java.net

Class MulticastSocket

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

```
public class MulticastSocket
extends DatagramSocket
```

The multicast datagram socket class is useful for sending and receiving IP multicast packets. A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.

A multicast group is specified by a class D IP address and by a standard UDP port number. Class D IP addresses are in the range `224.0.0.0` to `239.255.255.255`, inclusive. The address 224.0.0.0 is reserved and should not be used.

One would join a multicast group by first creating a MulticastSocket with the desired port, then invoking the `joinGroup(InetAddress groupAddr)` method:

```
 // join a Multicast group and send the group salutations
 ...
 String msg = "Hello";
 InetAddress group = InetAddress.getByName("228.5.6.7");
 MulticastSocket s = new MulticastSocket(6789);
 s.joinGroup(group);
 DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),
                             group, 6789);
 s.send(hi);
 // get their responses!
 byte[] buf = new byte[1000];
 DatagramPacket recv = new DatagramPacket(buf, buf.length);
 s.receive(recv);
 ...
 // OK, I'm done talking - leave the group...
 s.leaveGroup(group);
```

When one sends a message to a multicast group, **all** subscribing recipients to that host and port receive the message (within the time-to-live range of the packet, see below). The socket needn't be a member of the multicast group to send messages to it.

When a socket subscribes to a multicast group/port, it receives datagrams sent by other hosts to the group/port, as do all other members of the group and port. A socket relinquishes membership in a group by the leaveGroup(InetAddress addr) method. **Multiple MulticastSocket's** may subscribe to a multicast group and port concurrently, and they will all receive group datagrams.

Currently applets are not allowed to use multicast sockets.

Since:
JDK1.1

Constructor Summary

`MulticastSocket()` Create a multicast socket.
`MulticastSocket(int port)` Create a multicast socket and bind it to a specific port.
`MulticastSocket(SocketAddress bindaddr)` Create a MulticastSocket bound to the specified socket address.

Method Summary

`InetAddress`	`getInterface()` Retrieve the address of the network interface used for multicast packets.
`boolean`	`getLoopbackMode()` Get the setting for local loopback of multicast datagrams.
`NetworkInterface`	`getNetworkInterface()` Get the multicast network interface set.
`int`	`getTimeToLive()` Get the default time-to-live for multicast packets sent out on the socket.
`byte`	`getTTL()` **Deprecated.** *use the getTimeToLive method instead, which returns an **int** instead of a **byte**.*
`void`	`joinGroup(InetAddress mcastaddr)` Joins a multicast group.
`void`	`joinGroup(SocketAddress mcastaddr, NetworkInterface netIf)` Joins the specified multicast group at the specified interface.
`void`	`leaveGroup(InetAddress mcastaddr)` Leave a multicast group.
`void`	`leaveGroup(SocketAddress mcastaddr, NetworkInterface netIf)` Leave a multicast group on a specified local interface.
`void`	`send(DatagramPacket p, byte ttl)` **Deprecated.** *Use the following code or its equivalent instead: ...... int ttl = mcastSocket.getTimeToLive(); mcastSocket.setTimeToLive(newttl); mcastSocket.send(p); mcastSocket.setTimeToLive(ttl); ......*
`void`	`setInterface(InetAddress inf)` Set the multicast network interface used by methods whose behavior would be affected by the value of the network interface.
`void`	`setLoopbackMode(boolean disable)` Disable/Enable local loopback of multicast datagrams The option is used by the platform's networking code as a hint for setting whether multicast data will be looped back to the local socket.
`void`	`setNetworkInterface(NetworkInterface netIf)`

	Specify the network interface for outgoing multicast datagrams sent on this socket.
`void`	**`setTimeToLive`**`(int ttl)` Set the default time-to-live for multicast packets sent out on this `MulticastSocket` in order to control the scope of the multicasts.
`void`	**`setTTL`**`(byte ttl)` **Deprecated.** *use the setTimeToLive method instead, which uses* ***int*** *instead of* ***byte*** *as the type for ttl.*

Methods inherited from class java.net.DatagramSocket
`bind, close, connect, connect, disconnect, getBroadcast, getChannel, getInetAddress, getLocalAddress, getLocalPort, getLocalSocketAddress, getPort, getReceiveBufferSize, getRemoteSocketAddress, getReuseAddress, getSendBufferSize, getSoTimeout, getTrafficClass, isBound, isClosed, isConnected, receive, send, setBroadcast, setDatagramSocketImplFactory, setReceiveBufferSize, setReuseAddress, setSendBufferSize, setSoTimeout, setTrafficClass`

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

Constructor Detail

MulticastSocket

```
public MulticastSocket()
                throws IOException
```

Create a multicast socket.

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.

When the socket is created the `DatagramSocket.setReuseAddress(boolean)` method is called to enable the SO_REUSEADDR socket option.

Throws:

`IOException` - if an I/O exception occurs while creating the MulticastSocket
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.

See Also:

`SecurityManager.checkListen(int)`, `DatagramSocket.setReuseAddress(boolean)`

MulticastSocket

```
public MulticastSocket(int port)
```

```
                throws IOException
```

Create a multicast socket and bind it to a specific port.

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.

When the socket is created the `DatagramSocket.setReuseAddress(boolean)` method is called to enable the SO_REUSEADDR socket option.

Parameters:
`port` - port to use
Throws:
`IOException` - if an I/O exception occurs while creating the MulticastSocket
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.
See Also:
`SecurityManager.checkListen(int)`, `DatagramSocket.setReuseAddress(boolean)`

MulticastSocket

```
public MulticastSocket(SocketAddress bindaddr)
                throws IOException
```

Create a MulticastSocket bound to the specified socket address.

Or, if the address is `null`, create an unbound socket.

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

When the socket is created the `DatagramSocket.setReuseAddress(boolean)` method is called to enable the SO_REUSEADDR socket option.

Parameters:
`bindaddr` - Socket address to bind to, or `null` for an unbound socket.
Throws:
`IOException` - if an I/O exception occurs while creating the MulticastSocket
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.
Since:
1.4
See Also:
`SecurityManager.checkListen(int)`, `DatagramSocket.setReuseAddress(boolean)`

Method Detail

setTTL

```
@Deprecated
public void setTTL(byte ttl)
            throws IOException
```

Deprecated. *use the setTimeToLive method instead, which uses **int** instead of **byte** as the type for ttl.*

Set the default time-to-live for multicast packets sent out on this `MulticastSocket` in order to control the scope of the multicasts.

The ttl is an **unsigned** 8-bit quantity, and so **must** be in the range `0 <= ttl <= 0xFF` .

Parameters:
`ttl` - the time-to-live
Throws:
`IOException` - if an I/O exception occurs while setting the default time-to-live value
See Also:
`getTTL()`

---

setTimeToLive

```
public void setTimeToLive(int ttl)
                   throws IOException
```

Set the default time-to-live for multicast packets sent out on this `MulticastSocket` in order to control the scope of the multicasts.

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

Parameters:
`ttl` - the time-to-live
Throws:
`IOException` - if an I/O exception occurs while setting the default time-to-live value
See Also:
`getTimeToLive()`

---

getTTL

```
@Deprecated
public byte getTTL()
            throws IOException
```

Deprecated. *use the getTimeToLive method instead, which returns an **int** instead of a **byte**.*

Get the default time-to-live for multicast packets sent out on the socket.

Returns:
the default time-to-live value
Throws:
`IOException` - if an I/O exception occurs while getting the default time-to-live value

See Also:
`setTTL(byte)`

---

getTimeToLive

```
public int getTimeToLive()
                  throws IOException
```

Get the default time-to-live for multicast packets sent out on the socket.

Returns:
the default time-to-live value
Throws:
`IOException` - if an I/O exception occurs while getting the default time-to-live value
See Also:
`setTimeToLive(int)`

---

joinGroup

```
public void joinGroup(InetAddress mcastaddr)
               throws IOException
```

Joins a multicast group. Its behavior may be affected by `setInterface` or `setNetworkInterface`.

If there is a security manager, this method first calls its `checkMulticast` method with the `mcastaddr` argument as its argument.

Parameters:
`mcastaddr` - is the multicast address to join
Throws:
`IOException` - if there is an error joining or when the address is not a multicast address.
`SecurityException` - if a security manager exists and its `checkMulticast` method doesn't allow the join.
See Also:
`SecurityManager.checkMulticast(InetAddress)`

---

leaveGroup

```
public void leaveGroup(InetAddress mcastaddr)
                throws IOException
```

Leave a multicast group. Its behavior may be affected by `setInterface` or `setNetworkInterface`.

If there is a security manager, this method first calls its `checkMulticast` method with the `mcastaddr` argument as its argument.

Parameters:
`mcastaddr` - is the multicast address to leave
Throws:
`IOException` - if there is an error leaving or when the address is not a multicast address.
`SecurityException` - if a security manager exists and its `checkMulticast` method doesn't allow the operation.
See Also:
`SecurityManager.checkMulticast(InetAddress)`

---

joinGroup

```
public void joinGroup(SocketAddress mcastaddr,
                      NetworkInterface netIf)
               throws IOException
```

Joins the specified multicast group at the specified interface.

If there is a security manager, this method first calls its `checkMulticast` method with the `mcastaddr` argument as its argument.

Parameters:
`mcastaddr` - is the multicast address to join
`netIf` - specifies the local interface to receive multicast datagram packets, or *null* to defer to the interface set by `setInterface(InetAddress)` or `setNetworkInterface(NetworkInterface)`
Throws:
`IOException` - if there is an error joining or when the address is not a multicast address.
`SecurityException` - if a security manager exists and its `checkMulticast` method doesn't allow the join.
`IllegalArgumentException` - if mcastaddr is null or is a SocketAddress subclass not supported by this socket
Since:
1.4
See Also:
`SecurityManager.checkMulticast(InetAddress)`

---

leaveGroup

```
public void leaveGroup(SocketAddress mcastaddr,
                       NetworkInterface netIf)
                throws IOException
```

Leave a multicast group on a specified local interface.

If there is a security manager, this method first calls its `checkMulticast` method with the `mcastaddr` argument as its argument.

Parameters:
`mcastaddr` - is the multicast address to leave
`netIf` - specifies the local interface or *null* to defer to the interface set by

`setInterface(InetAddress)` or `setNetworkInterface(NetworkInterface)`

Throws:

`IOException` - if there is an error leaving or when the address is not a multicast address.
`SecurityException` - if a security manager exists and its `checkMulticast` method doesn't allow the operation.
`IllegalArgumentException` - if mcastaddr is null or is a SocketAddress subclass not supported by this socket

Since:

1.4

See Also:

`SecurityManager.checkMulticast(InetAddress)`

---

setInterface

```
public void setInterface(InetAddress inf)
                  throws SocketException
```

Set the multicast network interface used by methods whose behavior would be affected by the value of the network interface. Useful for multihomed hosts.

Parameters:

`inf` - the InetAddress

Throws:

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

See Also:

`getInterface()`

---

getInterface

```
public InetAddress getInterface()
                         throws SocketException
```

Retrieve the address of the network interface used for multicast packets.

Returns:

An `InetAddress` representing the address of the network interface used for multicast packets.

Throws:

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

See Also:

`setInterface(java.net.InetAddress)`

---

setNetworkInterface

```
public void setNetworkInterface(NetworkInterface netIf)
                         throws SocketException
```

Specify the network interface for outgoing multicast datagrams sent on this socket.

Parameters:
`netIf` - the interface

Throws:
`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

Since:
1.4

See Also:
`getNetworkInterface()`

---

getNetworkInterface

```
public NetworkInterface getNetworkInterface()
                                     throws SocketException
```

Get the multicast network interface set.

Returns:
the multicast `NetworkInterface` currently set

Throws:
`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

Since:
1.4

See Also:
`setNetworkInterface(NetworkInterface)`

---

setLoopbackMode

```
public void setLoopbackMode(boolean disable)
                     throws SocketException
```

Disable/Enable local loopback of multicast datagrams The option is used by the platform's networking code as a hint for setting whether multicast data will be looped back to the local socket.

Because this option is a hint, applications that want to verify what loopback mode is set to should call `getLoopbackMode()`

Parameters:
`disable` - `true` to disable the LoopbackMode

Throws:
`SocketException` - if an error occurs while setting the value

Since:
1.4

See Also:
`getLoopbackMode()`

---

getLoopbackMode

```
public boolean getLoopbackMode()
```

```
                              throws SocketException
```

Get the setting for local loopback of multicast datagrams.

Returns:
  true if the LoopbackMode has been disabled
Throws:
  `SocketException` - if an error occurs while getting the value
Since:
  1.4
See Also:
  `setLoopbackMode(boolean)`

---

send

```
@Deprecated
public void send(DatagramPacket p,
                                byte ttl)
          throws IOException
```

Deprecated. *Use the following code or its equivalent instead: ...... int ttl = mcastSocket.getTimeToLive(); mcastSocket.setTimeToLive(newttl); mcastSocket.send(p); mcastSocket.setTimeToLive(ttl); ......*

Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket. This method need only be used in instances where a particular TTL is desired; otherwise it is preferable to set a TTL once on the socket, and use that default TTL for all packets. This method does **not** alter the default TTL for the socket. Its behavior may be affected by `setInterface`.

If there is a security manager, 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()` and `ttl` as its arguments. 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` - is the packet to be sent. The packet should contain the destination multicast ip address and the data to be sent. One does not need to be the member of the group to send packets to a destination multicast address.
  `ttl` - optional time to live for multicast packet. default ttl is 1.
Throws:
  `IOException` - is raised if an error occurs i.e error while setting ttl.
  `SecurityException` - if a security manager exists and its `checkMulticast` or `checkConnect` method doesn't allow the send.
See Also:
  `DatagramSocket.send(java.net.DatagramPacket)`, `DatagramSocket.receive(java.net.DatagramPacket)`, `SecurityManager.checkMulticast(java.net.InetAddress, byte)`, `SecurityManager.checkConnect(java.lang.String, int)`
