Package
javax.bluetooth

Class Summary	
Interfaces	
`DiscoveryListener`	The `DiscoveryListener` interface allows an application to receive device discovery and service discovery events.
`L2CAPConnection`	The `L2CAPConnection` interface represents a connection-oriented L2CAP channel.
`L2CAPConnectionNotifier`	The `L2CAPConnectionNotifier` interface provides an L2CAP connection notifier.
`ServiceRecord`	The `ServiceRecord` interface describes characteristics of a Bluetooth service.
Classes	
`DataElement`	The `DataElement` class defines the various data types that a Bluetooth service attribute value may have.
`DeviceClass`	The `DeviceClass` class represents the class of device (CoD) record as defined by the Bluetooth specification.
`DiscoveryAgent`	The `DiscoveryAgent` class provides methods to perform device and service discovery.
`LocalDevice`	The `LocalDevice` class defines the basic functions of the Bluetooth manager.
`RemoteDevice`	The `RemoteDevice` class represents a remote Bluetooth device.
`UUID`	The `UUID` class defines universally unique identifiers.
Exceptions	
`BluetoothConnectionException`	This `BluetoothConnectionException` is thrown when a Bluetooth connection (L2CAP, RFCOMM, or OBEX over RFCOMM) cannot be established successfully.
`BluetoothStateException`	The `BluetoothStateException` is thrown when a request is made to the Bluetooth system that the system cannot support in its present state.
`ServiceRegistrationException`	The `ServiceRegistrationException` is thrown when there is a failure to add a service record to the local Service Discovery Database (SDDB) or to modify an existing service record in the SDDB.

javax.bluetooth
BluetoothConnectionException

Declaration

```
public class BluetoothConnectionException extends java.io.IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--javax.bluetooth.BluetoothConnectionException
```

Description

This `BluetoothConnectionException` is thrown when a Bluetooth connection (L2CAP, RFCOMM, or OBEX over RFCOMM) cannot be established successfully. The fields in this exception class indicate the cause of the exception. For example, an L2CAP connection may fail due to a security problem. This reason is passed on to the application through this class.

Member Summary

Fields

`public static final` **FAILED_NOINFO**
Indicates the connection to the server failed due to unknown reasons.

`public static final` **NO_RESOURCES**
Indicates the connection failed due to a lack of resources either on the local device or on the remote device.

`public static final` **SECURITY_BLOCK**
Indicates the connection failed because the security settings on the local device or the remote device were incompatible with the request.

`public static final` **TIMEOUT**
Indicates the connection to the server failed due to a timeout.

`public static final` **UNACCEPTABLE_PARAMS**
Indicates the connection failed because the configuration parameters provided were not acceptable to either the remote device or the local device.

`public static final` **UNKNOWN_PSM**
Indicates the connection to the server failed because no service for the given PSM was registered.

Constructors

`public` **BluetoothConnectionException(int)**
Creates a new `BluetoothConnectionException` with the error indicator specified.

`public` **BluetoothConnectionException(int, String)**
Creates a new `BluetoothConnectionException` with the error indicator and message specified.

Methods

Member Summary	
public int	getStatus() Gets the status set in the constructor that will indicate the reason for the exception.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
Methods inherited from class java.lang.Throwable
getMessage, printStackTrace, toString

Fields

FAILED_NOINFO

public static final int **FAILED_NOINFO**

Indicates the connection to the server failed due to unknown reasons.

The value for FAILED_NOINFO is 0x0004 (4).

NO_RESOURCES

public static final int **NO_RESOURCES**

Indicates the connection failed due to a lack of resources either on the local device or on the remote device.

The value for NO_RESOURCES is 0x0003 (3).

SECURITY_BLOCK

public static final int **SECURITY_BLOCK**

Indicates the connection failed because the security settings on the local device or the remote device were incompatible with the request.

The value for SECURITY_BLOCK is 0x0002 (2).

TIMEOUT

public static final int **TIMEOUT**

Indicates the connection to the server failed due to a timeout.

The value for TIMEOUT is 0x0005 (5).

UNACCEPTABLE_PARAMS

public static final int **UNACCEPTABLE_PARAMS**

Indicates the connection failed because the configuration parameters provided were not acceptable to either the remote device or the local device.

The value for `UNACCEPTABLE_PARAMS` is 0x0006 (6).

UNKNOWN_PSM

```
public static final int UNKNOWN_PSM
```

Indicates the connection to the server failed because no service for the given PSM was registered.

The value for `UNKNOWN_PSM` is 0x0001 (1).

Constructors

BluetoothConnectionException(int)

```
public BluetoothConnectionException(int error)
```

Creates a new `BluetoothConnectionException` with the error indicator specified.

Parameters:

`error` - indicates the exception condition; must be one of the constants described in this class

Throws:

`IllegalArgumentException` - if the input value is not one of the constants in this class

BluetoothConnectionException(int, String)

```
public BluetoothConnectionException(int error, java.lang.String msg)
```

Creates a new `BluetoothConnectionException` with the error indicator and message specified.

Parameters:

`error` - indicates the exception condition; must be one of the constants described in this class

`msg` - a description of the exception; may by `null`

Throws:

`IllegalArgumentException` - if the input value is not one of the constants in this class

Methods

getStatus()

```
public int getStatus()
```

Gets the status set in the constructor that will indicate the reason for the exception.

Returns: cause for the exception; will be one of the constants defined in this class

javax.bluetooth
BluetoothStateException

Declaration
```
public class BluetoothStateException extends java.io.IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--javax.bluetooth.BluetoothStateException
```

Description
The `BluetoothStateException` is thrown when a request is made to the Bluetooth system that the system cannot support in its present state. If, however, the Bluetooth system was not in this state, it could support this operation. For example, some Bluetooth systems do not allow the device to go into inquiry mode if a connection is established. This exception would be thrown if `startInquiry()` were called.

Member Summary	
Constructors	
public	`BluetoothStateException()` Creates a new `BluetoothStateException` without a detail message.
public	`BluetoothStateException(String)` Creates a `BluetoothStateException` with the specified detail message.

Inherited Member Summary
Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`
Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Constructors

BluetoothStateException()
```
public BluetoothStateException()
```
Creates a new `BluetoothStateException` without a detail message.

BluetoothStateException(String)

```
public BluetoothStateException(java.lang.String msg)
```

Creates a `BluetoothStateException` with the specified detail message.

Parameters:

`msg` - the reason for the exception

javax.bluetooth
DataElement

Declaration

```
public class DataElement

java.lang.Object
  |
  +--javax.bluetooth.DataElement
```

Description

The `DataElement` class defines the various data types that a Bluetooth service attribute value may have. The following table describes the data types and valid values that a `DataElement` object can store.

Data Type	Valid Values
`NULL`	represents a `null` value
`U_INT_1`	`long` value range [0, 255]
`U_INT_2`	`long` value range [0, 2^{16}-1]
`U_INT_4`	`long` value range [0, 2^{32}-1]
`U_INT_8`	`byte[]` value range [0, 2^{64}-1]
`U_INT_16`	`byte[]` value range [0, 2^{128}-1]
`INT_1`	`long` value range [-128, 127]
`INT_2`	`long` value range [-2^{15}, 2^{15}-1]
`INT_4`	`long` value range [-2^{31}, 2^{31}-1]
`INT_8`	`long` value range [-2^{63}, 2^{63}-1]
`INT_16`	`byte[]` value range [-2^{127}, 2^{127}-1]
`URL`	`java.lang.String`
`UUID`	`javax.bluetooth.UUID`
`BOOL`	`boolean`
`STRING`	`java.lang.String`
`DATSEQ`	`java.util.Enumeration`
`DATALT`	`java.util.Enumeration`

Member Summary

Fields

`public static final BOOL`

Defines data of type BOOL.

Member Summary

public static final	DATALT Defines data of type DATALT.
public static final	DATSEQ Defines data of type DATSEQ.
public static final	INT_1 Defines a signed integer of size one byte.
public static final	INT_16 Defines a signed integer of size sixteen bytes.
public static final	INT_2 Defines a signed integer of size two bytes.
public static final	INT_4 Defines a signed integer of size four bytes.
public static final	INT_8 Defines a signed integer of size eight bytes.
public static final	NULL Defines data of type NULL.
public static final	STRING Defines data of type STRING.
public static final	U_INT_1 Defines an unsigned integer of size one byte.
public static final	U_INT_16 Defines an unsigned integer of size sixteen bytes.
public static final	U_INT_2 Defines an unsigned integer of size two bytes.
public static final	U_INT_4 Defines an unsigned integer of size four bytes.
public static final	U_INT_8 Defines an unsigned integer of size eight bytes.
public static final	URL Defines data of type URL.
public static final	UUID Defines data of type UUID.

Constructors

public	DataElement(boolean) Creates a DataElement whose data type is BOOL and whose value is equal to bool
public	DataElement(int) Creates a DataElement of type NULL, DATALT, or DATSEQ.
public	DataElement(int, long) Creates a DataElement that encapsulates an integer value of size U_INT_1, U_INT_2, U_INT_4, INT_1, INT_2, INT_4, and INT_8.
public	DataElement(int, Object) Creates a DataElement whose data type is given by valueType and whose value is specified by the argument value.

Methods

public void	addElement(DataElement) Adds a DataElement to this DATALT or DATSEQ DataElement object.
public boolean	getBoolean() Returns the value of the DataElement if it is represented as a boolean.
public int	getDataType() Returns the data type of the object this DataElement represents.
public long	getLong() Returns the value of the DataElement if it can be represented as a long.

Member Summary	
public int	getSize() Returns the number of DataElements that are present in this DATALT or DATSEQ object.
public Object	getValue() Returns the value of this DataElement as an Object.
public void	insertElementAt(DataElement, int) Inserts a DataElement at the specified location.
public boolean	removeElement(DataElement) Removes the first occurrence of the DataElement from this object.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

BOOL

```
public static final int BOOL
```

Defines data of type BOOL.

The value of the constant BOOL is 0x28 (40).

DATALT

```
public static final int DATALT
```

Defines data of type DATALT. The service attribute value whose data has this type must consider only one of the elements of the set, i.e., the value is the not the whole set but only one element of the set. The user is free to choose any one element. The elements of the set can be of any type defined in this class, including DATALT.

The value of the constant DATALT is 0x38 (56).

DATSEQ

```
public static final int DATSEQ
```

Defines data of type DATSEQ. The service attribute value whose data has this type must consider all the elements of the list, i.e. the value is the whole set and not a subset. The elements of the set can be of any type defined in this class, including DATSEQ.

The value of the constant DATSEQ is 0x30 (48).

INT_1

```
public static final int INT_1
```

Defines a signed integer of size one byte.

The value of the constant `INT_1` is 0x10 (16).

INT_16

```
public static final int INT_16
```

Defines a signed integer of size sixteen bytes.

The value of the constant `INT_16` is 0x14 (20).

INT_2

```
public static final int INT_2
```

Defines a signed integer of size two bytes.

The value of the constant `INT_2` is 0x11 (17).

INT_4

```
public static final int INT_4
```

Defines a signed integer of size four bytes.

The value of the constant `INT_4` is 0x12 (18).

INT_8

```
public static final int INT_8
```

Defines a signed integer of size eight bytes.

The value of the constant `INT_8` is 0x13 (19).

NULL

```
public static final int NULL
```

Defines data of type NULL. The value for data type `DataElement.NULL` is implicit, i.e., there is no representation of it. Accordingly there is no method to retrieve it, and attempts to retrieve the value will throw an exception.

The value of `NULL` is 0x00 (0).

STRING

```
public static final int STRING
```

Defines data of type STRING.

The value of the constant `STRING` is 0x20 (32).

U_INT_1

```
public static final int U_INT_1
```

Defines an unsigned integer of size one byte.

The value of the constant `U_INT_1` is 0x08 (8).

U_INT_16

```
public static final int U_INT_16
```

Defines an unsigned integer of size sixteen bytes.

The value of the constant `U_INT_16` is 0x0C (12).

U_INT_2

```
public static final int U_INT_2
```

Defines an unsigned integer of size two bytes.

The value of the constant `U_INT_2` is 0x09 (9).

U_INT_4

```
public static final int U_INT_4
```

Defines an unsigned integer of size four bytes.

The value of the constant `U_INT_4` is 0x0A (10).

U_INT_8

```
public static final int U_INT_8
```

Defines an unsigned integer of size eight bytes.

The value of the constant `U_INT_8` is 0x0B (11).

URL

```
public static final int URL
```

Defines data of type URL.

The value of the constant `URL` is 0x40 (64).

UUID

```
public static final int UUID
```

Defines data of type UUID.

The value of the constant `UUID` is 0x18 (24).

Constructors

DataElement(boolean)

```
public DataElement(boolean bool)
```

Creates a `DataElement` whose data type is `BOOL` and whose value is equal to `bool`

Parameters:

`bool` - the value of the `DataElement` of type BOOL.

See Also: `BOOL`

DataElement(int)

```
public DataElement(int valueType)
```

Creates a `DataElement` of type `NULL`, `DATALT`, or `DATSEQ`.

Parameters:

`valueType` - the type of DataElement to create: `NULL`, `DATALT`, or `DATSEQ`

Throws:

`IllegalArgumentException` - if `valueType` is not `NULL`, `DATALT`, or `DATSEQ`

See Also: `NULL`, `DATALT`, `DATSEQ`

DataElement(int, long)

```
public DataElement(int valueType, long value)
```

Creates a `DataElement` that encapsulates an integer value of size `U_INT_1`, `U_INT_2`, `U_INT_4`, `INT_1`, `INT_2`, `INT_4`, and `INT_8`. The legal values for the `valueType` and the corresponding attribute values are:

Value Type	Value Range
`U_INT_1`	[0, 2^8-1]
`U_INT_2`	[0, 2^{16}-1]
`U_INT_4`	[0, 2^{32}-1]
`INT_1`	[-2^7, 2^7-1]
`INT_2`	[-2^{15}, 2^{15}-1]
`INT_4`	[-2^{31}, 2^{31}-1]
`INT_8`	[-2^{63}, 2^{63}-1]

All other pairings are illegal and will cause an `IllegalArgumentException` to be thrown.

Parameters:

`valueType` - the data type of the object that is being created; must be one of the following: `U_INT_1`, `U_INT_2`, `U_INT_4`, `INT_1`, `INT_2`, `INT_4`, or `INT_8`

`value` - the value of the object being created; must be in the range specified for the given `valueType`

Throws:

`IllegalArgumentException` - if the `valueType` is not valid or the `value` for the given legal `valueType` is outside the valid range

See Also: `U_INT_1`, `U_INT_2`, `U_INT_4`, `INT_1`, `INT_2`, `INT_4`, `INT_8`

DataElement(int, Object)

```
public DataElement(int valueType, java.lang.Object value)
```

Creates a `DataElement` whose data type is given by `valueType` and whose value is specified by the argument `value`. The legal values for the `valueType` and the corresponding attribute values are:

Value Type	Java Type / Value Range
URL	java.lang.String
UUID	javax.bluetooth.UUID
STRING	java.lang.String
INT_16	[-2^{127}, 2^{127}-1] as a byte array whose length must be 16
U_INT_8	[0, 2^{64}-1] as a byte array whose length must be 8
U_INT_16	[0, 2^{128}-1] as a byte array whose length must be 16

All other pairings are illegal and would cause an IllegalArgumentException exception.

Parameters:

valueType - the data type of the object that is being created; must be one of the following: URL, UUID, STRING, INT_16, U_INT_8, or U_INT_16

value - the value for the DataElement being created of type valueType

Throws:

IllegalArgumentException - if the value is not of the valueType type or is not in the range specified or is null

See Also: URL, UUID, STRING, U_INT_8, INT_16, U_INT_16

Methods

addElement(DataElement)

```
public void addElement(DataElement elem)
```

Adds a DataElement to this DATALT or DATSEQ DataElement object. The elem will be added at the end of the list. The elem can be of any DataElement type, i.e., URL, NULL, BOOL, UUID, STRING, DATSEQ, DATALT, and the various signed and unsigned integer types. The same object may be added twice. If the object is successfully added the size of the DataElement is increased by one.

Parameters:

elem - the DataElement object to add

Throws:

ClassCastException - if the method is invoked on a DataElement whose type is not DATALT or DATSEQ

NullPointerException - if elem is null

getBoolean()

```
public boolean getBoolean()
```

Returns the value of the DataElement if it is represented as a boolean.

Returns: the boolean value of this DataElement object

Throws:
`ClassCastException` - if the data type of this object is not of type `BOOL`

getDataType()

`public int getDataType()`

Returns the data type of the object this `DataElement` represents.

Returns: the data type of this `DataElement object; the legal return values are: URL, NULL, BOOL, UUID, STRING, DATSEQ, DATALT, U_INT_1, U_INT_2, U_INT_4, U_INT_8, U_INT_16, INT_1, INT_2, INT_4, INT_8, or INT_16`

getLong()

`public long getLong()`

Returns the value of the `DataElement` if it can be represented as a `long`. The data type of the object must be `U_INT_1`, `U_INT_2`, `U_INT_4`, `INT_1`, `INT_2`, `INT_4`, or `INT_8`.

Returns: the value of the `DataElement` as a `long`

Throws:
`ClassCastException` - if the data type of the object is not `U_INT_1`, `U_INT_2`, `U_INT_4`, `INT_1`, `INT_2`, `INT_4`, or `INT_8`

getSize()

`public int getSize()`

Returns the number of `DataElements` that are present in this `DATALT` or `DATSEQ` object. It is possible that the number of elements is equal to zero.

Returns: the number of elements in this `DATALT` or `DATSEQ`

Throws:
`ClassCastException` - if this object is not of type `DATALT` or `DATSEQ`

getValue()

`public java.lang.Object getValue()`

Returns the value of this `DataElement` as an `Object`. This method returns the appropriate Java object for the following data types: `URL`, `UUID`, `STRING`, `DATSEQ`, `DATALT`, `U_INT_8`, `U_INT_16`, and `INT_16`. Modifying the returned `Object` will not change this `DataElement`. The following are the legal pairs of data type and Java object type being returned.

`DataElement` Data Type	Java Data Type
`URL`	`java.lang.String`
`UUID`	`javax.bluetooth.UUID`
`STRING`	`java.lang.String`
`DATSEQ`	`java.util.Enumeration`
`DATALT`	`java.util.Enumeration`

U_INT_8	byte[] of length 8
U_INT_16	byte[] of length 16
INT_16	byte[] of length 16

Returns: the value of this object

Throws:

ClassCastException - if the object is not a URL, UUID, STRING, DATSEQ, DATALT, U_INT_8, U_INT_16, or INT_16

insertElementAt(DataElement, int)

```
public void insertElementAt(DataElement elem, int index)
```

Inserts a DataElement at the specified location. This method can be invoked only on a DATALT or DATSEQ DataElement. elem can be of any DataElement type, i.e., URL, NULL, BOOL, UUID, STRING, DATSEQ, DATALT, and the various signed and unsigned integers. The same object may be added twice. If the object is successfully added the size will be increased by one. Each element with an index greater than or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

The index must be greater than or equal to 0 and less than or equal to the current size. Therefore, DATALT and DATSEQ are zero-based objects.

Parameters:

elem - the DataElement object to add

index - the location at which to add the DataElement

Throws:

ClassCastException - if the method is invoked on an instance of DataElement whose type is not DATALT or DATSEQ

IndexOutOfBoundsException - if index is negative or greater than the size of the DATALT or DATSEQ

NullPointerException - if elem is null

removeElement(DataElement)

```
public boolean removeElement(DataElement elem)
```

Removes the first occurrence of the DataElement from this object. elem may be of any type, i.e., URL, NULL, BOOL, UUID, STRING, DATSEQ, DATALT, or the variously sized signed and unsigned integers. Only the first object in the list that is equal to elem will be removed. Other objects, if present, are not removed. Since this class doesnt override the equals() method of the Object class, the remove method compares only the references of objects. If elem is successfully removed the size of this DataElement is decreased by one. Each DataElement in the DATALT or DATSEQ with an index greater than the index of elem is shifted downward to have an index one smaller than the value it had previously.

Parameters:

elem - the DataElement to be removed

Returns: true if the input value was found and removed; else false

Throws:

ClassCastException - if this object is not of type DATALT or DATSEQ

`NullPointerException` - if `elem` is `null`

javax.bluetooth
DeviceClass

Declaration

```
public class DeviceClass

java.lang.Object
  |
  +--javax.bluetooth.DeviceClass
```

Description

The `DeviceClass` class represents the class of device (CoD) record as defined by the Bluetooth specification. This record is defined in the Bluetooth Assigned Numbers document and contains information on the type of the device and the type of services available on the device.

The Bluetooth Assigned Numbers document (http://www.bluetooth.org/assigned-numbers/baseband.htm) defines the service class, major device class, and minor device class. The table below provides some examples of possible return values and their meaning:

Method	Return Value	Class of Device
`getServiceClasses()`	0x22000	Networking and Limited Discoverable Major Service Classes
`getServiceClasses()`	0x100000	Object Transfer Major Service Class
`getMajorDeviceClass()`	0x00	Miscellaneous Major Device Class
`getMajorDeviceClass()`	0x200	Phone Major Device Class
`getMinorDeviceClass()`	0x0C	With a Computer Major Device Class, Laptop Minor Device Class
`getMinorDeviceClass()`	0x04	With a Phone Major Device Class, Cellular Minor Device Class

Member Summary

Constructors

`public` `DeviceClass(int)`
Creates a `DeviceClass` from the class of device record provided.

Methods

`public int` `getMajorDeviceClass()`
Retrieves the major device class.

`public int` `getMinorDeviceClass()`
Retrieves the minor device class.

`public int` `getServiceClasses()`
Retrieves the major service classes.

Inherited Member Summary
Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

DeviceClass(int)

```
public DeviceClass(int record)
```

Creates a `DeviceClass` from the class of device record provided. `record` must follow the format of the class of device record in the Bluetooth specification.

Parameters:

`record` - describes the classes of a device

Throws:

`IllegalArgumentException` - if `record` has any bits between 24 and 31 set

Methods

getMajorDeviceClass()

```
public int getMajorDeviceClass()
```

Retrieves the major device class. A device may have only a single major device class.

Returns: the major device class

getMinorDeviceClass()

```
public int getMinorDeviceClass()
```

Retrieves the minor device class.

Returns: the minor device class

getServiceClasses()

```
public int getServiceClasses()
```

Retrieves the major service classes. A device may have multiple major service classes. When this occurs, the major service classes are bitwise OR’ed together.

Returns: the major service classes

javax.bluetooth
DiscoveryAgent

Declaration
```
public class DiscoveryAgent

java.lang.Object
  |
  +--javax.bluetooth.DiscoveryAgent
```

Description
The `DiscoveryAgent` class provides methods to perform device and service discovery. A local device must have only one `DiscoveryAgent` object. This object must be retrieved by a call to `getDiscoveryAgent()` on the `LocalDevice` object.

Device Discovery
There are two ways to discover devices. First, an application may use `startInquiry()` to start an inquiry to find devices in proximity to the local device. Discovered devices are returned via the `deviceDiscovered()` method of the interface `DiscoveryListener`. The second way to discover devices is via the `retrieveDevices()` method. This method will return devices that have been discovered via a previous inquiry or devices that are classified as pre-known. (Pre-known devices are those devices that are defined in the Bluetooth Control Center as devices this device frequently contacts.) The `retrieveDevices()` method does not perform an inquiry, but provides a quick way to get a list of devices that may be in the area.

Service Discovery
The `DiscoveryAgent` class also encapsulates the functionality provided by the service discovery application profile. The class provides an interface for an application to search and retrieve attributes for a particular service. There are two ways to search for services. To search for a service on a single device, the `searchServices()` method should be used. On the other hand, if you don't care which device a service is on, the `selectService()` method does a service search on a set of remote devices.

Member Summary

Fields

`public static final` `CACHED`	Used with the `retrieveDevices()` method to return those devices that were found via a previous inquiry.
`public static final` `GIAC`	The inquiry access code for General/Unlimited Inquiry Access Code (GIAC).
`public static final` `LIAC`	The inquiry access code for Limited Dedicated Inquiry Access Code (LIAC).
`public static final` `NOT_DISCOVERABLE`	Takes the device out of discoverable mode.
`public static final` `PREKNOWN`	Used with the `retrieveDevices()` method to return those devices that are defined to be pre-known devices.

Methods

Member Summary	
public boolean	cancelInquiry(DiscoveryListener) Removes the device from inquiry mode.
public boolean	cancelServiceSearch(int) Cancels the service search transaction that has the specified transaction ID.
public RemoteDevice	retrieveDevices(int) Returns an array of Bluetooth devices that have either been found by the local device during previous inquiry requests or been specified as a pre-known device depending on the argument.
public int	searchServices(int[], UUID[], RemoteDevice, DiscoveryListener) Searches for services on a remote Bluetooth device that have all the UUIDs specified in uuidSet.
public String	selectService(UUID, int, boolean) Attempts to locate a service that contains uuid in the ServiceClassIDList of its service record.
public boolean	startInquiry(int, DiscoveryListener) Places the device into inquiry mode.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

CACHED

```
public static final int CACHED
```

Used with the retrieveDevices() method to return those devices that were found via a previous inquiry. If no inquiries have been started, this will cause the method to return null.

The value of CACHED is 0x00 (0).

See Also: retrieveDevices(int)

GIAC

```
public static final int GIAC
```

The inquiry access code for General/Unlimited Inquiry Access Code (GIAC). This is used to specify the type of inquiry to complete or respond to.

The value of GIAC is 0x9E8B33 (10390323). This value is defined in the Bluetooth Assigned Numbers document.

LIAC

```
public static final int LIAC
```

The inquiry access code for Limited Dedicated Inquiry Access Code (LIAC). This is used to specify the type of inquiry to complete or respond to.

The value of `LIAC` is 0x9E8B00 (10390272). This value is defined in the Bluetooth Assigned Numbers document.

NOT_DISCOVERABLE

```
public static final int NOT_DISCOVERABLE
```

Takes the device out of discoverable mode.

The value of `NOT_DISCOVERABLE` is 0x00 (0).

PREKNOWN

```
public static final int PREKNOWN
```

Used with the `retrieveDevices()` method to return those devices that are defined to be pre-known devices. Pre-known devices are specified in the BCC. These are devices that are specified by the user as devices with which the local device will frequently communicate.

The value of `PREKNOWN` is 0x01 (1).

See Also: `retrieveDevices(int)`

Methods

cancelInquiry(DiscoveryListener)

```
public boolean cancelInquiry(DiscoveryListener listener)
```

Removes the device from inquiry mode.

An `inquiryCompleted()` event will occur with a type of `INQUIRY_TERMINATED` as a result of calling this method. After receiving this event, no further `deviceDiscovered()` events will occur as a result of this inquiry.

This method will only cancel the inquiry if the `listener` provided is the listener that started the inquiry.

Parameters:
`listener` - the listener that is receiving inquiry events

Returns: `true` if the inquiry was canceled; otherwise `false` if the inquiry was not canceled or if the inquiry was not started using `listener`

Throws:
`NullPointerException` - if `listener` is `null`

cancelServiceSearch(int)

```
public boolean cancelServiceSearch(int transID)
```

Cancels the service search transaction that has the specified transaction ID. The ID was assigned to the transaction by the method `searchServices()`. A `serviceSearchCompleted()` event with a discovery type of `SERVICE_SEARCH_TERMINATED` will occur when this method is called. After receiving this event, no further `servicesDiscovered()` events will occur as a result of this search.

Parameters:

`transID` - the ID of the service search transaction to cancel; returned by `searchServices()`

Returns: `true` if the service search transaction is terminated, else `false` if the `transID` does not represent an active service search transaction

retrieveDevices(int)

```
public RemoteDevice[] retrieveDevices(int option)
```

Returns an array of Bluetooth devices that have either been found by the local device during previous inquiry requests or been specified as a pre-known device depending on the argument. The list of previously found devices is maintained by the implementation of this API. (In other words, maintenance of the list of previously found devices is an implementation detail.) A device can be set as a pre-known device in the Bluetooth Control Center.

Parameters:

`option` - `CACHED` if previously found devices should be returned; `PREKNOWN` if pre-known devices should be returned

Returns: an array containing the Bluetooth devices that were previously found if `option` is `CACHED`; an array of devices that are pre-known devices if `option` is `PREKNOWN`; `null` if no devices meet the criteria

Throws:

`IllegalArgumentException` - if `option` is not `CACHED` or `PREKNOWN`

searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)

```
public int searchServices(int[] attrSet, UUID[] uuidSet, RemoteDevice btDev,
            DiscoveryListener discListener)
            throws BluetoothStateException
```

Searches for services on a remote Bluetooth device that have all the UUIDs specified in `uuidSet`. Once the service is found, the attributes specified in `attrSet` and the default attributes are retrieved. The default attributes are ServiceRecordHandle (0x0000), ServiceClassIDList (0x0001), ServiceRecordState (0x0002), ServiceID (0x0003), and ProtocolDescriptorList (0x0004).If `attrSet` is `null` then only the default attributes will be retrieved. `attrSet` does not have to be sorted in increasing order, but must only contain values in the range [0 - (2^{16}-1)].

Parameters:

`attrSet` - indicates the attributes whose values will be retrieved on services which have the UUIDs specified in `uuidSet`

`uuidSet` - the set of UUIDs that are being searched for; all services returned will contain all the UUIDs specified here

`btDev` - the remote Bluetooth device to search for services on

`discListener` - the object that will receive events when services are discovered

Returns: the transaction ID of the service search; this number must be positive

Throws:

`BluetoothStateException` - if the number of concurrent service search transactions exceeds the limit specified by the `bluetooth.sd.trans.max` property obtained from the class `LocalDevice` or the system is unable to start one due to current conditions

`IllegalArgumentException` - if `attrSet` has an illegal service attribute ID or exceeds the property `bluetooth.sd.attr.retrievable.max` defined in the class `LocalDevice`; if `attrSet` or `uuidSet` is of length 0; if `attrSet` or `uuidSet` contains duplicates

`NullPointerException` - if `uuidSet`, `btDev`, or `discListener` is `null`; if an element in `uuidSet` array is `null`

See Also: `DiscoveryListener`

selectService(UUID, int, boolean)

```
public java.lang.String selectService(UUID uuid, int security, boolean master)
            throws BluetoothStateException
```

Attempts to locate a service that contains `uuid` in the ServiceClassIDList of its service record. This method will return a string that may be used in `Connector.open()` to establish a connection to the service. How the service is selected if there are multiple services with `uuid` and which devices to search is implementation dependent.

Parameters:

`uuid` - the UUID to search for in the ServiceClassIDList

`security` - specifies the security requirements for a connection to this service; must be one of `ServiceRecord.NOAUTHENTICATE_NOENCRYPT`, `ServiceRecord.AUTHENTICATE_NOENCRYPT`, or `ServiceRecord.AUTHENTICATE_ENCRYPT`

`master` - determines if this client must be the master of the connection; `true` if the client must be the master; `false` if the client can be the master or the slave

Returns: the connection string used to connect to the service with a UUID of `uuid`; or `null` if no service could be found with a UUID of `uuid` in the ServiceClassIDList

Throws:

`BluetoothStateException` - if the Bluetooth system cannot start the request due to the current state of the Bluetooth system

`NullPointerException` - if `uuid` is `null`

`IllegalArgumentException` - if `security` is not `ServiceRecord.NOAUTHENTICATE_NOENCRYPT`, `ServiceRecord.AUTHENTICATE_NOENCRYPT`, or `ServiceRecord.AUTHENTICATE_ENCRYPT`

See Also: `NOAUTHENTICATE_NOENCRYPT`, `AUTHENTICATE_NOENCRYPT`, `AUTHENTICATE_ENCRYPT`

startInquiry(int, DiscoveryListener)

```
public boolean startInquiry(int accessCode, DiscoveryListener listener)
            throws BluetoothStateException
```

Places the device into inquiry mode. The length of the inquiry is implementation dependent. This method will search for devices with the specified inquiry access code. Devices that responded to the inquiry are returned to the application via the method `deviceDiscovered()` of the interface `DiscoveryListener`. The `cancelInquiry()` method is called to stop the inquiry.

Parameters:

`accessCode` - the type of inquiry to complete

`listener` - the event listener that will receive device discovery events

Returns: `true` if the inquiry was started; `false` if the inquiry was not started because the `accessCode` is not supported

Throws:

`IllegalArgumentException` - if the access code provided is not `LIAC`, `GIAC`, or in the range 0x9E8B00 to 0x9E8B3F

`NullPointerException` - if `listener` is `null`

`BluetoothStateException` - if the Bluetooth device does not allow an inquiry to be started due to other operations that are being performed by the device

See Also: `cancelInquiry(DiscoveryListener)`, `GIAC`, `LIAC`

javax.bluetooth
DiscoveryListener

Declaration
`public interface` **`DiscoveryListener`**

Description
The `DiscoveryListener` interface allows an application to receive device discovery and service discovery events. This interface provides four methods, two for discovering devices and two for discovering services.

Member Summary

Fields

`public static final` `INQUIRY_COMPLETED`
Indicates the normal completion of device discovery.

`public static final` `INQUIRY_ERROR`
Indicates that the inquiry request failed to complete normally, but was not cancelled.

`public static final` `INQUIRY_TERMINATED`
Indicates device discovery has been canceled by the application and did not complete.

`public static final` `SERVICE_SEARCH_COMPLETED`
Indicates the normal completion of service discovery.

`public static final` `SERVICE_SEARCH_DEVICE_NOT_REACHABLE`
Indicates the service search could not be completed because the remote device provided to `DiscoveryAgent.searchServices()` could not be reached.

`public static final` `SERVICE_SEARCH_ERROR`
Indicates the service search terminated with an error.

`public static final` `SERVICE_SEARCH_NO_RECORDS`
Indicates the service search has completed with no service records found on the device.

`public static final` `SERVICE_SEARCH_TERMINATED`
Indicates the service search has been canceled by the application and did not complete.

Methods

`public void` `deviceDiscovered(RemoteDevice, DeviceClass)`
Called when a device is found during an inquiry.

`public void` `inquiryCompleted(int)`
Called when an inquiry is completed.

`public void` `servicesDiscovered(int, ServiceRecord[])`
Called when service(s) are found during a service search.

`public void` `serviceSearchCompleted(int, int)`
Called when a service search is completed or was terminated because of an error.

Fields

INQUIRY_COMPLETED
`public static final int` **`INQUIRY_COMPLETED`**

Indicates the normal completion of device discovery. Used with the `inquiryCompleted()` method.

The value of `INQUIRY_COMPLETED` is 0x00 (0).

See Also: `inquiryCompleted(int)`, `startInquiry(int, DiscoveryListener)`

INQUIRY_ERROR

`public static final int INQUIRY_ERROR`

Indicates that the inquiry request failed to complete normally, but was not cancelled.

The value of `INQUIRY_ERROR` is 0x07 (7).

See Also: `inquiryCompleted(int)`, `startInquiry(int, DiscoveryListener)`

INQUIRY_TERMINATED

`public static final int INQUIRY_TERMINATED`

Indicates device discovery has been canceled by the application and did not complete. Used with the `inquiryCompleted()` method.

The value of `INQUIRY_TERMINATED` is 0x05 (5).

See Also: `inquiryCompleted(int)`, `startInquiry(int, DiscoveryListener)`, `cancelInquiry(DiscoveryListener)`

SERVICE_SEARCH_COMPLETED

`public static final int SERVICE_SEARCH_COMPLETED`

Indicates the normal completion of service discovery. Used with the `serviceSearchCompleted()` method.

The value of `SERVICE_SEARCH_COMPLETED` is 0x01 (1).

See Also: `serviceSearchCompleted(int, int)`, `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`

SERVICE_SEARCH_DEVICE_NOT_REACHABLE

`public static final int SERVICE_SEARCH_DEVICE_NOT_REACHABLE`

Indicates the service search could not be completed because the remote device provided to `DiscoveryAgent.searchServices()` could not be reached. Used with the `serviceSearchCompleted()` method.

The value of `SERVICE_SEARCH_DEVICE_NOT_REACHABLE` is 0x06 (6).

See Also: `serviceSearchCompleted(int, int)`, `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`

SERVICE_SEARCH_ERROR

`public static final int SERVICE_SEARCH_ERROR`

Indicates the service search terminated with an error. Used with the `serviceSearchCompleted()` method.

The value of `SERVICE_SEARCH_ERROR` is 0x03 (3).

See Also: `serviceSearchCompleted(int, int)`, `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`

SERVICE_SEARCH_NO_RECORDS

```
public static final int SERVICE_SEARCH_NO_RECORDS
```

Indicates the service search has completed with no service records found on the device. Used with the `serviceSearchCompleted()` method.

The value of `SERVICE_SEARCH_NO_RECORDS` is 0x04 (4).

See Also: `serviceSearchCompleted(int, int)`, `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`

SERVICE_SEARCH_TERMINATED

```
public static final int SERVICE_SEARCH_TERMINATED
```

Indicates the service search has been canceled by the application and did not complete. Used with the `serviceSearchCompleted()` method.

The value of `SERVICE_SEARCH_TERMINATED` is 0x02 (2).

See Also: `serviceSearchCompleted(int, int)`, `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`, `cancelServiceSearch(int)`

Methods

deviceDiscovered(RemoteDevice, DeviceClass)

```
public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
```

Called when a device is found during an inquiry. An inquiry searches for devices that are discoverable. The same device may be returned multiple times.

Parameters:

`btDevice` - the device that was found during the inquiry

`cod` - the service classes, major device class, and minor device class of the remote device

See Also: `startInquiry(int, DiscoveryListener)`

inquiryCompleted(int)

```
public void inquiryCompleted(int discType)
```

Called when an inquiry is completed. The `discType` will be `INQUIRY_COMPLETED` if the inquiry ended normally or `INQUIRY_TERMINATED` if the inquiry was canceled by a call to `DiscoveryAgent.cancelInquiry()`. The `discType` will be `INQUIRY_ERROR` if an error occurred while processing the inquiry causing the inquiry to end abnormally.

Parameters:

`discType` - the type of request that was completed; either `INQUIRY_COMPLETED`, `INQUIRY_TERMINATED`, or `INQUIRY_ERROR`

See Also: `INQUIRY_COMPLETED`, `INQUIRY_TERMINATED`, `INQUIRY_ERROR`

servicesDiscovered(int, ServiceRecord[])

```
public void servicesDiscovered(int transID, ServiceRecord[] servRecord)
```

Called when service(s) are found during a service search.

Parameters:

`transID` - the transaction ID of the service search that is posting the result

`service` - a list of services found during the search request

See Also: `searchServices(int[], UUID[], RemoteDevice, DiscoveryListener)`

serviceSearchCompleted(int, int)

```
public void serviceSearchCompleted(int transID, int respCode)
```

Called when a service search is completed or was terminated because of an error. Legal status values in the `respCode` argument include `SERVICE_SEARCH_COMPLETED`, `SERVICE_SEARCH_TERMINATED`, `SERVICE_SEARCH_ERROR`, `SERVICE_SEARCH_NO_RECORDS` and `SERVICE_SEARCH_DEVICE_NOT_REACHABLE`. The following table describes when each `respCode` will be used:

`respCode`	Reason
`SERVICE_SEARCH_COMPLETED`	if the service search completed normally
`SERVICE_SEARCH_TERMINATED`	if the service search request was cancelled by a call to `DiscoveryAgent.cancelService-Search()`
`SERVICE_SEARCH_ERROR`	if an error occurred while processing the request
`SERVICE_SEARCH_NO_RECORDS`	if no records were found during the service search
`SERVICE_SEARCH_DEVICE_NOT_REACHABLE`	if the device specified in the search request could not be reached or the local device could not establish a connection to the remote device

Parameters:

`transID` - the transaction ID identifying the request which initiated the service search

`respCode` - the response code that indicates the status of the transaction

javax.bluetooth
L2CAPConnection

Declaration
```
public interface L2CAPConnection extends javax.microedition.io.Connection
```

All Superinterfaces: javax.microedition.io.Connection

Description
The `L2CAPConnection` interface represents a connection-oriented L2CAP channel. This interface is to be used as part of the CLDC Generic Connection Framework.

To create a client connection, the protocol is `btl2cap`. The target is the combination of the address of the Bluetooth device to connect to and the Protocol Service Multiplexor (PSM) of the service. The PSM value is used by the L2CAP to determine which higher level protocol or application is the recipient of the messages the layer receives.

The parameters defined specific to L2CAP are ReceiveMTU (Maximum Transmission Unit (MTU)) and TransmitMTU. The ReceiveMTU and TransmitMTU parameters are optional. ReceiveMTU specifies the maximum payload size this connection can accept, and TransmitMTU specifies the maximum payload size this connection can send. An example of a valid L2CAP client connection string is:
```
btl2cap://0050CD00321B:1003;ReceiveMTU=512;TransmitMTU=512
```

Member Summary

Fields

`public static final`	`DEFAULT_MTU` Default MTU value for connection-oriented channels is 672 bytes.
`public static final`	`MINIMUM_MTU` Minimum MTU value for connection-oriented channels is 48 bytes.

Methods

`public int`	`getReceiveMTU()` Returns the ReceiveMTU that the connection supports.
`public int`	`getTransmitMTU()` Returns the MTU that the remote device supports.
`public boolean`	`ready()` Determines if there is a packet that can be read via a call to `receive()`.
`public int`	`receive(byte[])` Reads a packet of data.
`public void`	`send(byte[])` Requests that data be sent to the remote device.

Inherited Member Summary
Methods inherited from interface javax.microedition.io.Connection
`close`

Fields

DEFAULT_MTU

```
public static final int DEFAULT_MTU
```

Default MTU value for connection-oriented channels is 672 bytes.

The value of `DEFAULT_MTU` is 0x02A0 (672).

MINIMUM_MTU

```
public static final int MINIMUM_MTU
```

Minimum MTU value for connection-oriented channels is 48 bytes.

The value of `MINIMUM_MTU` is 0x30 (48).

Methods

getReceiveMTU()

```
public int getReceiveMTU()
            throws IOException
```

Returns the ReceiveMTU that the connection supports. If the connection string did not specify a ReceiveMTU, the value returned will be less than or equal to the `DEFAULT_MTU`. Also, if the connection string did specify an MTU, this value will be less than or equal to the value specified in the connection string.

Returns: the maximum number of bytes that can be read in a single call to `receive()`

Throws:

`IOException` - if the connection is closed

getTransmitMTU()

```
public int getTransmitMTU()
            throws IOException
```

Returns the MTU that the remote device supports. This value is obtained after the connection has been configured. If the application had specified TransmitMTU in the `Connector.open()` string then this value should be equal to that. If the application did not specify any TransmitMTU, then this value should be less than or equal to the ReceiveMTU the remote device advertised during channel configuration.

Returns: the maximum number of bytes that can be sent in a single call to `send()` without losing any data

Throws:

`IOException` - if the connection is closed

ready()

```
public boolean ready()
            throws IOException
```

Determines if there is a packet that can be read via a call to `receive()`. If `true`, a call to `receive()` will not block the application.

Returns: `true` if there is data to read; `false` if there is no data to read

Throws:

`IOException` - if the connection is closed

See Also: `receive(byte[])`

receive(byte[])

```
public int receive(byte[] inBuf)
            throws IOException
```

Reads a packet of data. The amount of data received in this operation is related to the value of Receive-MTU. If the size of `inBuf` is greater than or equal to ReceiveMTU, then no data will be lost. Unlike `read()` on an `java.io.InputStream`, if the size of `inBuf` is smaller than ReceiveMTU, then the portion of the L2CAP payload that will fit into `inBuf` will be placed in `inBuf`, the rest will be discarded. If the application is aware of the number of bytes (less than ReceiveMTU) it will receive in any transaction, then the size of `inBuf` can be less than ReceiveMTU and no data will be lost. If `inBuf` is of length 0, all data sent in one packet is lost unless the length of the packet is 0.

Parameters:

`inBuf` - byte array to store the received data

Returns: the actual number of bytes read; 0 if a zero length packet is received; 0 if `inBuf` length is zero

Throws:

`IOException` - if an I/O error occurs or the connection has been closed

`InterruptedIOException` - if the request timed out

`NullPointerException` - if `inBuf` is `null`

send(byte[])

```
public void send(byte[] data)
            throws IOException
```

Requests that data be sent to the remote device. The TransmitMTU determines the amount of data that can be successfully sent in a single send operation. If the size of `data` is greater than the TransmitMTU, then only the first TransmitMTU bytes of the packet are sent, and the rest will be discarded. If `data` is of length 0, an empty L2CAP packet will be sent.

Parameters:

`data` - data to be sent

Throws:

`IOException` - if `data` cannot be sent successfully or if the connection is closed

`NullPointerException` - if the `data` is `null`

javax.bluetooth
L2CAPConnectionNotifier

Declaration
`public interface` **`L2CAPConnectionNotifier extends javax.microedition.io.Connection`**

All Superinterfaces: javax.microedition.io.Connection

Description
The `L2CAPConnectionNotifier` interface provides an L2CAP connection notifier.

To create a server connection, the protocol must be `btl2cap`. The target contains “localhost:” and the UUID of the service. The parameters are ReceiveMTU and TransmitMTU, the same parameters used to define a client connection. Here is an example of a valid server connection string:

```
btl2cap://
localhost:3B9FA89520078C303355AAA694238F07;ReceiveMTU=512;TransmitMTU=512
```

A call to Connector.open() with this string will return a `javax.bluetooth.L2CAPConnectionNotifier` object. An `L2CAPConnection` object is obtained from the `L2CAPConnectionNotifier` by calling the method `acceptAndOpen()`.

Member Summary	
Methods	
public `L2CAPConnection`	`acceptAndOpen()` Waits for a client to connect to this L2CAP service.

Inherited Member Summary
Methods inherited from interface javax.microedition.io.Connection
`close`

Methods

acceptAndOpen()

```
public L2CAPConnection acceptAndOpen()
            throws IOException
```

Waits for a client to connect to this L2CAP service. Upon connection returns an `L2CAPConnection` that can be used to communicate with this client.

A service record associated with this connection will be added to the SDDB associated with this `L2CAPConnectionNotifier` object if one does not exist in the SDDB. This method will put the local device in connectable mode so that it may respond to connection attempts by clients.

The following checks are done to verify that any modifications made by the application to the service record after it was created by `Connector.open()` have not created an invalid service record. If any of these checks fail, then a `ServiceRegistrationException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btl2cap` service record, must be present in the service record.
- L2CAP must be in the ProtocolDescriptorList.
- The PSM value must not have changed in the service record.

This method will not ensure that the service record created is a completely valid service record. It is the responsibility of the application to ensure that the service record follows all of the applicable syntactic and semantic rules for service record correctness.

Returns: a connection to communicate with the client

Throws:

`IOException` - if the notifier is closed before `acceptAndOpen()` is called

`ServiceRegistrationException` - if the structure of the associated service record is invalid or if the service record could not be added successfully to the local SDDB. The structure of service record is invalid if the service record is missing any mandatory service attributes, or has changed any of the values described above which are fixed and cannot be changed. Failures to add the record to the SDDB could be due to insufficient disk space, database locks, etc.

`BluetoothStateException` - if the server device could not be placed in connectable mode because the device user has configured the device to be non-connectable.

javax.bluetooth
LocalDevice

Declaration

```
public class LocalDevice

java.lang.Object
  |
  +--javax.bluetooth.LocalDevice
```

Description

The `LocalDevice` class defines the basic functions of the Bluetooth manager. The Bluetooth manager provides the lowest level of interface possible into the Bluetooth stack. It provides access to and control of the local Bluetooth device.

This class produces a singleton object.

Member Summary

Methods

`public String`	`getBluetoothAddress()` Retrieves the Bluetooth address of the local device.
`public DeviceClass`	`getDeviceClass()` Retrieves the `DeviceClass` object that represents the service classes, major device class, and minor device class of the local device.
`public int`	`getDiscoverable()` Retrieves the local device's discoverable mode.
`public DiscoveryAgent`	`getDiscoveryAgent()` Returns the discovery agent for this device.
`public String`	`getFriendlyName()` Retrieves the name of the local device.
`public static LocalDevice`	`getLocalDevice()` Retrieves the `LocalDevice` object for the local Bluetooth device.
`public static String`	`getProperty(String)` Retrieves Bluetooth system properties.
`public ServiceRecord`	`getRecord(Connection)` Gets the service record corresponding to a `btspp`, `btl2cap`, or `btgoep` notifier.
`public boolean`	`setDiscoverable(int)` Sets the discoverable mode of the device.
`public void`	`updateRecord(ServiceRecord)` Updates the service record in the local SDDB that corresponds to the `ServiceRecord` parameter.

Inherited Member Summary

Methods inherited from class java.lang.Object

Inherited Member Summary
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getBluetoothAddress()

```
public java.lang.String getBluetoothAddress()
```

Retrieves the Bluetooth address of the local device. The Bluetooth address will never be `null`. The Bluetooth address will be 12 characters long. Valid characters are 0-9 and A-F.

Returns: the Bluetooth address of the local device

getDeviceClass()

```
public DeviceClass getDeviceClass()
```

Retrieves the `DeviceClass` object that represents the service classes, major device class, and minor device class of the local device. This method will return `null` if the service classes, major device class, or minor device class could not be determined.

Returns: the service classes, major device class, and minor device class of the local device, or `null` if the service classes, major device class or minor device class could not be determined

getDiscoverable()

```
public int getDiscoverable()
```

Retrieves the local device's discoverable mode. The return value will be `DiscoveryAgent.GIAC`, `DiscoveryAgent.LIAC`, `DiscoveryAgent.NOT_DISCOVERABLE`, or a value in the range 0x9E8B00 to 0x9E8B3F.

Returns: the discoverable mode the device is presently in

See Also: `GIAC`, `LIAC`, `NOT_DISCOVERABLE`

getDiscoveryAgent()

```
public DiscoveryAgent getDiscoveryAgent()
```

Returns the discovery agent for this device. Multiple calls to this method will return the same object. This method will never return `null`.

Returns: the discovery agent for the local device

getFriendlyName()

```
public java.lang.String getFriendlyName()
```

Retrieves the name of the local device. The Bluetooth specification calls this name the "Bluetooth device name" or the "user-friendly name".

Returns: the name of the local device; `null` if the name could not be retrieved

getLocalDevice()

```
public static LocalDevice getLocalDevice()
            throws BluetoothStateException
```

Retrieves the `LocalDevice` object for the local Bluetooth device. Multiple calls to this method will return the same object. This method will never return `null`.

Returns: an object that represents the local Bluetooth device

Throws:

`BluetoothStateException` - if the Bluetooth system could not be initialized

getProperty(String)

```
public static java.lang.String getProperty(java.lang.String property)
```

Retrieves Bluetooth system properties. The following properties must be supported, but additional values are allowed:

Property Name	Description
bluetooth.api.version	The version of the Java API for Bluetooth wireless technology that is supported. For this version it will be set to “1.0”.
bluetooth.master.switch	Is master/slave switch allowed? Valid values are either “true” or “false”.
bluetooth.sd.attr.retrievable.max	Maximum number of service attributes to be retrieved per service record. The string will be in Base 10 digits.
bluetooth.connected.devices.max	The maximum number of connected devices supported. This number may be greater than 7 if the implementation handles parked connections. The string will be in Base 10 digits.
bluetooth.l2cap.receiveMTU.max	The maximum ReceiveMTU size in bytes supported in L2CAP. The string will be in Base 10 digits, e.g. “32”.
bluetooth.sd.trans.max	Maximum number of concurrent service discovery transactions. The string will be in Base 10 digits.
bluetooth.connected.inquiry.scan	Is Inquiry scanning allowed during connection? Valid values are either “true” or “false”.
bluetooth.connected.page.scan	Is Page scanning allowed during connection? Valid values are either “true” or “false”.
bluetooth.connected.inquiry	Is Inquiry allowed during a connection? Valid values are either “true” or “false”.
bluetooth.connected.page	Is paging allowed during a connection? In other words, can a connection be established to one device if it is already connected to another device. Valid values are either “true” or “false”.

Parameters:

`property` - the property to retrieve as defined in this class.

Returns: the value of the property specified; `null` if the `property` is not defined

getRecord(Connection)

```
public ServiceRecord getRecord(javax.microedition.io.Connection notifier)
```

Gets the service record corresponding to a `btspp`, `btl2cap`, or `btgoep` notifier. In the case of a run-before-connect service, the service record returned by `getRecord()` was created by the same call to `Connector.open()` that created the `notifier`.

If a connect-anytime server application does not already have a service record in the SDDB, either because a service record for this service was never added to the SDDB or because the service record was added and then removed, then the `ServiceRecord` returned by `getRecord()` was created by the same call to `Connector.open()` that created the notifier.

In the case of a connect-anytime service, there may be a service record in the SDDB corresponding to this service prior to application startup. In this case, the `getRecord()` method must return a `ServiceRecord` whose contents match those of the corresponding service record in the SDDB. If a connect-anytime server application made changes previously to its service record in the SDDB (for example, during a previous execution of the server), and that service record is still in the SDDB, then those changes must be reflected in the `ServiceRecord` returned by `getRecord()`.

Two invocations of this method with the same `notifier` argument return objects that describe the same service attributes, but the return values may be different object references.

Parameters:

`notifier` - a connection that waits for clients to connect to a Bluetooth service

Returns: the `ServiceRecord` associated with `notifier`

Throws:

`IllegalArgumentException` - if `notifier` is closed, or if `notifier` does not implement one of the following interfaces: `javax.microedition.io.StreamConnectionNotifier`, `javax.bluetooth.L2CapConnectionNotifier`, or `javax.obex.SessionNotifier`. This exception is also thrown if `notifier` is not a Bluetooth notifier, e.g., a `StreamConnectionNotifier` created with a scheme other than `btspp`.

`NullPointerException` - if `notifier` is `null`

setDiscoverable(int)

```
public boolean setDiscoverable(int mode)
            throws BluetoothStateException
```

Sets the discoverable mode of the device. The `mode` may be any number in the range 0x9E8B00 to 0x9E8B3F as defined by the Bluetooth Assigned Numbers Document. When this specification was defined, only GIAC (`DiscoveryAgent.GIAC`) and LIAC (`DiscoveryAgent.LIAC`) were defined, but Bluetooth profiles may add additional access codes in the future. To determine what values may be used, check the Bluetooth Assigned Numbers document at http://www.bluetooth.org/assigned-numbers/baseband.htm. If `DiscoveryAgent.GIAC` or `DiscoveryAgent.LIAC` are provided, then this method will attempt to put the device into general or limited discoverable mode, respectively. To take a device out of discoverable mode, provide the `DiscoveryAgent.NOT_DISCOVERABLE` flag. The BCC decides if the request will be granted. In addition to the BCC, the Bluetooth system could effect the discoverability of a device.

According to the Bluetooth Specification, a device should only be limited discoverable (`DiscoveryAgent.LIAC`) for 1 minute. This is handled by the implementation of the API. After the minute is up, the device will revert back to the previous discoverable mode.

Parameters:

`mode` - the mode the device should be in; valid modes are `DiscoveryAgent.GIAC`, `DiscoveryAgent.LIAC`, `DiscoveryAgent.NOT_DISCOVERABLE` and any value in the range 0x9E8B00 to 0x9E8B3F

Returns: `true` if the request succeeded, otherwise `false` if the request failed because the BCC denied the request; `false` if the Bluetooth system does not support the access mode specified in `mode`

Throws:

`IllegalArgumentException` - if the `mode` is not `DiscoveryAgent.GIAC`, `DiscoveryAgent.LIAC`, `DiscoveryAgent.NOT_DISCOVERABLE`, or in the range 0x9E8B00 to 0x9E8B3F

`BluetoothStateException` - if the Bluetooth system is in a state that does not allow the discoverable mode to be changed

See Also: `GIAC`, `LIAC`, `NOT_DISCOVERABLE`

updateRecord(ServiceRecord)

```
public void updateRecord(ServiceRecord srvRecord)
            throws ServiceRegistrationException
```

Updates the service record in the local SDDB that corresponds to the `ServiceRecord` parameter. Updating is possible only if `srvRecord` was obtained using the `getRecord()` method. The service record in the SDDB is modified to have the same service attributes with the same contents as `srvRecord`.

If `srvRecord` was obtained from the SDDB of a remote device using the service search methods, updating is not possible and this method will throw an `IllegalArgumentException`.

If the `srvRecord` parameter is a `btspp` service record, then before the SDDB is changed the following checks are performed. If any of these checks fail, then an `IllegalArgumentException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btspp` service record, must be present in `srvRecord`.
- L2CAP and RFCOMM must be in the ProtocolDescriptorList.
- `srvRecord` must not have changed the RFCOMM server channel number from the channel number that is currently in the SDDB version of this service record.

If the `srvRecord` parameter is a `btl2cap` service record, then before the SDDB is changed the following checks are performed. If any of these checks fail, then an `IllegalArgumentException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btl2cap` service record, must be present in `srvRecord`.
- L2CAP must be in the ProtocolDescriptorList.
- `srvRecord` must not have changed the PSM value from the PSM value that is currently in the SDDB version of this service record.

If the `srvRecord` parameter is a `btgoep` service record, then before the SDDB is changed the following checks are performed. If any of these checks fail, then an `IllegalArgumentException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btgoep` service record, must be present in `srvRecord`.
- L2CAP, RFCOMM and OBEX must all be in the ProtocolDescriptorList.

- `srvRecord` must not have changed the RFCOMM server channel number from the channel number that is currently in the SDDB version of this service record.

`updateRecord()` is not required to ensure that `srvRecord` is a completely valid service record. It is the responsibility of the application to ensure that `srvRecord` follows all of the applicable syntactic and semantic rules for service record correctness.

If there is currently no SDDB version of the `srvRecord` service record, then this method will do nothing.

Parameters:

`srvRecord` - the new contents to use for the service record in the SDDB

Throws:

`NullPointerException` - if `srvRecord` is `null`

`IllegalArgumentException` - if the structure of the `srvRecord` is missing any mandatory service attributes, or if an attempt has been made to change any of the values described as fixed.

`ServiceRegistrationException` - if the local SDDB could not be updated successfully due to insufficient disk space, database locks, etc.

javax.bluetooth
RemoteDevice

Declaration

```
public class RemoteDevice

java.lang.Object
  |
  +--javax.bluetooth.RemoteDevice
```

Description

The `RemoteDevice` class represents a remote Bluetooth device. It provides basic information about a remote device including the device's Bluetooth address and its friendly name.

Member Summary

Constructors

protected	`RemoteDevice(String)` Creates a Bluetooth device based upon its address.

Methods

public boolean	`authenticate()` Attempts to authenticate this `RemoteDevice`.
public boolean	`authorize(Connection)` Determines if this `RemoteDevice` should be allowed to continue to access the local service provided by the `Connection`.
public boolean	`encrypt(Connection, boolean)` Attempts to turn encryption on or off for an existing connection.
public boolean	`equals(Object)` Determines if two `RemoteDevice`s are equal.
public final String	`getBluetoothAddress()` Retrieves the Bluetooth address of this device.
public String	`getFriendlyName(boolean)` Returns the name of this device.
public static RemoteDevice	`getRemoteDevice(Connection)` Retrieves the Bluetooth device that is at the other end of the Bluetooth Serial Port Profile connection, L2CAP connection, or OBEX over RFCOMM connection provided.
public int	`hashCode()` Computes the hash code for this object.
public boolean	`isAuthenticated()` Determines if this `RemoteDevice` has been authenticated.
public boolean	`isAuthorized(Connection)` Determines if this `RemoteDevice` has been authorized previously by the BCC of the local device to exchange data related to the service associated with the connection.
public boolean	`isEncrypted()` Determines if data exchanges with this `RemoteDevice` are currently being encrypted.
public boolean	`isTrustedDevice()` Determines if this is a trusted device according to the BCC.

Inherited Member Summary
Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Constructors

RemoteDevice(String)

```
protected RemoteDevice(java.lang.String address)
```

Creates a Bluetooth device based upon its address. The Bluetooth address must be 12 hex characters long. Valid characters are 0-9, a-f, and A-F. There is no preceding "0x" in the string. For example, valid Bluetooth addresses include but are not limited to:

```
008037144297
00af8300cd0b
014bd91DA8FC
```

Parameters:

`address` - the address of the Bluetooth device as a 12 character hex string

Throws:

`NullPointerException` - if `address` is `null`

`IllegalArgumentException` - if `address` is the address of the local device or is not a valid Bluetooth address

Methods

authenticate()

```
public boolean authenticate()
            throws IOException
```

Attempts to authenticate this `RemoteDevice`. Authentication is a means of verifying the identity of a remote device. Authentication involves a device-to-device challenge and response scheme that requires a 128-bit common secret link key derived from a PIN code shared by both devices. If either side's PIN code does not match, the authentication process fails and the method returns `false`. The method will also return `false` if authentication is incompatible with the current security settings of the local device established by the BCC, if the stack does not support authentication at all, or if the stack does not support authentication subsequent to connection establishment.

If this `RemoteDevice` has previously been authenticated, then this method returns `true` without attempting to re-authenticate this `RemoteDevice`.

Returns: `true` if authentication is successful; otherwise `false`

Throws:

`IOException` - if there are no open connections between the local device and this `RemoteDevice`

authorize(Connection)

```
public boolean authorize(javax.microedition.io.Connection conn)
            throws IOException
```

Determines if this `RemoteDevice` should be allowed to continue to access the local service provided by the `Connection`. In Bluetooth, authorization is defined as the process of deciding if device X is allowed to access service Y. The implementation of the `authorize(Connection conn)` method asks the Bluetooth Control Center (BCC) to decide if it is acceptable for `RemoteDevice` to continue to access a local service over the connection `conn`. In devices with a user interface, the BCC is expected to consult with the user to obtain approval.

Some Bluetooth systems may allow the user to permanently authorize a remote device for all local services. When a device is authorized in this way, it is known as a "trusted device" —- see `isTrustedDevice()`.

The `authorize()` method will also check that the identity of the `RemoteDevice` can be verified through authentication. If this `RemoteDevice` has been authorized for `conn` previously, then this method returns `true` without attempting to re-authorize this `RemoteDevice`.

Parameters:

`conn` - the connection that this `RemoteDevice` is using to access a local service

Returns: `true` if this `RemoteDevice` is successfully authenticated and authorized, otherwise `false` if authentication or authorization fails

Throws:

`IllegalArgumentException` - if `conn` is not a connection to this `RemoteDevice`, or if the local device initiated the connection, i.e., the local device is the client rather than the server. This exception is also thrown if `conn` was created by `RemoteDevice` using a scheme other than `btspp`, `btl2cap`, or `btgoep`. This exception is thrown if `conn` is a notifier used by a server to wait for a client connection, since the notifier is not a connection to this `RemoteDevice`.

`IOException` - if `conn` is closed

See Also: `isTrustedDevice()`

encrypt(Connection, boolean)

```
public boolean encrypt(javax.microedition.io.Connection conn, boolean on)
            throws IOException
```

Attempts to turn encryption on or off for an existing connection. In the case where the parameter `on` is `true`, this method will first authenticate this `RemoteDevice` if it has not already been authenticated. Then it will attempt to turn on encryption. If the connection is already encrypted then this method returns `true`. Otherwise, when the parameter `on` is `true`, either:

- the method succeeds in turning on encryption for the connection and returns `true`, or
- the method was unsuccessful in turning on encryption and returns `false`. This could happen because the stack does not support encryption or because encryption conflicts with the user's security settings for the device.

In the case where the parameter `on` is `false`, there are again two possible outcomes:

- encryption is turned off on the connection and `true` is returned, or
- encryption is left on for the connection and `false` is returned.

Encryption may be left on following `encrypt(conn, false)` for a variety of reasons. The user's current security settings for the device may require encryption or the stack may not have a mechanism to turn off encryption. Also, the BCC may have determined that encryption will be kept on for the physical link to this `RemoteDevice`. The details of the BCC are implementation dependent, but encryption might be left on because other connections to the same device need encryption. (All of the connections over the same physical link must be encrypted if any of them are encrypted.)

While attempting to turn encryption off may not succeed immediately because other connections need encryption on, there may be a delayed effect. At some point, all of the connections over this physical link needing encryption could be closed or also have had the method `encrypt(conn, false)` invoked for them. In this case, the BCC may turn off encryption for all connections over this physical link. (The policy used by the BCC is implementation dependent.) It is recommended that applications do `encrypt(conn, false)` once they no longer need encryption to allow the BCC to determine if it can reduce the overhead on connections to this `RemoteDevice`.

The fact that `encrypt(conn, false)` may not succeed in turning off encryption has very few consequences for applications. The stack handles encryption and decryption, so the application does not have to do anything different depending on whether the connection is still encrypted or not.

Parameters:

`conn` - the connection whose need for encryption has changed

`on` - `true` attempts to turn on encryption; `false` attempts to turn off encryption

Returns: `true` if the change succeeded, otherwise `false` if it failed

Throws:

`IOException` - if `conn` is closed

`IllegalArgumentException` - if `conn` is not a connection to this `RemoteDevice`; if `conn` was created by the client side of the connection using a scheme other than `btspp`, `btl2cap`, or `btgoep` (for example, this exception will be thrown if `conn` was created using the `file` or `http` schemes.); if `conn` is a notifier used by a server to wait for a client connection, since the notifier is not a connection to this `RemoteDevice`

equals(Object)

```
public boolean equals(java.lang.Object obj)
```

Determines if two `RemoteDevice`s are equal. Two devices are equal if they have the same Bluetooth device address.

Overrides: java.lang.Object.equals(java.lang.Object) in class java.lang.Object

Parameters:

`obj` - the object to compare to

Returns: `true` if both devices have the same Bluetooth address; `false` if both devices do not have the same address; `false` if `obj` is `null`; `false` if `obj` is not a `RemoteDevice`

getBluetoothAddress()

```
public final java.lang.String getBluetoothAddress()
```

Retrieves the Bluetooth address of this device. The Bluetooth address will be 12 characters long. Valid characters are 0-9 and A-F. This method will never return `null`.

Returns: the Bluetooth address of the remote device

getFriendlyName(boolean)

```
public java.lang.String getFriendlyName(boolean alwaysAsk)
            throws IOException
```

Returns the name of this device. The Bluetooth specification calls this name the “Bluetooth device name” or the “user-friendly name”. This method will only contact the remote device if the name is not known or `alwaysAsk` is `true`.

Parameters:

`alwaysAsk` - if `true` then the device will be contacted for its name, otherwise, if there exists a known name for this device, the name will be returned without contacting the remote device

Returns: the name of the device, or `null` if the Bluetooth system does not support this feature; if the local device is able to contact the remote device, the result will never be `null`; if the remote device does not have a name then an empty string will be returned

Throws:

`IOException` - if the remote device can not be contacted or the remote device could not provide its name

getRemoteDevice(Connection)

```
public static RemoteDevice getRemoteDevice(javax.microedition.io.Connection conn)
            throws IOException
```

Retrieves the Bluetooth device that is at the other end of the Bluetooth Serial Port Profile connection, L2CAP connection, or OBEX over RFCOMM connection provided. This method will never return `null`.

Parameters:

`conn` - the Bluetooth Serial Port connection, L2CAP connection, or OBEX over RFCOMM connection whose remote Bluetooth device is needed

Returns: the remote device involved in the connection

Throws:

`IllegalArgumentException` - if `conn` is not a Bluetooth Serial Port Profile connection, L2CAP connection, or OBEX over RFCOMM connection; if `conn` is a `L2CAPConnectionNotifier`, `StreamConnectionNotifier`, or `SessionNotifier`

`IOException` - if the connection is closed

`NullPointerException` - if `conn` is `null`

hashCode()

```
public int hashCode()
```

Computes the hash code for this object. This method will return the same value when it is called multiple times on the same object.

Overrides: java.lang.Object.hashCode() in class java.lang.Object

Returns: the hash code for this object

isAuthenticated()

```
public boolean isAuthenticated()
```

Determines if this `RemoteDevice` has been authenticated.

A device may have been authenticated by this application or another application. Authentication applies to an ACL link between devices and not on a specific L2CAP, RFCOMM, or OBEX connection. Therefore, if `authenticate()` is performed when an L2CAP connection is made to device A, then `isAuthenticated()` may return `true` when tested as part of making an RFCOMM connection to device A.

Returns: `true` if this `RemoteDevice` has previously been authenticated; `false` if it has not been authenticated or there are no open connections between the local device and this `RemoteDevice`

isAuthorized(Connection)

```
public boolean isAuthorized(javax.microedition.io.Connection conn)
            throws IOException
```

Determines if this `RemoteDevice` has been authorized previously by the BCC of the local device to exchange data related to the service associated with the connection. Both clients and servers can call this method. However, for clients this method returns `false` for all legal values of the `conn` argument.

Parameters:

`conn` - a connection that this `RemoteDevice` is using to access a service or provide a service

Returns: `true` if `conn` is a server-side connection and this `RemoteDevice` has been authorized; `false` if `conn` is a client-side connection, or a server-side connection that has not been authorized

Throws:

`IllegalArgumentException` - if `conn` is not a connection to this `RemoteDevice`; if `conn` was not created using one of the schemes `btspp`, `btl2cap`, or `btgoep`; or if `conn` is a notifier used by a server to wait for a client connection, since the notifier is not a connection to this `RemoteDevice`.

`IOException` - if `conn` is closed

isEncrypted()

```
public boolean isEncrypted()
```

Determines if data exchanges with this `RemoteDevice` are currently being encrypted.

Encryption may have been previously turned on by this or another application. Encryption applies to an ACL link between devices and not on a specific L2CAP, RFCOMM, or OBEX connection. Therefore, if `encrypt()` is performed with the `on` parameter set to `true` when an L2CAP connection is made to device A, then `isEncrypted()` may return `true` when tested as part of making an RFCOMM connection to device A.

Returns: `true` if data exchanges with this `RemoteDevice` are being encrypted; `false` if they are not being encrypted, or there are no open connections between the local device and this `RemoteDevice`

isTrustedDevice()

```
public boolean isTrustedDevice()
```

Determines if this is a trusted device according to the BCC.

Returns: `true` if the device is a trusted device, otherwise `false`

javax.bluetooth
ServiceRecord

Declaration

```
public interface ServiceRecord
```

Description

The `ServiceRecord` interface describes characteristics of a Bluetooth service. A `ServiceRecord` contains a set of service attributes, where each service attribute is an (ID, value) pair. A Bluetooth attribute ID is a 16-bit unsigned integer, and an attribute value is a `DataElement`.

The structure and use of service records is specified by the Bluetooth specification in the Service Discovery Protocol (SDP) document. Most of the Bluetooth Profile specifications also describe the structure of the service records used by the Bluetooth services that conform to the profile.

An SDP Server maintains a Service Discovery Database (SDDB) of service records that describe the services on the local device. Remote SDP clients can use the SDP to query an SDP server for any service records of interest. A service record provides sufficient information to allow an SDP client to connect to the Bluetooth service on the SDP server's device.

`ServiceRecord`s are made available to a client application via an argument of the `servicesDiscovered` method of the `DiscoveryListener` interface. `ServiceRecord`s are available to server applications via the method `getRecord()` on `LocalDevice`.

There might be many service attributes in a service record, and the SDP protocol makes it possible to specify the subset of the service attributes that an SDP client wants to retrieve from a remote service record. The `ServiceRecord` interface treats certain service attribute IDs as default IDs, and, if present, these service attributes are automatically retrieved during service searches.

The Bluetooth Assigned Numbers document (http://www.bluetooth.org/assigned-numbers/sdp.htm) defines a large number of service attribute IDs. Here is a subset of the most common service attribute IDs and their types.

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer
ServiceClassIDList	0x0001	DATSEQ of UUIDs
ServiceRecordState	0x0002	32-bit unsigned integer
ServiceID	0x0003	UUID
ProtocolDescriptorList	0x0004	DATSEQ of DATSEQ of UUID and optional parameters
BrowseGroupList	0x0005	DATSEQ of UUIDs
LanguageBasedAttributeIDList	0x0006	DATSEQ of DATSEQ triples
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer
ServiceAvailability	0x0008	8-bit unsigned integer
BluetoothProfileDescriptorList	0x0009	DATSEQ of DATSEQ pairs

DocumentationURL	0x000A	URL
ClientExecutableURL	0x000B	URL
IconURL	0x000C	URL
VersionNumberList	0x0200	DATSEQ of 16-bit unsigned integers
ServiceDatabaseState	0x0201	32-bit unsigned integer

The following table lists the common string-valued attribute ID offsets used in a `ServiceRecord`. These offsets must be added to a base value to obtain the actual service ID. (For more information, see the Service Discovery Protocol Specification located in the Bluetooth Core Specification at http://www.bluetooth.com/dev/specifications.asp).

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceName	0x0000	String
ServiceDescription	0x0001	String
ProviderName	0x0002	String

Member Summary

Fields

`public static final` **`AUTHENTICATE_ENCRYPT`**
Authentication and encryption are required for connections to this service.

`public static final` **`AUTHENTICATE_NOENCRYPT`**
Authentication is required for connections to this service, but not encryption.

`public static final` **`NOAUTHENTICATE_NOENCRYPT`**
Authentication and encryption are not needed on a connection to this service.

Methods

`public int` **`getAttributeIDs()`**
Returns the service attribute IDs whose value could be retrieved by a call to `getAttributeValue()`.

`public DataElement` **`getAttributeValue(int)`**
Returns the value of the service attribute ID provided it is present in the service record, otherwise this method returns `null`.

`public String` **`getConnectionURL(int, boolean)`**
Returns a String including optional parameters that can be used by a client to connect to the service described by this `ServiceRecord`.

`public RemoteDevice` **`getHostDevice()`**
Returns the remote Bluetooth device that populated the service record with attribute values.

`public boolean` **`populateRecord(int[])`**
Retrieves the values by contacting the remote Bluetooth device for a set of service attribute IDs of a service that is available on a Bluetooth device.

`public boolean` **`setAttributeValue(int, DataElement)`**
Modifies this `ServiceRecord` to contain the service attribute defined by the attribute-value pair `(attrID, attrValue)`.

Member Summary	
public void	setDeviceServiceClasses(int) Used by a server application to indicate the major service class bits that should be activated in the server's DeviceClass when this ServiceRecord is added to the SDDB.

Fields

AUTHENTICATE_ENCRYPT

```
public static final int AUTHENTICATE_ENCRYPT
```

Authentication and encryption are required for connections to this service. Used with `getConnectionURL()` method.

`AUTHENTICATE_ENCRYPT` is set to the constant value 0x02 (2).

See Also: `getConnectionURL(int, boolean)`

AUTHENTICATE_NOENCRYPT

```
public static final int AUTHENTICATE_NOENCRYPT
```

Authentication is required for connections to this service, but not encryption. It is OK for encryption to be either on or off for the connection. Used with `getConnectionURL()` method.

`AUTHENTICATE_NOENCRYPT` is set to the constant value 0x01 (1).

See Also: `getConnectionURL(int, boolean)`

NOAUTHENTICATE_NOENCRYPT

```
public static final int NOAUTHENTICATE_NOENCRYPT
```

Authentication and encryption are not needed on a connection to this service. Used with `getConnectionURL()` method.

`NOAUTHENTICATE_NOENCRYPT` is set to the constant value 0x00 (0).

See Also: `getConnectionURL(int, boolean)`

Methods

getAttributeIDs()

```
public int[] getAttributeIDs()
```

Returns the service attribute IDs whose value could be retrieved by a call to `getAttributeValue()`. The list of attributes being returned is not sorted and includes default attributes.

Returns: an array of service attribute IDs that are in this object and have values for them; if there are no attribute IDs that have values, this method will return an array of length zero.

See Also: `getAttributeValue(int)`

getAttributeValue(int)

```
public DataElement getAttributeValue(int attrID)
```

Returns the value of the service attribute ID provided it is present in the service record, otherwise this method returns `null`.

Parameters:

`attrID` - the attribute whose value is to be returned

Returns: the value of the attribute ID if present in the service record, otherwise `null`

Throws:

`IllegalArgumentException` - if `attrID` is negative or greater than or equal to 2^{16}

getConnectionURL(int, boolean)

```
public java.lang.String getConnectionURL(int requiredSecurity, boolean mustBeMaster)
```

Returns a String including optional parameters that can be used by a client to connect to the service described by this `ServiceRecord`. The return value can be used as the first argument to `Connector.open()`. In the case of a Serial Port service record, this string might look like "btspp://0050CD00321B:3;authenticate=true;encrypt=false;master=true", where "0050CD00321B" is the Bluetooth address of the device that provided this `ServiceRecord`, "3" is the RFCOMM server channel mentioned in this `ServiceRecord`, and there are three optional parameters related to security and master/slave roles.

If this method is called on a `ServiceRecord` returned from `LocalDevice.getRecord()`, it will return the connection string that a remote device will use to connect to this service.

Parameters:

`requiredSecurity` - determines whether authentication or encryption are required for a connection

`mustBeMaster` - `true` indicates that this device must play the role of master in connections to this service; `false` indicates that the local device is willing to be either the master or the slave

Returns: a string that can be used to connect to the service or `null` if the ProtocolDescriptorList in this ServiceRecord is not formatted according to the Bluetooth specification

Throws:

`IllegalArgumentException` - if `requiredSecurity` is not one of the constants `NOAUTHENTICATE_NOENCRYPT`, `AUTHENTICATE_NOENCRYPT`, or `AUTHENTICATE_ENCRYPT`

See Also: `NOAUTHENTICATE_NOENCRYPT`, `AUTHENTICATE_NOENCRYPT`, `AUTHENTICATE_ENCRYPT`

getHostDevice()

```
public RemoteDevice getHostDevice()
```

Returns the remote Bluetooth device that populated the service record with attribute values. It is important to note that the Bluetooth device that provided the value might not be reachable anymore, since it can move, turn off, or change its security mode denying all further transactions.

Returns: the remote Bluetooth device that populated the service record, or `null` if the local device populated this `ServiceRecord`

populateRecord(int[])

```
public boolean populateRecord(int[] attrIDs)
            throws IOException
```

Retrieves the values by contacting the remote Bluetooth device for a set of service attribute IDs of a service that is available on a Bluetooth device. (This involves going over the air and contacting the remote device for the attribute values.) The system might impose a limit on the number of service attribute ID values one can request at a time. Applications can obtain the value of this limit as a String by calling `LocalDevice.getProperty("bluetooth.sd.attr.retrievable.max")`. The method is blocking and will return when the results of the request are available. Attribute IDs whose values could be obtained are added to this service record. If there exist attribute IDs for which values are retrieved this will cause the old values to be overwritten. If the remote device cannot be reached, an `IOException` will be thrown.

Parameters:

`attrIDs` - the list of service attributes IDs whose value are to be retrieved; the number of attributes cannot exceed the property `bluetooth.sd.attr.retrievable.max`; the attributes in the request must be legal, i.e. their values are in the range of [0, 2^{16}-1]. The input attribute IDs can include attribute IDs from the default attribute set too.

Returns: `true` if the request was successful in retrieving values for some or all of the attribute IDs; `false` if it was unsuccessful in retrieving any values

Throws:

`IOException` - if the local device is unable to connect to the remote Bluetooth device that was the source of this `ServiceRecord`; if this `ServiceRecord` was deleted from the SDDB of the remote device

`IllegalArgumentException` - if the size of `attrIDs` exceeds the system specified limit as defined by `bluetooth.sd.attr.retrievable.max`; if the `attrIDs` array length is zero; if any of their values are not in the range of [0, 2^{16}-1]; if `attrIDs` has duplicate values

`NullPointerException` - if `attrIDs` is `null`

`RuntimeException` - if this `ServiceRecord` describes a service on the local device rather than a service on a remote device

setAttributeValue(int, DataElement)

```
public boolean setAttributeValue(int attrID, DataElement attrValue)
```

Modifies this `ServiceRecord` to contain the service attribute defined by the attribute-value pair `(attrID, attrValue)`. If the `attrID` does not exist in the `ServiceRecord`, this attribute-value pair is added to this `ServiceRecord` object. If the `attrID` is already in this `ServiceRecord`, the value of the attribute is changed to `attrValue`. If `attrValue` is `null`, the attribute with the attribute ID of `attrID` is removed from this `ServiceRecord` object. If `attrValue` is `null` and `attrID` does not exist in this object, this method will return `false`.

This method makes no modifications to a service record in the SDDB. In order for any changes made by this method to be reflected in the SDDB, a call must be made to the `acceptAndOpen()` method of the associated notifier to add this `ServiceRecord` to the SDDB for the first time, or a call must be made to the `updateRecord()` method of `LocalDevice` to modify the version of this `ServiceRecord` that is already in the SDDB.

This method prevents the ServiceRecordHandle from being modified by throwing an `IllegalArgumentException`.

Parameters:

`attrID` - the service attribute ID

`attrValue` - the `DataElement` which is the value of the service attribute

Returns: `true` if the service attribute was successfully added, removed, or modified; `false` if `attrValue` is `null` and `attrID` is not in this object

Throws:

`IllegalArgumentException` - if `attrID` does not represent a 16-bit unsigned integer; if `attrID` is the value of ServiceRecordHandle (0x0000)

`RuntimeException` - if this method is called on a `ServiceRecord` that was created by a call to `DiscoveryAgent.searchServices()`

setDeviceServiceClasses(int)

```
public void setDeviceServiceClasses(int classes)
```

Used by a server application to indicate the major service class bits that should be activated in the server's `DeviceClass` when this `ServiceRecord` is added to the SDDB. When client devices do device discovery, the server's `DeviceClass` is provided as one of the arguments of the `deviceDiscovered` method of the `DiscoveryListener` interface. Client devices can consult the `DeviceClass` of the server device to get a general idea of the kind of device this is (e.g., phone, PDA, or PC) and the major service classes it offers (e.g., rendering, telephony, or information). A server application should use the `setDeviceServiceClasses` method to describe its service in terms of the major service classes. This allows clients to obtain a `DeviceClass` for the server that accurately describes all of the services being offered.

When `acceptAndOpen()` is invoked for the first time on the notifier associated with this `ServiceRecord`, the `classes` argument from the `setDeviceServiceClasses` method is OR'ed with the current setting of the major service class bits of the local device. The OR operation potentially activates additional bits. These bits may be retrieved by calling `getDeviceClass()` on the `LocalDevice` object. Likewise, a call to `LocalDevice.updateRecord()` will cause the major service class bits to be OR'ed with the current settings and updated.

The documentation for `DeviceClass` gives examples of the integers that describe each of the major service classes and provides a URL for the complete list. These integers can be used individually or OR'ed together to describe the appropriate value for `classes`.

Later, when this `ServiceRecord` is removed from the SDDB, the implementation will automatically deactivate the device bits that were activated as a result of the call to `setDeviceServiceClasses`. The only exception to this occurs if there is another `ServiceRecord` that is in the SDDB and `setDeviceServiceClasses` has been sent to that other `ServiceRecord` to request that some of the same bits be activated.

Parameters:

`classes` - an integer whose binary representation indicates the major service class bits that should be activated

Throws:

`IllegalArgumentException` - if `classes` is not an OR of one or more of the major service class integers in the Bluetooth Assigned Numbers document. While Limited Discoverable Mode is included in this list of major service classes, its bit is activated by placing the device in Limited Discoverable Mode (see the GAP specification), so if bit 13 is set this exception will be thrown.

`RuntimeException` - if the `ServiceRecord` receiving the message was obtained from a remote device

javax.bluetooth
ServiceRegistrationException

Declaration
```
public class ServiceRegistrationException extends java.io.IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--javax.bluetooth.ServiceRegistrationException
```

Description
The `ServiceRegistrationException` is thrown when there is a failure to add a service record to the local Service Discovery Database (SDDB) or to modify an existing service record in the SDDB. The failure could be because the SDDB has no room for new records or because the modification being attempted to a service record violated one of the rules about service record updates. This exception will also be thrown if it was not possible to obtain an RFCOMM server channel needed for a `btspp` service record.

Member Summary	
Constructors	
`public`	`ServiceRegistrationException()` Creates a `ServiceRegistrationException` without a detailed message.
`public`	`ServiceRegistrationException(String)` Creates a `ServiceRegistrationException` with a detailed message.

Inherited Member Summary
Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`
Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Constructors

ServiceRegistrationException()
```
public ServiceRegistrationException()
```

Creates a `ServiceRegistrationException` without a detailed message.

ServiceRegistrationException(String)

```
public ServiceRegistrationException(java.lang.String msg)
```

Creates a `ServiceRegistrationException` with a detailed message.

Parameters:

`msg` - the reason for the exception

javax.bluetooth
UUID

Declaration

```
public class UUID

java.lang.Object
  |
  +--javax.bluetooth.UUID
```

Description

The `UUID` class defines universally unique identifiers. These 128-bit unsigned integers are guaranteed to be unique across all time and space. Accordingly, an instance of this class is immutable. The Bluetooth specification provides an algorithm describing how a 16-bit or 32-bit UUID could be promoted to a 128-bit UUID. Accordingly, this class provides an interface that assists applications in creating 16-bit, 32-bit, and 128-bit long UUIDs. The methods supported by this class allow equality testing of two UUID objects.

The Bluetooth Assigned Numbers document (http://www.bluetooth.org/assigned-numbers/sdp.htm) defines a large number of UUIDs for protocols and service classes. The table below provides a short list of the most common UUIDs defined in the Bluetooth Assigned Numbers document.

Name	Value	Size
Base UUID Value (Used in promoting 16-bit and 32-bit UUIDs to 128-bit UUIDs)	0x00000000000010008000000805F9B34FB	128-bit
SDP	0x0001	16-bit
RFCOMM	0x0003	16-bit
OBEX	0x0008	16-bit
HTTP	0x000C	16-bit
L2CAP	0x0100	16-bit
BNEP	0x000F	16-bit
Serial Port	0x1101	16-bit
ServiceDiscoveryServerServiceClassID	0x1000	16-bit
BrowseGroupDescriptorServiceClassID	0x1001	16-bit
PublicBrowseGroup	0x1002	16-bit
OBEX Object Push Profile	0x1105	16-bit
OBEX File Transfer Profile	0x1106	16-bit
Personal Area Networking User	0x1115	16-bit
Network Access Point	0x1116	16-bit
Group Network	0x1117	16-bit

Member Summary	
Constructors	
`public`	`UUID(long)` Creates a `UUID` object from `long` value `uuidValue`.
`public`	`UUID(String, boolean)` Creates a `UUID` object from the string provided.
Methods	
`public boolean`	`equals(Object)` Determines if two `UUID`s are equal.
`public int`	`hashCode()` Computes the hash code for this object.
`public String`	`toString()` Returns the string representation of the 128-bit UUID object.

Inherited Member Summary
Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, wait, wait, wait`

Constructors

UUID(long)

```
public UUID(long uuidValue)
```

Creates a `UUID` object from `long` value `uuidValue`. A UUID is defined as an unsigned integer whose value can range from [0 to 2^{128}-1]. However, this constructor allows only those values that are in the range of [0 to 2^{32} -1]. Negative values and values in the range of [2^{32}, 2^{63} -1] are not allowed and will cause an `IllegalArgumentException` to be thrown.

Parameters:

`uuidValue` - the 16-bit or 32-bit value of the UUID

Throws:

`IllegalArgumentException` - if `uuidValue` is not in the range [0, 2^{32} -1]

UUID(String, boolean)

```
public UUID(java.lang.String uuidValue, boolean shortUUID)
```

Creates a `UUID` object from the string provided. The characters in the string must be from the hexadecimal set [0-9, a-f, A-F]. It is important to note that the prefix "0x" generally used for hex representation of numbers is not allowed. If the string does not have characters from the hexadecimal set, an exception will be thrown. The string length has to be positive and less than or equal to 32. A string length that exceeds 32 is illegal and will cause an exception. Finally, a `null` input is also considered illegal and causes an exception.

If `shortUUID` is `true`, `uuidValue` represents a 16-bit or 32-bit UUID. If `uuidValue` is in the range 0x0000 to 0xFFFF then this constructor will create a 16-bit UUID. If `uuidValue` is in the range

0x000010000 to 0xFFFFFFFF, then this constructor will create a 32-bit UUID. Therefore, `uuidValue` may only be 8 characters long.

On the other hand, if `shortUUID` is `false`, then `uuidValue` represents a 128-bit UUID. Therefore, `uuidValue` may only be 32 character long

Parameters:

`uuidValue` - the string representation of a 16-bit, 32-bit or 128-bit UUID

`shortUUID` - indicates the size of the UUID to be constructed; `true` is used to indicate short UUIDs, i.e. either 16-bit or 32-bit; `false` indicates an 128-bit UUID

Throws:

`NumberFormatException` - if `uuidValue` has characters that are not defined in the hexadecimal set [0-9, a-f, A-F]

`IllegalArgumentException` - if `uuidValue` length is zero; if `shortUUID` is `true` and `uuidValue`'s length is greater than 8; if `shortUUID` is `false` and `uuidValue`'s length is greater than 32

`NullPointerException` - if `uuidValue` is `null`

Methods

equals(Object)

```
public boolean equals(java.lang.Object value)
```

Determines if two `UUID`s are equal. They are equal if their 128 bit values are the same. This method will return `false` if `value` is `null` or is not a `UUID` object.

Overrides: java.lang.Object.equals(java.lang.Object) in class java.lang.Object

Parameters:

`value` - the object to compare to

Returns: `true` if the 128 bit values of the two objects are equal, otherwise `false`

hashCode()

```
public int hashCode()
```

Computes the hash code for this object. This method retains the same semantic contract as defined in the class `java.lang.Object` while overriding the implementation.

Overrides: java.lang.Object.hashCode() in class java.lang.Object

Returns: the hash code for this object

toString()

```
public java.lang.String toString()
```

Returns the string representation of the 128-bit UUID object. The string being returned represents a UUID that contains characters from the hexadecimal set, [0-9, A-F]. It does not include the prefix "0x" that is generally used for hex representation of numbers. The return value will never be `null`.

Overrides: java.lang.Object.toString() in class java.lang.Object

Returns: the string representation of the UUID