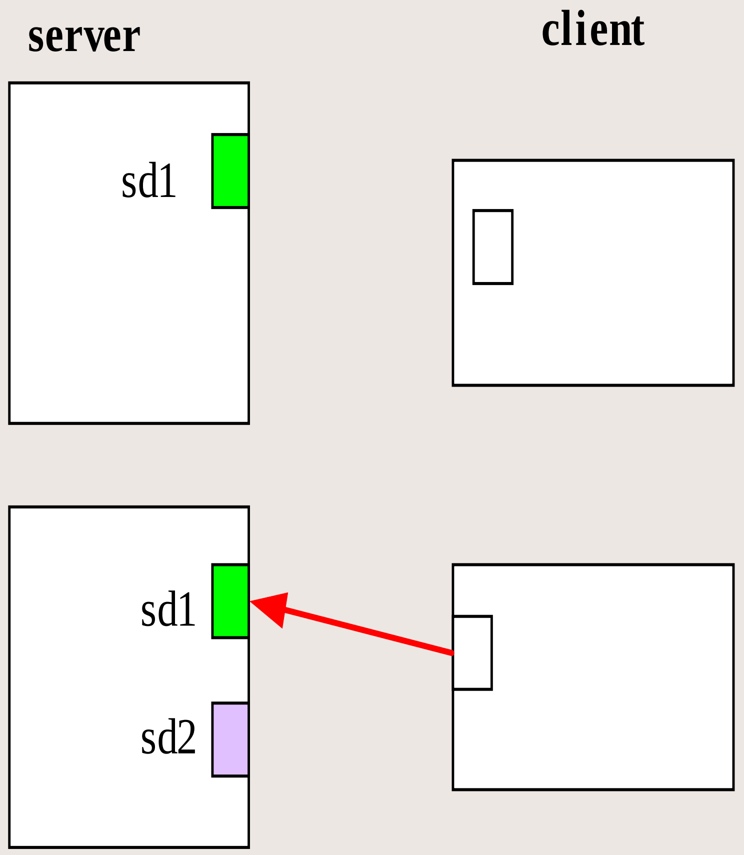
Note: Accept is a blocking operation.

Key methods in the Socket class

Method/constructor	**Description**
Socket(InetAddress address, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address
void close() throws IOException	Closes this socket.
InputStream getInputStream( ) throws IOException	Returns an input stream so that data may be read from this socket.
OutputStream getOutputStream( )throws IOException	Returns an output stream so that data may be written to this socket.
void **setSoTimeout**(int timeout) throws SocketException	Set a timeout period for blocking so that a read( ) call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised

A read operation on the InputStream is blocking.
A write operation is nonblocking.

Connection-oriented socket API-3



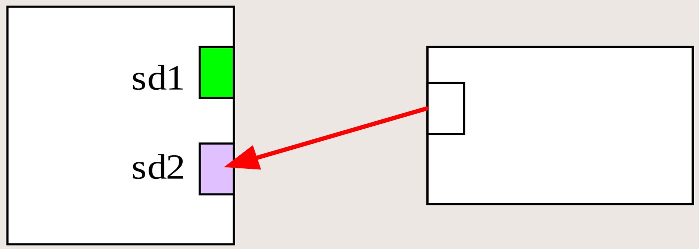
1. Server establishes a socket sd1 with local address, then listens for incoming connection on sd1

Client establishes a socket with remote (server's) address.

2. Server accepts the connection request and creates a new socket sd2 as a result.

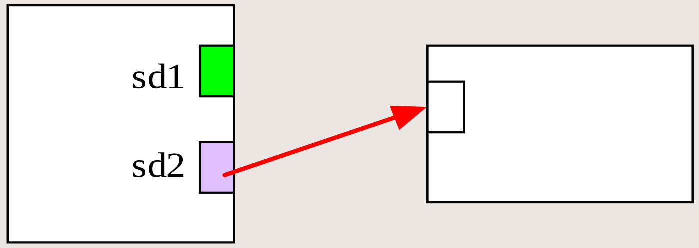
Connection-oriented socket API-3

3. Server issues receive operation using sd2.

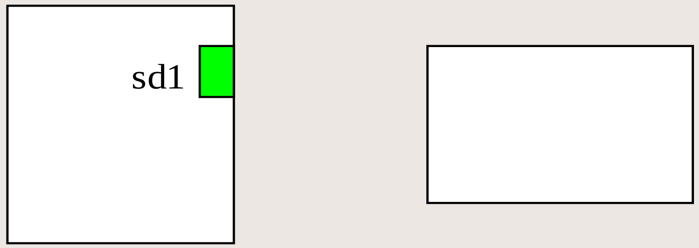


Client issues send operation.

4. Server sends response using sd2.

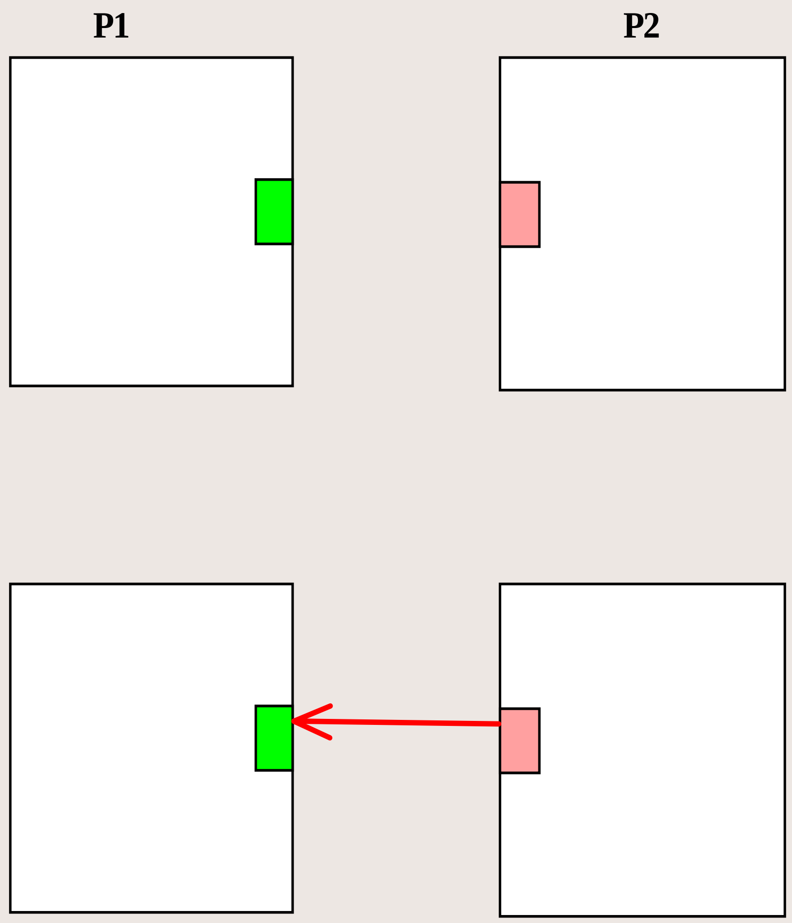


5. When the protocol has completed, server closes sd2; sd1 is used to accept the next connection



Client closes its socket when the protocol has completed

Connectionless socket API



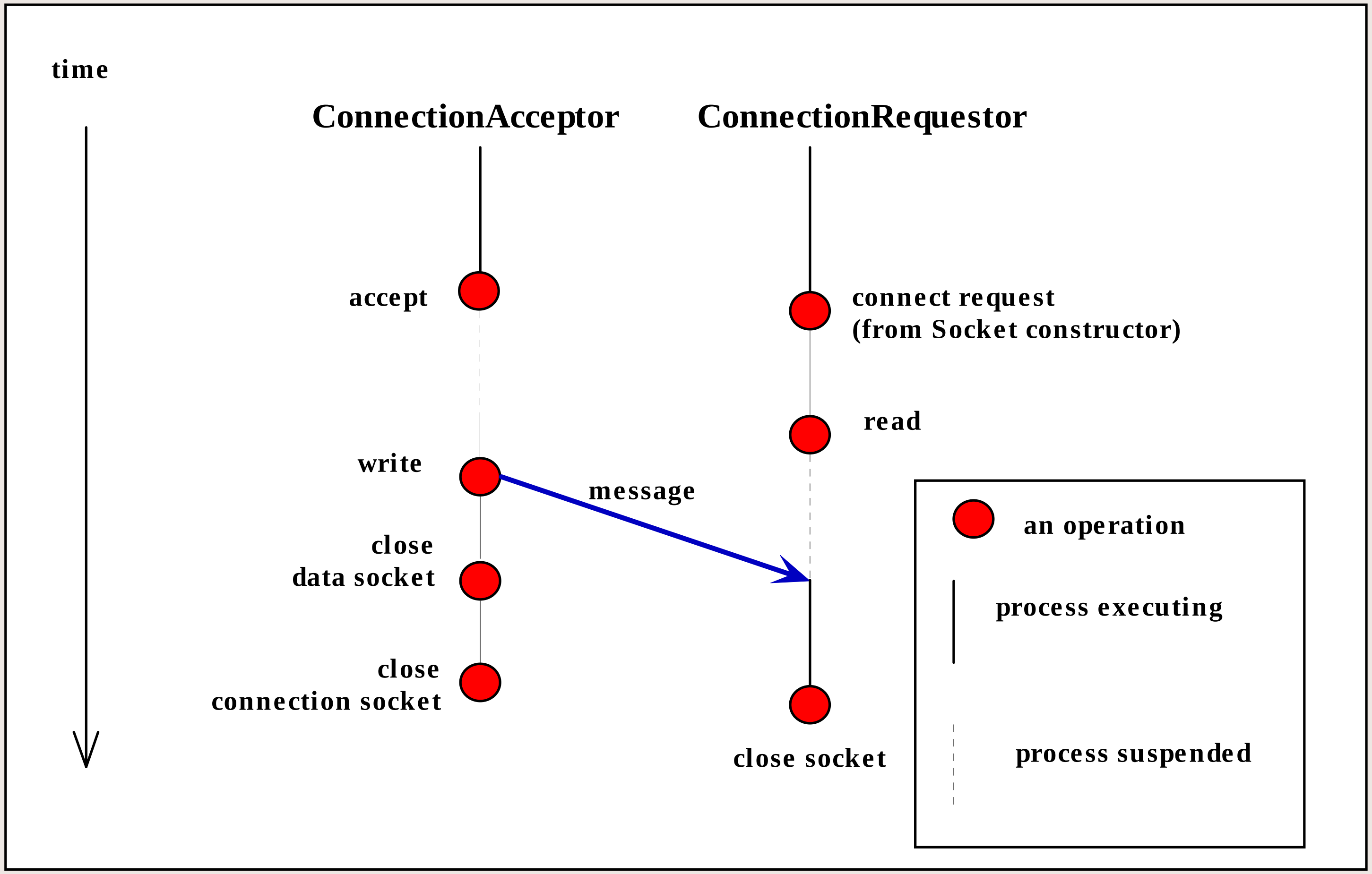
P1 establishes a local socket

P2 establishes a local socket

P1 issues a receive operation to receive the datagram.

P2 sends a datagram addressed to P1

Example 4 Event Diagram



Example4

Example4ConnectionAcceptor

```
try {
   nt portNo;
   String message;
   // instantiates a socket for accepting connection
    ServerSocket connectionSocket = new ServerSocket(portNo);
   Socket dataSocket = connectionSocket.accept();

   // get a output stream for writing to the data socket
   OutputStream outStream = dataSocket.getOutputStream();
   // create a PrinterWriter object for character-mode output
   PrintWriter socketOutput =
      new PrintWriter(new OutputStreamWriter(outStream));
   // write a message into the data stream
   socketOutput.println(message);
   //The ensuing flush method call is necessary for the data to
   // be written to the socket data stream before the
   // socket is closed.
   socketOutput.flush();
   dataSocket.close( );
   connectionSocket.close( );
 } // end try
catch (Exception ex) {
 System.out.println(ex);
}
```

Example4ConnectionReceiver

```
 try {
      InetAddress acceptorHost =
       InetAddress.getByName(args[0]);
       int acceptorPort = Integer.parseInt(args[1]);
    // instantiates a data socket
       Socket mySocket = new Socket(acceptorHost, acceptorPort);
    // get an input stream for reading from the data socket
        InputStream inStream = mySocket.getInputStream();
    // create a BufferedReader object for text-line input
      BufferedReader socketInput =
        new BufferedReader(new InputStreamReader(inStream));
    // read a line from the data stream
      String message = socketInput.readLine( );
      System.out.println("\t" + message);
      mySocket.close( );
} // end try
catch (Exception ex) {
    System.out.println(ex);
}
```

Secure Sockets

http://java.sun.com/products/jsse/

- **Secure sockets perform encryption on the data transmitted.**
- **The Java^{TM} Secure Socket Extension (JSSE) is a Java package that enables secure Internet communications.**
- **It implements a Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols**
- **It includes functionalities for data encryption, server authentication, message integrity, and optional client authentication.**
- **Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol.**

The Java Secure Socket Extension API

http://java.sun.com/products/jsse/doc/apidoc/index.html

- **Import javax.net.ssl;**
- **Class SSLServerSocket is a subclass of ServerSocket, and inherits all its methods.**
- **Class SSLSocket is a subclass of Socket, and inherits all its methods.**
- **There are also classes for**
  - **Certification**
  - **Handshaking**
  - **KeyManager**
  - **SSLsession**

Summary

- In this chapter, we introduced the socket application program interface for interprocess communication.
- The socket API is widely available as a programming facility for IPC at a relatively low level of abstraction.

Summary - 2

- Using the Java socket APIs, we introduced two types of sockets:
  - The datagram sockets, which uses the User Datagram Protocol (UDP) at the transport layer to provide the sending and receiving of discrete data packets known as datagrams.
  - The stream-mode socket, which uses the Tranport Layer Protocol (TCP) at the transport layer to provide the sending and receiving of data using a data stream.

Summary - 3

Key points of the Java datagram socket API:

- It supports both connectionless communication and connection-oriented communication.
- Each process must create a DatagramSocket object to send/receive datagrams.
- Each datagram is encapsulated in a DatagramPacket object.

Summary - 4

Key points of the Java datagram socket API continued:

- In connectionless communication, a datagram socket can be used to send to or receive from any other datagram socket; in connection-oriented communication, a datagram socket can only be used to send to or receive from the datagram socket attached to the other end of the connection.
- Data for a datagram are placed in a byte array; if a byte array of insufficient length is provided by a receiver, the data received will be truncated.
- The receive operation is blocking; the send operation is non-blocking.

Summary - 5

Key points of the Java stream-mode socket API:

- It supports connection-oriented communication only.
- A process plays the role of connection-acceptor, and creates a connection socket using the ServerSocket class. It then accepts connection requests from other processes.
- A process (a connection-requestor) creates a data socket using the Socket class, the constructor of which issues a connection request to the connection-acceptor.

Summary - 6

More key points of the Java stream-mode socket API :

- When a connection request is granted, the connection-acceptor creates a data socket, of the Socket class, to send and receive data to/from the connection-requestor. The connection-requestor can also send and/or receive data to/from the connection-acceptor using its data socket.
- The receive (read), the connection-accept operation, and the connection-request operations are blocking; the send (write) operation is non-blocking.
- The reading and writing of data into the socket of data stream are decoupled: they can be performed in different data units. See Figure 8.

Summary - 7

Key points of secure sockets:

- Secure socket APIs are available for transmission of confidential data.
- Well known secure socket APIs include the Secure Socket Layer (SSL) and Java's Secure Socket Extension (JSSE). Secure socket APIs have methods that are similar to the connection-oriented socket APIs.