java.net

Class DatagramSocket

```
java.lang.Object
  └ java.net.DatagramSocket
```

Direct Known Subclasses:
MulticastSocket

```
public class DatagramSocket
extends Object
```

This class represents a socket for sending and receiving datagram packets.

A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

UDP broadcasts sends are always enabled on a DatagramSocket. In order to receive broadcast packets a DatagramSocket should be bound to the wildcard address. In some implementations, broadcast packets may also be received when a DatagramSocket is bound to a more specific address.

Example: `DatagramSocket s = new DatagramSocket(null); s.bind(new InetSocketAddress(8888));` Which is equivalent to: `DatagramSocket s = new DatagramSocket(8888);` Both cases will create a DatagramSocket able to receive broadcasts on UDP port 8888.

Since:
JDK1.0
See Also:
`DatagramPacket`, `DatagramChannel`

Constructor Summary

	`DatagramSocket``()` Constructs a datagram socket and binds it to any available port on the local host machine.
protected	**`DatagramSocket`**`(DatagramSocketImpl impl)` Creates an unbound datagram socket with the specified DatagramSocketImpl.
	`DatagramSocket``(int port)` Constructs a datagram socket and binds it to the specified port on the local host machine.

	DatagramSocket(int port, InetAddress laddr) Creates a datagram socket, bound to the specified local address.
	DatagramSocket(SocketAddress bindaddr) Creates a datagram socket, bound to the specified local socket address.

Method Summary

void	**bind**(SocketAddress addr) Binds this DatagramSocket to a specific address & port.
void	**close**() Closes this datagram socket.
void	**connect**(InetAddress address, int port) Connects the socket to a remote address for this socket.
void	**connect**(SocketAddress addr) Connects this socket to a remote socket address (IP address + port number).
void	**disconnect**() Disconnects the socket.
boolean	**getBroadcast**() Tests if SO_BROADCAST is enabled.
DatagramChannel	**getChannel**() Returns the unique DatagramChannel object associated with this datagram socket, if any.
InetAddress	**getInetAddress**() Returns the address to which this socket is connected.
InetAddress	**getLocalAddress**() Gets the local address to which the socket is bound.
int	**getLocalPort**() Returns the port number on the local host to which this socket is bound.
SocketAddress	**getLocalSocketAddress**() Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
int	**getPort**() Returns the port for this socket.
int	**getReceiveBufferSize**() Get value of the SO_RCVBUF option for this DatagramSocket, that is the buffer size used by the platform for input on this DatagramSocket.
SocketAddress	**getRemoteSocketAddress**() Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	**getReuseAddress**() Tests if SO_REUSEADDR is enabled.
int	**getSendBufferSize**() Get value of the SO_SNDBUF option for this DatagramSocket, that is the buffer size used by the platform for output on this DatagramSocket.
int	**getSoTimeout**()

	Retrive setting for SO_TIMEOUT.
int	**getTrafficClass**() Gets traffic class or type-of-service in the IP datagram header for packets sent from this DatagramSocket.
boolean	**isBound**() Returns the binding state of the socket.
boolean	**isClosed**() Returns whether the socket is closed or not.
boolean	**isConnected**() Returns the connection state of the socket.
void	**receive**(DatagramPacket p) Receives a datagram packet from this socket.
void	**send**(DatagramPacket p) Sends a datagram packet from this socket.
void	**setBroadcast**(boolean on) Enable/disable SO_BROADCAST.
static void	**setDatagramSocketImplFactory**(DatagramSocketImplFactory fac) Sets the datagram socket implementation factory for the application.
void	**setReceiveBufferSize**(int size) Sets the SO_RCVBUF option to the specified value for this DatagramSocket.
void	**setReuseAddress**(boolean on) Enable/disable the SO_REUSEADDR socket option.
void	**setSendBufferSize**(int size) Sets the SO_SNDBUF option to the specified value for this DatagramSocket.
void	**setSoTimeout**(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
void	**setTrafficClass**(int tc) Sets traffic class or type-of-service octet in the IP datagram header for datagrams sent from this DatagramSocket.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DatagramSocket

```
public DatagramSocket()
               throws SocketException
```

Constructs a datagram socket and binds it to any available port on the local host machine. The socket will be bound to the wildcard address, an IP address chosen by the kernel.

If there is a security manager, its `checkListen` method is first called with 0 as its argument to ensure the operation is allowed. This could result in a SecurityException.

Throws:

`SocketException` - if the socket could not be opened, or the socket could not bind to the specified local port.

`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.

See Also:

`SecurityManager.checkListen(int)`

DatagramSocket

```
protected DatagramSocket(DatagramSocketImpl impl)
```

Creates an unbound datagram socket with the specified DatagramSocketImpl.

Parameters:

`impl` - an instance of a **DatagramSocketImpl** the subclass wishes to use on the DatagramSocket.

Since:

1.4

DatagramSocket

```
public DatagramSocket(SocketAddress bindaddr)
              throws SocketException
```

Creates a datagram socket, bound to the specified local socket address.

If, if the address is `null`, creates an unbound socket.

If there is a security manager, its `checkListen` method is first called with the port from the socket address as its argument to ensure the operation is allowed. This could result in a SecurityException.

Parameters:

`bindaddr` - local socket address to bind, or `null` for an unbound socket.

Throws:

`SocketException` - if the socket could not be opened, or the socket could not bind to the specified local port.

`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.

Since:

1.4

See Also:

`SecurityManager.checkListen(int)`

DatagramSocket

```
public DatagramSocket(int port)
               throws SocketException
```

Constructs a datagram socket and binds it to the specified port on the local host machine. The socket will be bound to the wildcard address, an IP address chosen by the kernel.

If there is a security manager, its `checkListen` method is first called with the `port` argument as its argument to ensure the operation is allowed. This could result in a SecurityException.

Parameters:
`port` - port to use.
Throws:
`SocketException` - if the socket could not be opened, or the socket could not bind to the specified local port.
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.
See Also:
`SecurityManager.checkListen(int)`

DatagramSocket

```
public DatagramSocket(int port,
                      InetAddress laddr)
               throws SocketException
```

Creates a datagram socket, bound to the specified local address. The local port must be between 0 and 65535 inclusive. If the IP address is 0.0.0.0, the socket will be bound to the wildcard address, an IP address chosen by the kernel.

If there is a security manager, its `checkListen` method is first called with the `port` argument as its argument to ensure the operation is allowed. This could result in a SecurityException.

Parameters:
`port` - local port to use
`laddr` - local address to bind
Throws:
`SocketException` - if the socket could not be opened, or the socket could not bind to the specified local port.
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.
Since:
JDK1.1
See Also:
`SecurityManager.checkListen(int)`

Method Detail

bind

```
public void bind(SocketAddress addr)
          throws SocketException
```

Binds this DatagramSocket to a specific address & port.

If the address is `null`, then the system will pick up an ephemeral port and a valid local address to bind the socket.

Parameters:
`addr` - The address & port to bind to.
Throws:
`SocketException` - if any error happens during the bind, or if the socket is already bound.
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.
`IllegalArgumentException` - if addr is a SocketAddress subclass not supported by this socket.
Since:
1.4

---

connect

```
public void connect(InetAddress address,
                    int port)
```

Connects the socket to a remote address for this socket. When a socket is connected to a remote address, packets may only be sent to or received from that address. By default a datagram socket is not connected.

If the remote destination to which the socket is connected does not exist, or is otherwise unreachable, and if an ICMP destination unreachable packet has been received for that address, then a subsequent call to send or receive may throw a PortUnreachableException. Note, there is no guarantee that the exception will be thrown.

A caller's permission to send and receive datagrams to a given host and port are checked at connect time. When a socket is connected, receive and send **will not perform any security checks** on incoming and outgoing packets, other than matching the packet's and the socket's address and port. On a send operation, if the packet's address is set and the packet's address and the socket's address do not match, an IllegalArgumentException will be thrown. A socket connected to a multicast address may only be used to send packets.

Parameters:
`address` - the remote address for the socket
`port` - the remote port for the socket.
Throws:
`IllegalArgumentException` - if the address is null, or the port is out of range.
`SecurityException` - if the caller is not allowed to send datagrams to and receive datagrams from the address and port.
See Also:
`disconnect()`, `send(java.net.DatagramPacket)`, `receive(java.net.DatagramPacket)`

---

connect

```
public void connect(SocketAddress addr)
             throws SocketException
```

Connects this socket to a remote socket address (IP address + port number).

Parameters:
`addr` - The remote address.
Throws:
`SocketException` - if the connect fails
`IllegalArgumentException` - if addr is null or addr is a SocketAddress subclass not supported by this socket
Since:
1.4
See Also:
`connect(java.net.InetAddress, int)`

---

disconnect

```
public void disconnect()
```

Disconnects the socket. This does nothing if the socket is not connected.

See Also:
`connect(java.net.InetAddress, int)`

---

isBound

```
public boolean isBound()
```

Returns the binding state of the socket.

Returns:
true if the socket succesfuly bound to an address
Since:
1.4

---

isConnected

```
public boolean isConnected()
```

Returns the connection state of the socket.

Returns:
true if the socket succesfuly connected to a server

Since:
1.4

---

getInetAddress

```
public InetAddress getInetAddress()
```

Returns the address to which this socket is connected. Returns null if the socket is not connected.

Returns:
the address to which this socket is connected.

---

getPort

```
public int getPort()
```

Returns the port for this socket. Returns -1 if the socket is not connected.

Returns:
the port to which this socket is connected.

---

getRemoteSocketAddress

```
public SocketAddress getRemoteSocketAddress()
```

Returns the address of the endpoint this socket is connected to, or `null` if it is unconnected.

Returns:
a `SocketAddress` representing the remote endpoint of this socket, or `null` if it is not connected yet.
Since:
1.4
See Also:
`getInetAddress()`, `getPort()`, `connect(SocketAddress)`

---

getLocalSocketAddress

```
public SocketAddress getLocalSocketAddress()
```

Returns the address of the endpoint this socket is bound to, or `null` if it is not bound yet.

Returns:
a `SocketAddress` representing the local endpoint of this socket, or `null` if it is not bound yet.
Since:

1.4

See Also:

`getLocalAddress()`, `getLocalPort()`, `bind(SocketAddress)`

---

send

```
public void send(DatagramPacket p)
          throws IOException
```

Sends a datagram packet from this socket. The `DatagramPacket` includes information indicating the data to be sent, its length, the IP address of the remote host, and the port number on the remote host.

If there is a security manager, and the socket is not currently connected to a remote address, this method first performs some security checks. First, if `p.getAddress().isMulticastAddress()` is true, this method calls the security manager's `checkMulticast` method with `p.getAddress()` as its argument. If the evaluation of that expression is false, this method instead calls the security manager's `checkConnect` method with arguments `p.getAddress().getHostAddress()` and `p.getPort()`. Each call to a security manager method could result in a SecurityException if the operation is not allowed.

Parameters:

`p` - the `DatagramPacket` to be sent.

Throws:

`IOException` - if an I/O error occurs.

`SecurityException` - if a security manager exists and its `checkMulticast` or `checkConnect` method doesn't allow the send.

`PortUnreachableException` - may be thrown if the socket is connected to a currently unreachable destination. Note, there is no guarantee that the exception will be thrown.

`IllegalBlockingModeException` - if this socket has an associated channel, and the channel is in non-blocking mode.

See Also:

`DatagramPacket`, `SecurityManager.checkMulticast(InetAddress)`, `SecurityManager.checkConnect(java.lang.String, int)`

---

receive

```
public void receive(DatagramPacket p)
            throws IOException
```

Receives a datagram packet from this socket. When this method returns, the `DatagramPacket`'s buffer is filled with the data received. The datagram packet also contains the sender's IP address, and the port number on the sender's machine.

This method blocks until a datagram is received. The `length` field of the datagram packet object contains the length of the received message. If the message is longer than the packet's length, the message is truncated.

If there is a security manager, a packet cannot be received if the security manager's `checkAccept` method does not allow it.

Parameters:
p - the `DatagramPacket` into which to place the incoming data.
Throws:
`IOException` - if an I/O error occurs.
`SocketTimeoutException` - if setSoTimeout was previously called and the timeout has expired.
`PortUnreachableException` - may be thrown if the socket is connected to a currently unreachable destination. Note, there is no guarantee that the exception will be thrown.
`IllegalBlockingModeException` - if this socket has an associated channel, and the channel is in non-blocking mode.
See Also:
`DatagramPacket`, `DatagramSocket`

---

getLocalAddress

```
public InetAddress getLocalAddress()
```

Gets the local address to which the socket is bound.

If there is a security manager, its `checkConnect` method is first called with the host address and `-1` as its arguments to see if the operation is allowed.

Returns:
the local address to which the socket is bound, or an `InetAddress` representing any local address if either the socket is not bound, or the security manager `checkConnect` method does not allow the operation
Since:
1.1
See Also:
`SecurityManager.checkConnect(java.lang.String, int)`

---

getLocalPort

```
public int getLocalPort()
```

Returns the port number on the local host to which this socket is bound.

Returns:
the port number on the local host to which this socket is bound.

---

setSoTimeout

```
public void setSoTimeout(int timeout)
                  throws SocketException
```

Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a call to receive() for this DatagramSocket will block for only this amount of time. If the timeout expires, a **java.net.SocketTimeoutException** is raised, though

the DatagramSocket is still valid. The option **must** be enabled prior to entering the blocking operation to have effect. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout.

Parameters:
`timeout` - the specified timeout in milliseconds.
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
Since:
JDK1.1
See Also:
`getSoTimeout()`

---

getSoTimeout

```
public int getSoTimeout()
                 throws SocketException
```

Retrive setting for SO_TIMEOUT. 0 returns implies that the option is disabled (i.e., timeout of infinity).

Returns:
the setting for SO_TIMEOUT
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
Since:
JDK1.1
See Also:
`setSoTimeout(int)`

---

setSendBufferSize

```
public void setSendBufferSize(int size)
                       throws SocketException
```

Sets the SO_SNDBUF option to the specified value for this `DatagramSocket`. The SO_SNDBUF option is used by the network implementation as a hint to size the underlying network I/O buffers. The SO_SNDBUF setting may also be used by the network implementation to determine the maximum size of the packet that can be sent on this socket.

As SO_SNDBUF is a hint, applications that want to verify what size the buffer is should call `getSendBufferSize()`.

Increasing the buffer size may allow multiple outgoing packets to be queued by the network implementation when the send rate is high.

Note: If `send(DatagramPacket)` is used to send a `DatagramPacket` that is larger than the setting of SO_SNDBUF then it is implementation specific if the packet is sent or discarded.

Parameters:

`size` - the size to which to set the send buffer size. This value must be greater than 0.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
`IllegalArgumentException` - if the value is 0 or is negative.

See Also:

`getSendBufferSize()`

---

getSendBufferSize

```
public int getSendBufferSize()
                    throws SocketException
```

Get value of the SO_SNDBUF option for this `DatagramSocket`, that is the buffer size used by the platform for output on this `DatagramSocket`.

Returns:

the value of the SO_SNDBUF option for this `DatagramSocket`

Throws:

`SocketException` - if there is an error in the underlying protocol, such as an UDP error.

See Also:

`setSendBufferSize(int)`

---

setReceiveBufferSize

```
public void setReceiveBufferSize(int size)
                          throws SocketException
```

Sets the SO_RCVBUF option to the specified value for this `DatagramSocket`. The SO_RCVBUF option is used by the the network implementation as a hint to size the underlying network I/O buffers. The SO_RCVBUF setting may also be used by the network implementation to determine the maximum size of the packet that can be received on this socket.

Because SO_RCVBUF is a hint, applications that want to verify what size the buffers were set to should call `getReceiveBufferSize()`.

Increasing SO_RCVBUF may allow the network implementation to buffer multiple packets when packets arrive faster than are being received using `receive(DatagramPacket)`.

Note: It is implementation specific if a packet larger than SO_RCVBUF can be received.

Parameters:

`size` - the size to which to set the receive buffer size. This value must be greater than 0.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
`IllegalArgumentException` - if the value is 0 or is negative.

See Also:

`getReceiveBufferSize()`

---

getReceiveBufferSize

```
public int getReceiveBufferSize()
                           throws SocketException
```

Get value of the SO_RCVBUF option for this `DatagramSocket`, that is the buffer size used by the platform for input on this `DatagramSocket`.

Returns:
the value of the SO_RCVBUF option for this `DatagramSocket`
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
See Also:
`setReceiveBufferSize(int)`

---

setReuseAddress

```
public void setReuseAddress(boolean on)
                     throws SocketException
```

Enable/disable the SO_REUSEADDR socket option.

For UDP sockets it may be necessary to bind more than one socket to the same socket address. This is typically for the purpose of receiving multicast packets (See `MulticastSocket`). The `SO_REUSEADDR` socket option allows multiple sockets to be bound to the same socket address if the `SO_REUSEADDR` socket option is enabled prior to binding the socket using `bind(SocketAddress)`.

When a `DatagramSocket` is created the initial setting of `SO_REUSEADDR` is disabled.

The behaviour when `SO_REUSEADDR` is enabled or disabled after a socket is bound (See `isBound()`) is not defined.

Parameters:
`on` - whether to enable or disable the
Throws:
`SocketException` - if an error occurs enabling or disabling the `SO_RESUEADDR` socket option, or the socket is closed.
Since:
1.4
See Also:
`getReuseAddress()`, `bind(SocketAddress)`, `isBound()`, `isClosed()`

---

getReuseAddress

```
public boolean getReuseAddress()
                        throws SocketException
```

Tests if SO_REUSEADDR is enabled.

Returns:
a `boolean` indicating whether or not SO_REUSEADDR is enabled.
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
Since:
1.4
See Also:
`setReuseAddress(boolean)`

---

setBroadcast

```
public void setBroadcast(boolean on)
                  throws SocketException
```

Enable/disable SO_BROADCAST.

Parameters:
`on` - whether or not to have broadcast turned on.
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
Since:
1.4
See Also:
`getBroadcast()`

---

getBroadcast

```
public boolean getBroadcast()
                      throws SocketException
```

Tests if SO_BROADCAST is enabled.

Returns:
a `boolean` indicating whether or not SO_BROADCAST is enabled.
Throws:
`SocketException` - if there is an error in the underlying protocol, such as an UDP error.
Since:
1.4
See Also:
`setBroadcast(boolean)`

---

setTrafficClass

```
public void setTrafficClass(int tc)
                      throws SocketException
```

Sets traffic class or type-of-service octet in the IP datagram header for datagrams sent from this DatagramSocket. As the underlying network implementation may ignore this value applications should consider it a hint.

The tc **must** be in the range `0 <= tc <= 255` or an IllegalArgumentException will be thrown.

Notes:

for Internet Protocol v4 the value consists of an octet with precedence and TOS fields as detailed in RFC 1349. The TOS field is bitset created by bitwise-or'ing values such the following :-

- `IPTOS_LOWCOST (0x02)`
- `IPTOS_RELIABILITY (0x04)`
- `IPTOS_THROUGHPUT (0x08)`
- `IPTOS_LOWDELAY (0x10)`

The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.

Setting bits in the precedence field may result in a SocketException indicating that the operation is not permitted.

for Internet Protocol v6 `tc` is the value that would be placed into the sin6_flowinfo field of the IP header.

Parameters:
`tc` - an `int` value for the bitset.
Throws:
`SocketException` - if there is an error setting the traffic class or type-of-service
Since:
1.4
See Also:
`getTrafficClass()`

---

getTrafficClass

```
public int getTrafficClass()
                    throws SocketException
```

Gets traffic class or type-of-service in the IP datagram header for packets sent from this DatagramSocket.

As the underlying network implementation may ignore the traffic class or type-of-service set using `setTrafficClass(int)` this method may return a different value than was previously set using the `setTrafficClass(int)` method on this DatagramSocket.

Returns:
the traffic class or type-of-service already set
Throws:
`SocketException` - if there is an error obtaining the traffic class or type-of-service value.
Since:
1.4
See Also:
`setTrafficClass(int)`

---

close

```
public void close()
```

Closes this datagram socket.

Any thread currently blocked in {#link receive} upon this socket will throw a `SocketException`.

If this socket has an associated channel then the channel is closed as well.

---

isClosed

```
public boolean isClosed()
```

Returns whether the socket is closed or not.

Returns:
true if the socket has been closed
Since:
1.4

---

getChannel

```
public DatagramChannel getChannel()
```

Returns the unique `DatagramChannel` object associated with this datagram socket, if any.

A datagram socket will have a channel if, and only if, the channel itself was created via the `DatagramChannel.open` method.

Returns:
the datagram channel associated with this datagram socket, or `null` if this socket was not created for a channel
Since:
1.4

---

setDatagramSocketImplFactory

```
public static void setDatagramSocketImplFactory(DatagramSocketImplFactory fac)
                                         throws IOException
```

Sets the datagram socket implementation factory for the application. The factory can be specified only once.

When an application creates a new datagram socket, the socket implementation factory's `createDatagramSocketImpl` method is called to create the actual datagram socket implementation.

Passing `null` to the method is a no-op unless the factory was already set.

If there is a security manager, this method first calls the security manager's `checkSetFactory` method to ensure the operation is allowed. This could result in a SecurityException.

Parameters:

`fac` - the desired factory.

Throws:

`IOException` - if an I/O error occurs when setting the datagram socket factory.
`SocketException` - if the factory is already defined.
`SecurityException` - if a security manager exists and its `checkSetFactory` method doesn't allow the operation.

See Also:

`DatagramSocketImplFactory.createDatagramSocketImpl()`, `SecurityManager.checkSetFactory()`
