Java^{TM} Telephony API (JTAPI)

Programmer’s Reference

(JTAPI v1.2 Specification)

Notice

Every effort was made to ensure that the information in this book was complete and accurate at the time of printing. However, information is subject to change.

Your Responsibility for Your System's Security

Toll fraud is the unauthorized use of your telecommunications system by an unauthorized party, for example, persons other than your company's employees, agents, subcontractors, or persons working on your company's behalf. Note that there may be a risk of toll fraud associated with your telecommunications system and, if toll fraud occurs, it can result in substantial additional charges for your telecommunications services. You and your system manager are responsible for the security of your system, such as programming and configuring your equipment to prevent unauthorized use. The system manager is also responsible for reading all installation, instruction, and system administration documents provided with this product in order to fully understand the features that can introduce risk of toll fraud and the steps that can be taken to reduce that risk. Avaya does not warrant that this product is immune from or will prevent unauthorized use of common-carrier telecommunication services or facilities accessed through or connected to it. Avaya Technologies will not be responsible for any charges that result from such unauthorized use.

Avaya Fraud Intervention

If you *suspect that you are being victimized* by toll fraud and you need technical support or assistance, call Technical Service Center

Toll Fraud Intervention Hotline at **1 800 643 235**3.

Trademarks

Acknowledgment

This document was prepared by Avaya University Information Development, Holmdel, NJ 07733-3030.

About This Document

This document consists of Sun Microsystem’s Java Telephony API specification files that have been downloaded from Sun Microsystem’s Java Telephony API web site (this document is the JTAPI v1.2 specification). As of this writing, Avaya is providing the latest available version. To obtain the very latest version of the JTAPI specification files, go directly to the web site at:

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

What is JTAPI?

The Java Telephony API (JTAPI) specifies the standard telephony application programming interface for computer-telephony applications under Java. It is the definition for a reusable set of call control objects that bring cross-platform and cross-implementation portability to telephony applications. It is a simple, extensible, object-oriented model that addresses a broad range of computer-telephony tasks. The Java Telephony API represents the combined efforts of design teams from Sun, Avaya, Nortel, Novell, Intel, and IBM, all operating under the direction of JavaSoft.

Purpose of this Document

As a JTAPI Specification, this document provides code-level descriptions of JTAPI libraries and classes . Use it as a reference to the telephony services oriented libraries and classes that you use to develop JTAPI applications. This document assumes that you have an implementation of the defined interfaces in the form of Java classes in order to produce a working product.

Intended Audience

This document is for application developers who are programming applications that use JTAPI. This document assumes a familiarity with the Java programming language.

Related Documents

Use this document along with the *Centre Vu Computer-Telephony Java Telephony API (JTAPI) Client Programmer’s Guide* (also referred to as the JTAPI Programmer's Guide) to develop Java based telephony applications. Refer to the Related Documents section of the JTAPI Programmer's Guide for a more extensive list of Computer- Telephony Integration (CTI) documents.

Package Index

Other Packages

- package javax.telephony
- package javax.telephony.callcenter
- package javax.telephony.callcenter.capabilities
- package javax.telephony.callcenter.events
- package javax.telephony.callcontrol
- package javax.telephony.callcontrol.capabilities
- package javax.telephony.callcontrol.events
- package javax.telephony.capabilities
- package javax.telephony.events
- package javax.telephony.media
- package javax.telephony.media.capabilities
- package javax.telephony.media.events
- package javax.telephony.phone
- package javax.telephony.phone.capabilities
- package javax.telephony.phone.events
- package javax.telephony.privatedata
- package javax.telephony.privatedata.capabilities
- package javax.telephony.privatedata.events

All Packages

package javax.telephony

Interface Index

- Address
- AddressObserver
- Call
- CallObserver
- Connection
- JtapiPeer
- Provider
- ProviderObserver
- Terminal
- TerminalConnection
- TerminalObserver

Class Index

- JtapiPeerFactory

Exception Index

- InvalidArgumentException
- InvalidPartyException
- InvalidStateException
- JtapiPeerUnavailableException
- MethodNotSupportedException
- PlatformException
- PrivilegeViolationException
- ProviderUnavailableException
- ResourceUnavailableException

Interface javax.telephony.Address

public interface **Address**

Introduction

An Address object represents what we commonly think of as a "telephone number." In implementations where the underlying network is not a telephone network, this address may represent something else. For example, if the underlying network is IP, this address might represent an IP address (e.g. 18.203.0.49). An Address object has a string *name* which corresponds to this telephone address. The Address object does not attempt to interpret this string in any way. This name is first assigned when the Address object is created and does not change throughout the lifetime of the object. The method `Address.getName()` returns the name of the Address object.

Address objects may be classified into two categories: *local* and *remote*. Local Address objects are those addresses which are part of the Provider's local domain. These Address objects are created by the implementation of the Provider object when it is first instantiated. All of the Provider's local addresses are reported via the `Provider.getAddresses()` method. Remote Address objects are those outside of the Provider's domain which the Provider learns about during its lifetime through various happenings (e.g. an incoming call from a currently unknown address). Remote Addresses are **not** reported via the `Provider.getAddresses()` method. Note that applications never explicitly create new Address objects.

Address and Terminal Objects

Address and Terminal objects exist in a many-to-many relationship. An Address object may have zero or more Terminals associated with it. Each Terminal associated with an Address must reflect its association with the Address. Since the implementation creates Address (and Terminal) objects, it is responsible for insuring the correctness of these relationships. The Terminals associated with an Address is given by the `Address.getTerminals()` method.

An association between an Address and Terminal indicates that the Terminal is addressable via that Address. In many instances, a telephone set (represented by a Terminal object) has only one telephone number (represented by an Address object) associated with it. In more complex configurations, telephone sets may have several telephone numbers associated with them. Likewise, a telephone number may appear on more than one telephone set. For example, feature phones in PBX environments may exhibit this configuration.

Address and Call Objects

Address objects represent the *logical* endpoints of a telephone call. A logical view of a telephone call views the call as originating from one Address endpoint and terminates at another Address endpoint.

Address objects are related to Call objects via the Connection object. The Connection object has a *state* which describes the current relationship between the Call and the Address. Each Address object may be part of more than one telephone call, and in each case, is represented by a separate Connection object. The `Address.getConnections()` method returns all Connection objects currently associated with the Call.

An Address is associated with a Call until the Connection moves into the `Connection.DISCONNECTED` state. At that time, the Connection is no longer reported via the `Address.getConnections()` method. Therefore, the `Address.getConnections()` method will never report a Connection in the `Connection.DISCONNECTED` state.

Existing Telephone Calls

The Java Telephony API specification states that the implementation is responsible for reporting all existing telephone calls when a Provider is first created. This implies that an Address object must report information regarding existing telephone calls to that Address. In other words, Address objects must reports all Connection objects which represent existing telephone calls.

Address Observers and Events

All changes in an Address object are reported via the AddressObserver interface. Applications instantiate an object which implements this interface and begins this delivery of events to this object using the `Address.addObserver()` method. All Address-related events extend the `AddrEv` interface provided in the core package. Applications receive events on an observer until the observer is removed via the `Address.removeObserver()` method or until the Address is no longer observable. In these instances, each AddressObserver receives a AddrObservationEndedEv as its final event.

Currently in the core package, the only Address-related event is AddrObservationEndedEv.

Call Observers

At times, applications may want to monitor a particular Address for all Calls which come to that Address. For example, a customer service agent application is only interested in telephone calls associated with a particular agent address. To achieve this sort of Address-based Call monitoring applications may *add* CallObservers to an Address via the `Address.addCallObserver()` method.

When a CallObserver is added to an Address, this observer instance is immediately added to all Calls at this Address and is added to all Calls which come to this Address in the future. These observers remain on the telephone call as long as the Address is associated with the telephone call.

The specification of `Address.addCallObserver()` contains more precise semantics.

See Also:

AddressObserver, CallObserver

Method Index

- **addCallObserver**(CallObserver)

  Adds an observer to a Call object when this Address object first becomes part of that Call.

- **addObserver**(AddressObserver)

  Adds an observer to the Address.

- **getAddressCapabilities**(Terminal)

  Gets the AddressCapabilities object with respect to a Terminal If null is passed as a Terminal parameter, the general/provider-wide Address capabilities are returned. **Deprecated.**

- **getCallObservers**()

  Returns a list of all CallObservers associated with this Address object.

- **getCapabilities**()

  Returns the dynamic capabilities for this instance of the Address object.

- **getConnections**()

  Returns an array of Connection objects currently associated with this Address object at the instant the `getConnections()` method is called.

- **getName**()

  Returns the name of the Address.

- **getObservers**()

  Returns a list of all AddressObservers associated with this Address object.

- **getProvider**()

  Returns the Provider associated with this Address.

- **getTerminals**()

  Returns an array of Terminals associated with this Address object.

- **removeCallObserver**(CallObserver)

  Removes the given CallObserver from the Address.

- **removeObserver**(AddressObserver)

  Removes the given observer from the Address.

Methods

getName

```
public abstract String getName()
```

Returns the name of the Address. Each Address possesses a unique name. This name is a way of referencing an endpoint in a telephone call.

Returns:

The name of this Address.

getProvider

```
public abstract Provider getProvider()
```

Returns the Provider associated with this Address. This Provider object is valid throughout the lifetime of the Address and does not change once the Address is created.

Returns:

The Provider associated with this Address.

getTerminals

```
public abstract Terminal[] getTerminals()
```

Returns an array of Terminals associated with this Address object. If no Terminals are associated with this Address, this method returns null. This list does not change throughout the lifetime of the Address object.

Returns:

An array of Terminal objects associated with this Address.

getConnections

```
public abstract Connection[] getConnections()
```

Returns an array of Connection objects currently associated with this Address object at the instant the `getConnections()` method is called. When a Connection moves into the `Connection.DISCONNECTED` state, the Address object loses the reference to the Connection and the Connection no longer returned by this method. Therefore, all Connections returned by this method will never be in the `Connection.DISCONNECTED` state. If the Address has no Connections associated with it, this method returns null.

Post-conditions:

1. Let Connection c[] = this.getConnections()
2. c == null or c.length >= 1
3. For all i, c[i].getState() != Connection.DISCONNECTED

Returns:

An array of Connection objects associated with this Address.

addObserver

```
public abstract void addObserver(AddressObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to the Address. The AddressObserver reports all Address- related state changes as events. The Address object will report events to this AddressObserver object for the lifetime of the Address object or until the observer is removed with the `Address.removeObserver()` method or until the Address is no longer observable. In these instances, a AddrObservationEndedEv is delivered to the observer as its final event. The observer will receive no events after AddrObservationEndedEv unless the observer is explicitly re-added via the `Address.addObserver()` method. Also, once an observer receives an AddrObservationEndedEv, it will no longer be reported via the `Address.getObservers()`.

If an application attempts to add an instance of an observer already present on this Address, this attempt will silently fail, i.e. multiple instances of an observer are not added and no exception will be thrown.

Currently, only the AddrObservationEndedEv event is defined by the core package and delivered to the AddressObserver.

Post-conditions:

1. observer is an element of this.getObservers()

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time

Throws: ResourceUnavailableException

The resource limit for the number of observers has been exceeded.

getObservers

```
 public abstract AddressObserver[] getObservers()
```

Returns a list of all AddressObservers associated with this Address object. If there are no observers associated with this Address object, this method returns null.

Post-conditions:

1. Let AddressObserver[] obs = this.getObservers()
2. obs == null or obs.length >= 1

Returns:

An array of AddressObserver objects associated with this Address.

removeObserver

```
 public abstract void removeObserver(AddressObserver observer)
```

Removes the given observer from the Address. If successful, the observer will no longer receive events generated by this Address object. As its final event, the AddressObserver receives is an AddrObservationEndedEv event.

If an observer is not part of the Address, then this method fails silently, i.e. no observer is removed and no exception is thrown.

Post-conditions:

1. An AddrObservationEndedEv event is reported to the observer as its final event.
2. observer is not an element of this.getObservers()

Parameters:

observer - The observer which is being removed.

addCallObserver

```
 public abstract void addCallObserver(CallObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to a Call object when this Address object first becomes part of that Call. This method permits applications to select an Address object in which they are interested and automatically have the implementation attach an observer to all present and future Calls which come to this Address.

JTAPI v1.0 Semantics

In version 1.0 of the Java Telephony API specification, the application monitored the Address object for the AddrCallAtAddrEv event. This event indicated that a Call has come to this Address. Then, applications would manually add an observer to the Call. With this architecture, potentially dangerous race conditions existed while an application was adding an observer to the Call. As a result, this mechanism has been replaced in version 1.1.

JTAPI v1.1 Semantics

In version 1.1 of the specification, the AddrCallAtAddrEv event does not exist and this method replaces the functionality described above. Instead of monitoring for a AddrCallAtAddrEv event, this application simply uses the `Address.addCallObserver()` method, and observer will be added to new telephone calls at this Address automatically.

If an application attempts to add an instance of a call observer already present on this Address, these repeated attempts will silently fail, i.e. multiple instances of a call observer are not added and no exception will be thrown.

When a call observer is added to an Address with this method, the following behavior is exhibited by the implementation.

1. It is immediately associated with any existing calls at the Address and a snapshot of those calls are reported to the call observer. For each of these calls, the observer is reported via `Call.getObservers()`.
2. It is associated with all future calls which comes to this Address. For each new call, the observer is reported via `Call.getObservers().`

Note that the definition of the term ".. when a call is at an Address" means that the Call contains one Connection which has this Address as its Address.

Call Observer Lifetime

For all call observers which are present on Calls because of this method, the following behavior is exhibited with respect to the lifetime of the call.

1. The call observer receives events until the Call is no longer at this Address. In this case, the call observer will be re-applied to the Call if the Call returns to this Address at some point in the future.
2. The call observer is removed with `Call.removeObserver()`. Note that this only affects the instance of the call observer for that particular call. If the Call subsequently leaves and returns to the Address, the observer will be re-applied.
3. The Call can no longer be monitored by the implementation.
4. The Call moves into the `Call.INVALID` state.

When the CallObserver leaves the Call because of one of the reasons above, it receives a CallObservationEndedEv as its final event.

Call Observer on Multiple Addresses and Terminals

It is possible for an application to add CallObservers at more than one Address and Terminal (using `Address.addCallObserver()` and `Terminal.addCallObserver()`, respectively). The rules outlined above still apply, with the following additions:

1. A CallObserver is not added to a Call more than once, even if it has been added to more than one Address/Terminal which are present on the Call.
2. The CallObserver leaves the call only if *all* of the Addresses and Terminals on which the Call Observer was added leave the Call. If one of those Addresses/Terminals becomes part of the Call again, the call observer is re-applied to the Call.

Post-Conditions:

1. observer is an element of this.getCallObservers()
2. observer is an element of Call.getObservers() for each Call associated with the Connections from this.getConnections().
3. An array of snapshot events are reported to the observer for existing calls associated with this Address.

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time.

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

See Also:

Call

getCallObservers

```
public abstract CallObserver[] getCallObservers()
```

Returns a list of all CallObservers associated with this Address object. That is, it returns a list of CallObserver object which have been added via the `Address.addCallObserver()` method. If there are no CallObservers associated with this Address object, this method returns null.

Post-conditions:

1. Let CallObserver[] obs = this.getCallObservers()
2. obs == null or obs.length >= 1

Returns:

An array of CallObserver objects associated with this Address.

● removeCallObserver

```
public abstract void removeCallObserver(CallObserver observer)
```

Removes the given CallObserver from the Address. In other words, it removes a CallObserver which was added via the `Address.addCallObserver()` method. If successful, the observer will no longer be added to new Calls which are presented to this Address, however it does not affect CallObservers which have already been added at a Call.

Also, if the CallObserver is not part of the Address, then this method fails silently, i.e. no observer is removed and no exception is thrown.

Post-conditions:

1. observer is not an element of this.getCallObservers()

Parameters:

observer - The CallObserver which is being removed.

● getCapabilities

```
public abstract AddressCapabilities getCapabilities()
```

Returns the dynamic capabilities for this instance of the Address object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation-specific knowledge. These indications do not guarantee that a particular method will succeed when invoked, however.

The dynamic Address capabilities require no additional arguments.

Returns:

The dynamic Address capabilities.

● getAddressCapabilities

```
public abstract AddressCapabilities getAddressCapabilities(Terminal terminal)
throws InvalidArgumentException, PlatformException
```

Note: getAddressCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Address.getCapabilities() method.*

Gets the AddressCapabilities object with respect to a Terminal If null is passed as a Terminal parameter, the general/provider-wide Address capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Address.getCapabilities()` method returns the dynamic Address capabilities. This method now should simply invoke the `Address.getCapabilities()` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

Interface javax.telephony.AddressObserver

public interface **AddressObserver**

Introduction

The `AddressObserver` interface reports all changes which happen to the Address object. These changes are reported as events to the `AddressObserver.addressChangedEvent()` method. Applications must instantiate an object which implements this interface and then use the `Address.addObserver()`) method to register the object to receive all future events associated with the Address object.

The `AddressObserver.addressChangedEvent()` method receives an array of events which all must extend the `AddrEv` interface. Since several changes may happen to a single JTAPI object at once, a list of events is needed to convey those changes which happen at the same time. Applications iterate through the array of events provided.

Address Observation Ending

At various times, the underlying implementation may not be able to observe the Address. In these instances, the AddressObserver is sent an AddrObservationEndedEv event. This indicates that the application will not receive further events associated with the Address object. The observer is no longer reported via the `Address.getObservers()` method.

See Also:

AddrEv, AddrObservationEndedEv

Method Index

addressChangedEvent(AddrEv[])

Reports all events associated with the Address object.

Methods

addressChangedEvent

```
public abstract void addressChangedEvent(AddrEv eventList[])
```

Reports all events associated with the Address object. This method passes an array of AddrEv objects as its arguments which correspond to the list of events representing the changes to the Address object.

Parameters:

eventList - The list of Address events.

Interface javax.telephony.Call

public interface **Call**

Introduction

A Call object models a telephone call. A Call can have zero or more Connections. A two-party call has two Connections, and a conference call has three or more Connections. Each Connection models the relationship between a Call and an Address, where an Address identifies a particular party or set of parties on a Call.

Creating Call Objects

Applications create instances of a Call object with the `Provider.createCall()` method, which returns a Call object that has zero Connections and is in the `Call.IDLE state`. The Call maintains a reference to its Provider for the life of that Call object. This Provider object instance does not change throughout the lifetime of the Call object. The Provider associated with a Call is obtained via the `Call.getProvider()` method.

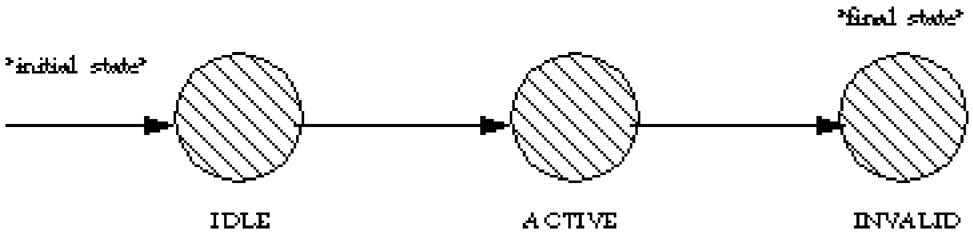
Call States

A Call has a *state* which is obtained via the `Call.getState()` method. This state describes the current progress of a telephone call, where is it in its life cycle, and how many Connections exist on the Call. The Call state may be one of three values: `Call.IDLE`, `Call.ACTIVE`, or `Call.INVALID`. The following is a description of each state:

`Call.IDLE`	This is the initial state for all Calls. In this state, the Call has zero Connections, that is `Call.getConnections()` *must* return null.
`Call.ACTIVE`	A Call with some current ongoing activity is in this state. Calls with one or more associated Connections must be in this state. If a Call is in this state, the `Call.getConnections()` method must return an array of size at least one.
`Call.INVALID`	This is the final state for all Calls. Call objects which lose all of their Connections objects (via a transition of the Connection object into the `Connection.DISCONNECTED` state) moves into this state. Calls in this state have zero Connections and these Call objects may not be used for any future action. In this state, the `Call.getConnections()` *must* return null.

Call State Transitions

The possible Call state transitions are given in the diagram below:



Calls and Connections

A Call maintain a list of the Connections on that Call. Applications obtain an array of Connections associated with the Call via the `Call.getConnections()` method. A Call retains a reference to a Connection only if it is not in the `Connection.DISCONNECTED` state. Therefore, if a Call has a reference to a Connection, then that Connection must not be in the `Connection.DISCONNECTED` state. When a Connection moves into the `Connection.DISCONNECTED` state (e.g. when a party hangs up), the Call loses its reference to that Connection which is no longer reported via the `Call.getConnections()` method.

The `Call.connect()` method

The primary method on this interface is the `Call.connect()` method. Applications use this method to place a telephone call from an originating endpoint to a destination address string. The result of this method on the call model is to create an originating and destination Connection and move the Call into the `Call.ACTIVE`. As the new telephone call progresses during its lifetime, the state of various objects associated with the Call may change and new objects may be created and associated with the Call. See the specification of the `Call.connect()` method below for more details.

Observers and Events

The `CallObserver` interface reports all events pertaining to the Call object. Events delivered to this interface must extend the `CallEv` interface. Applications add observers to a Call object via the `Call.addObserver()` method.

Connection-related and TerminalConnection-related events are also reported via the `CallObserver` interface. These events include the creation of these objects and their state changes. Events which are reported via the `CallObserver` interface pertaining to Connections and TerminalConnections extend the `ConnEv` interface and the `TermConnEv` interface, respectively.

An event is delivered to the application whenever the state of the Call changes. The event interfaces corresponding to Call state changes are: `CallActiveEv` and `CallInvalidEv`.

At times the Call may be unobservable by the implementation. In this case, a CallObservationEndedEv is delivered to the `CallObserver` interface. This is the final event receives by the observer and is no longer reported via the `Call.getObservers()` method.

Applications may observe a Call by adding an observer via the Address or Terminal objects using the `Address.addCallObserver()` and `Terminal.addCallObserver()` methods. These methods provide the ability for an application to receive Call-related events when a Call contains a particular Address or Terminal. See the specifications for Address and Terminal for more details.

See Also:

CallObserver, Connection, Address, Terminal, TerminalConnection, CallEv

---

Variable Index

- **ACTIVE**

  The `Call.ACTIVE` state indicates the Call has one or more Connections, none of which are in the `Connection.DISCONNECTED` state.

- **IDLE**

  The `Call.IDLE` state indicates the Call has zero Connections.

- **INVALID**

  The `Call.INVALID` state indicates the Call has lost all of its connections, ie.

Method Index

- **addObserver**(CallObserver)

  Adds an observer to the Call.

- **connect**(Terminal, Address, String)

  Places a telephone call from an originating endpoint to a destination address string.

- **getCallCapabilities**(Terminal, Address)

  Gets the CallCapabilities object with respect to a Terminal and an Address. **Deprecated.**

- **getCapabilities**(Terminal, Address)

Returns the dynamic capabilities for the instance of the Call object.

- **getConnections**()

  Returns an array of Connections associated with this call.
- **getObservers**()

  Returns an array of all `CallObservers` on this Call.
- **getProvider**()

  Returns the Provider associated with the Call.
- **getState**()

  Returns the current state of the telephone call.
- **removeObserver**(CallObserver)

  Removes the given observer from the Call.

Variables

IDLE

```
public static final int IDLE
```

The `Call.IDLE` state indicates the Call has zero Connections. It is the initial state of all Call objects.

ACTIVE

```
public static final int ACTIVE
```

The `Call.ACTIVE` state indicates the Call has one or more Connections, none of which are in the `Connection.DISCONNECTED` state.

INVALID

```
public static final int INVALID
```

The `Call.INVALID` state indicates the Call has lost all of its connections, ie. all of its associated Connection objects have moved into the `Connection.DISCONNECTED` state and are no longer associated with the Call. A Call in this state cannot be used for future actions.

Methods

getConnections

```
public abstract Connection[] getConnections()
```

Returns an array of Connections associated with this call. Note that none of the Connections returned will be in the `Connection.DISCONNECTED` state. Also, if the Call is in the `Call.IDLE` state or the `Call.INVALID` state, this method returns null. Otherwise, it returns one or more Connection objects.

Post-conditions:

1. Let Connection[] conn = Call.getConnections()
2. if this.getState() == Call.IDLE then conn = null
3. if this.getState() == Call.INVALID then conn = null
4. if this.getState() == Call.ACTIVE then conn.length >= 1
5. For all i, conn[i].getState() != Connection.DISCONNECTED

Returns:

An array of the Connections associated with the call.

getProvider

```
public abstract Provider getProvider()
```

Returns the Provider associated with the Call. This Provider reference remains valid throughout the lifetime of the Call object, despite the state of the Call object. This Provider reference does not change once the Call object has been created.

Returns:

The Provider associated with the call.

getState

```
public abstract int getState()
```

Returns the current state of the telephone call. The state will be either `Call.IDLE`, `Call.ACTIVE`, or `Call.INVALID`.

Returns:

The current state of the call.

connect

```
public abstract Connection[] connect(Terminal origterm,
                                     Address origaddr,
                                     String dialedDigits) throws
ResourceUnavailableException, PrivilegeViolationException, InvalidPartyException,
InvalidArgumentException, InvalidStateException, MethodNotSupportedException
```

Places a telephone call from an originating endpoint to a destination address string.

The Call must be in the `Call.IDLE` state (and therefore have no existing associated Connections and the Provider must be in the `Provider.IN_SERVICE` state. The successful effect of this method is to place the telephone call and create and return two Connections associated with this Call.

Method Arguments

This method has three arguments. The first argument is the originating Terminal for the telephone call. The second argument is the originating Address for the telephone Call. This Terminal/Address pair must reference one another. That is, the originating Address must appear on the Terminal (via `Address.getTerminals()`Terminal.getAddress()). If not, an InvalidArgumentException is thrown.

The third argument is a destination string whose value represents the address to which the telephone call is placed. This destination address must be valid and complete.

Method Post-conditions

This method returns successfully when the Provider can successfully initiate the placing of the telephone call. As a result, when the `Call.connect()` method returns, the Call will be in the `Call.ACTIVE` state and exactly two Connections will be created and returned. The Connection associated with the originating endpoint is the first Connection in the returned array, and the Connection associated with the destination endpoint is the second Connection in the returned array. These two Connections must at least be in the `Connection.IDLE` state. That is, if one of the Connections progresses beyond the `Connection.IDLE` state while this method is completing, this Connection may be in a state other than the `Connection.IDLE`. This state must be reflected by an event sent to the application.

Call Scenarios or Flows

As a telephone call progresses during its lifetime, the various objects associated with the Call may change state and new objects may be created an associated with the Call. These changes typically occur after this method has successfully returned.

How the Call and its associated objects progress depends upon real-world conditions. There is a large but finite number of ways in which the state of a Call may progress. It is difficult to enumerate each possible way in which the state of a Call may progress. Instead, several illustrative *scenarios* (also called *flows*) are presented for common real-world conditions. These scenarios obey the valid state transitions as defined by the Call, Connection, and TerminalConnection objects.

Two common scenarios are presented below. Note that there may exist additional scenarios very similar to these which may only differ in a single step or state change. Any implementation which adheres to the rules outlined by the Call, Connection, and TerminalConnection objects is considered a proper implementations with respect to call flows.

Normal `Call.connect()` Scenario

This this scenario, a telephone call is placed to a telephone number address. This address is outside the Provider's domain. The destination party answers the telephone call.

1. The `Call.connect()` method is invoked with the given arguments. Two Connection objects are created and returned, each in the `Connection.IDLE` state.

   Events delivered to the application: a CallActivEv and two ConnCreatedEv, one for each Connection.

2. Once the Provider begins to place the telephone call, the originating Connection moves from the `Connection.IDLE` state into the `Connection.CONNECTED` state. A TerminalConnection is created in the `TerminalConnection.IDLE` state and moves into the to model the `TerminalConnection.ACTIVE` relationship between the originating Terminal and the Connection.

   Events delivered to the application: a ConnConnectedEv for the originating Connection, a TermConnCreatedEv and a TermConnActiveEv for the new TerminalConnection.

   Note: Depending upon the configuration of the switch, additional TerminalConnection objects associated with the originating Connection may be created. If the originating Address has more than on Terminal, these additional Terminals may be involved in the telephone call. For each TerminalConnection created a TermConnCreatedEv is delivered. Typically, these TerminalConnections will be in the `TerminalConnection.PASSIVE` state and a TermConnPassiveEv is delivered for each.

3. The destination Connection into the `Connection.INPROGRESS` state as the Call proceeds.

   Events delivered to the application: a ConnInProgressEv for the destination Connection.

4. The destination Connection moves into the `Connection.ALERTING` state as the destination is alerted to the telephone call. TerminalConnection object may be created to model the relationship between any known destination Terminals associated with the Call, each in the `TerminalConnection.RINGING` state. If the destination Terminals are unknown, then no TerminalConnections are created.

   Events delivered to the application: a TermConnAlerting for the destination Connection, a TermConnCreatedEv and TermConnRingingEv for any destination TerminalConnections created.

5. The destination Connection moves into the `Connection.CONNECTED` state when the called party answers the telephone call. If the destination Terminals are known, the answering TerminalConnection moves into the `TerminalConnection.ACTIVE` state.

   Events delivered to the application: a ConnConnectedEv for the destination Connection and a TermConnActiveEv for the answering TerminalConnection, if known.

   Note: For all other non-answering destination TerminalConnections known, either one of two state changes will occur depending upon the configuration of the switch. These TerminalConnections will either move into the `TerminalConnection.PASSIVE` state or the `TerminalConnection.DROPPED` state, depending upon whether or not these Terminals continue as part of the telephone call. For each, either a TermConnPassiveEv or TermConnDroppedEv is delivered.

At the conclusion of this scenario, the Call will be in the `Call.ACTIVE` state, both Connections will be in the `Connection.CONNECTED` state, and the originating TerminalConnection will be in the `TerminalConnection.ACTIVE` state.

Failure `Call.connect()` Scenario

In this scenario, a telephone call is placed to a destination address. The destination cannot be reached, perhaps because the destination address is busy.

The first three steps of the first scenario are the same as in this scenario. They are not repeated here for brevity. The fourth and final step of this scenario is:

1. The destination Connection moves into the `Connection.FAILED` because the destination party could not be reached. (e.g. busy)

   Events delivered to the application: a ConnFailedEv is delivered for the destination Connection.

At the conclusion of this scenario, the Call will be in the `Call.ACTIVE` state, the originating Connection will be in the `Connection.CONNECTED` state, the destination Connection will be in the `Connection.FAILED` state, and the originating TerminalConnection will be in the `TerminalConnection.ACTIVE` state.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. Let Connection c[] = this.getConnections()
4. c.length == 2
5. c[0].getState() == Connection.IDLE (at least)
6. c[1].getState() == Connection.IDLE (at least)

Parameters:

origterm - The originating Terminal for this telephone call.

origaddr - The originating Address for this telephone call.

destaddr - The destination address string for this telephone call.

Throws: ResourceUnavailableException

An internal resource necessary for placing the phone call is unavailable.

Throws: PrivilegeViolationException

The application does not have the proper authority to place a telephone call.

Throws: InvalidPartyException

Either the originator or the destination does not represent a valid party required to place a telephone call.

Throws: InvalidArgumentException

An argument provided is not valid either by not providing enough information for this method or is inconsistent with another argument.

Throws: InvalidStateException

Some object required by this method is not in a valid state as designated by the pre-conditions for this method.

Throws: MethodNotSupportedException

The implementation does not support this method.

See Also:

CallActiveEv, ConnAlertingEv, ConnConnectedEv, ConnCreatedEv, ConnInProgressEv, TermConnActiveEv, TermConnCreatedEv, TermConnDroppedEv, TermConnPassiveEv, TermConnRingingEv

addObserver

```
 public abstract void addObserver(CallObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to the Call. The `CallObserver` reports all Call-related events. This includes changes in the state of the Call and all Connection-related and TerminalConnection-related events. The observer added with this method will report events on the call for as long as the implementation can observer the Call. In the case that the implementation can no longer observe the Call, the applications receives a CallObservationEndedEv. The observer receives no more events after it receives the CallObservationEndedEv and is no longer reported via the `Call.getObservers()` method.

Observer Lifetime

The `CallObserver` will receive events until one of the following occurs, whereupon the observer receives a CallObservationEndedEv and the observer is no longer reported via the `Call.getObservers()` method.

1. The observer is removed by the application with `Call.removeObserver()`.

2. The implementation can no longer monitor the call.
3. The Call has completed and moved into the `Call.INVALID` state.

Event Snapshots

By default, when an observer is added to a telephone call, the first batch of events may be a "snapshot". That is, if the observer was added after state changes in the Call, the first batch of events will inform the application of the current state of the Call. Note that these snapshot events do NOT provide a history of all events on the Call, rather they provide the minimum necessary information to bring the application up-to-date with the current state of the Call. The meta code for all of these events will be `Ev.META_SNAPSHOT`.

CallObservers from Addresses and Terminals

There may be additional call observers on the call which were not added by this method. These observers may have become part of the call via the `Address.addCallObserver()` and `Terminal.addCallObserver()` methods. See the specifications for these methods for more information.

Multiple Invocations

If an application attempts to add an instance of an observer already present on this Call, there are two possible outcomes:

1. If the observer was added by the application using this method, then a repeated invocation will silently fail, i.e. multiple instances of an observer are not added and no exception will be thrown.
2. If the observer is part of the call because an application invoked `Address.addCallObserver()` or `Terminal.addCallObserver()`, either of these methods modifies the behavior of that observer as if it were added via this method instead. That is, the observer is no longer removed when the call leaves the Address or Terminal. The observer now receives events until one of the conditions in "Observer Lifetime" is met.

Post-Conditions:

1. observer is an element of `this.getObservers()`
2. A snapshot of events is delivered to the observer, if appropriate.

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

getObservers

```
public abstract CallObserver[] getObservers()
```

Returns an array of all `CallObservers` on this Call. If no observers are on this Call object, then this method returns null. This method returns all observers on this call no matter how they were added to the Call. Call observers may be added to this call in one of three ways:

1. The application added the observer via `Call.addObserver()`.
2. The application added the observer via `Address.addCallObserver()` and the call came to that Address.
3. The application added the observer via `Terminal.addCallObserver()` and the call came to that Terminal.

An instance of a CallObserver object will only appear once in this list.

Post-Conditions:

1. Let CallObserver[] obs = this.getObservers()
2. obs == null or obs.length >= 1

Returns:

An array of CallObserver objects associated with this Call.

removeObserver

```
public abstract void removeObserver(CallObserver observer)
```

Removes the given observer from the Call. If successful, the observer will receive a CallObservationEndedEv as the last event it receives and will no longer be reported via the `Call.getObservers()` method.

This method has different effects depending upon how the observer was added to the Call, as follows:

1. If the observer was added via `Call.addObserver()`, this method removes the observer until it is re-applied by the application.
2. If the observer was added via `Address.addCallObserver()` or `Terminal.addCallObserver()`, this method removes the observer for this call only. It does not affect whether this observer will be added to future calls which come to that Address. See `Address.addCallObserver()` and `Terminal.addCallObserver()` for more details.

If an observer is not part of the Call, then this method fails silently, i.e. no observer is removed and no exception is thrown.

Post-Conditions:

1. observer is not an element of this.getObservers()
2. CallObservationEndedEv is delivered to the application

Parameters:

observer - The observer which is being removed.

getCapabilities

```
 public abstract CallCapabilities getCapabilities(Terminal terminal,
                                                  Address address) throws
InvalidArgumentException
```

Returns the dynamic capabilities for the instance of the Call object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation-specific knowledge. These indications do not guarantee that a particular method can be successfully invoked, however.

The dynamic call capabilities are based upon a Terminal/Address pair as well as the instance of the Call object. These parameters are used to determine whether certain call actions are possible at the present. For example, the `CallCapabilities.canConnect()` method will indicate whether a telephone call can be placed using the Terminal/Address pair as the originating endpoint.

Parameters:

terminal - Dynamic capabilities are with respect to this Terminal.

address - Dynamic capabilities are with respect to this Address.

Returns:

The dynamic Call capabilities.

Throws: InvalidArgumentException

Indicates the Terminal and/or Address argument provided was not valid.

getCallCapabilities

```
 public abstract CallCapabilities getCallCapabilities(Terminal term,
                                                      Address addr) throws
InvalidArgumentException, PlatformException
```

Note: getCallCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Call.getCapabilities() method.*

Gets the CallCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal parameter, the general/provider-wide Call capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Call.getCapabilities()` method returns the dynamic Call capabilities. This method now should simply invoke the `Call.getCapabilities()` method with the given Terminal and Address arguments.

Parameters:

term - Dynamic Call capabilities in JTAPI v1.2 are with respect to this Terminal.

addr - Dynamic Call capabilities in JTAPI v1.2 are with respect to this Address.

Throws: InvalidArgumentException

Indicates the Terminal and/or Address argument provided was not valid.

Throws: PlatformException

A platform-specific exception occurred.

Interface javax.telephony.CallObserver

public interface **CallObserver**

Introduction

The `CallObserver` interface reports all changes which happen to the Call object and all of the Connection and TerminalConnection objects which are part of the Call. These changes are reported as events to the `CallObserver.callChangedEvent()` method. Applications must instantiate an object which implements this interface and then add the observer to the call using one of several mechanisms described below to receive all future events associated with the Call and its Connections and TerminalConnections.

The `CallObserver.callChangedEvent()` method receives an array of events which all must extend the `CallEv` interface. Since several changes may happen to a single JTAPI object at once, a list of events is needed to convey those changes which happen at the same time. Applications iterate through the array of events provided.

Adding an Observer to a Call

Applications may add an observer to a Call via one of several mechanisms. They may directly add an observer via the `Call.addObserver()` method. Applications may also add observers to Calls indirectly via the `Address.addCallObserver()` and `Terminal.addCallObserver()` methods. These methods add the given observer to the Call when the Call comes to the Address or Terminal. See the specifications for Call, Address, and Terminal for more information.

Call State Changes

In the core package, an event is delivered whenever the Call changes state. The event interfaces which correspond to these state changes for the core package are: `CallActiveEv` and `CallInvalidEv`.

Connection Events

All events pertaining to the Connection object are reported on this interface. Connection events extend the `ConnEv` event, which in turn, extends the `CallEv` event. In the core package, an event is delivered to this interface whenever the Connection changes state.

TerminalConnection Events

All events pertaining to the TerminalConnection object are reported on this interface. TerminalConnection events extend the `TermConnEv` interface, which in turn, extends the `CallEv` interface. In the core package, an event is delivered to this interface whenever the TerminalConnection changes state.

Call Observation Ending

At various times, the underlying implementation may not be able to observe the Call. In these instances, the CallObserver is sent an CallObservationEndedEv event. This indicates that the application will not receive further events associated with the Call object. This observer is no longer reported via the `Call.getObservers()` method.

See Also:

CallEv, ConnEv, TermConnEv, CallObservationEndedEv, CallActiveEv, CallInvalidEv, ConnAlertingEv, ConnConnectedEv, ConnCreatedEv, ConnDisconnectedEv, ConnFailedEv, ConnInProgressEv, ConnUnknownEv, TermConnActiveEv, TermConnCreatedEv, TermConnDroppedEv, TermConnPassiveEv, TermConnRingingEv, TermConnUnknownEv

Method Index

- **callChangedEvent**(CallEv[])

  Reports all events associated with the Call object.

Methods

callChangedEvent

```
public abstract void callChangedEvent(CallEv eventList[])
```

Reports all events associated with the Call object. This method passes an array of CallEv objects as its arguments which correspond to the list of events representing the changes to the Call object as well as changes to all of the Connection and TerminalConnection objects associated with this Call.

Parameters:

eventList - The list of Call events.

All Packages This Package Previous Next

Interface javax.telephony.Connection

public interface **Connection**

Introduction

A Connection represents a link (i.e. an association) between a Call object and an Address object. The purpose of a Connection object is to describe the relationship between a Call object and an Address object. A Connection object exists if the Address is a part of the telephone call. Each Connection has a *state* which describes the particular stage of the relationship between the Call and Address. These states and their meanings are described below. Applications use the `Connection.getCall()` and `Connection.getAddress()` methods to obtain the Call and Address associated with this Connection, respectively.

From one perspective, an application may view a Call only in terms of the Address/Connection objects which are part of the Call. This is termed a *logical* view of the Call because it ignores the details provided by the Terminal and TerminalConnection objects which are also associated with a Call. In many instances, simple applications (such as an *outcall* program) may only need to concern itself with the logical view. In this logical view, a telephone call is views as two or more endpoint addresses in communication. The Connection object describes the state of each of these endpoint addresses with respect to the Call.

Calls and Addresses

Connection objects are immutable in terms of their Call and Address references. In other words, the Call and Address object references do not change throughout the lifetime of the Connection object instance. The same Connection object may not be used in another telephone call. The existence of a Connection implies that its Address is associated with its Call in the manner described by the Connection's state.

Although a Connection's Address and Call references remain valid throughout the lifetime of the Connection object, the same is not true for the Call and Address object's references to this Connection. Particularly, when a Connection moves into the Connection.DISCONNECTED state, it is no longer listed by the `Call.getConnections()` and `Address.getConnections()` methods. Typically, when a Connection moves into the `Connection.DISCONNECTED` state, the application loses its references to it to facilitate its garbage collection.

TerminalConnections

Connections objects are containers for zero or more TerminalConnection objects. Connection objects are containers for zero or more TerminalConnection objects. Connection objects represent the relationship between the Call and the Address, whereas TerminalConnection objects represent the relationship between the Connection and the Terminal. The relationship between the Call and the Address may be viewed as a logical view of the Call. The relationship between a Connection and a Terminal represents the physical view of the Call, i.e. at which Terminal the telephone calls terminates. The TerminalConnection object specification provides further information.

Connection States

Below is a description of each Connection state in real-world terms. These real-world descriptions have no bearing on the specifications of methods, they only serve to provide a more intuitive understanding of what is going on. Several methods in this specification state pre-conditions based upon the state of the Connection.

`Connection.IDLE`	This state is the initial state for all new Connections. Connections which are in the `Connection.IDLE` state are not actively part of a telephone call, yet their references to the Call and Address objects are valid. Connections typically do not stay in the `Connection.IDLE` state for long, quickly transitioning to other states.
`Connection.DISCONNECTED`	This state implies it is no longer part of the telephone call, although its references to Call and Address still remain valid. A Connection in this state is interpreted as once previously belonging to this telephone call.
`Connection.INPROGRESS`	This state implies that the Connection, which represents the destination end of a telephone call, is in the process of contacting the destination side. Under certain circumstances, the Connection may not progress beyond this state. Extension packages elaborate further on this state in various situations.
`Connection.ALERTING`	This state implies that the Address is being notified of an incoming call.
`Connection.CONNECTED`	This state implies that a Connection and its Address is actively part of a telephone call. In common terms, two people talking to one another are represented by two Connections in the `Connection.CONNECTED` state.

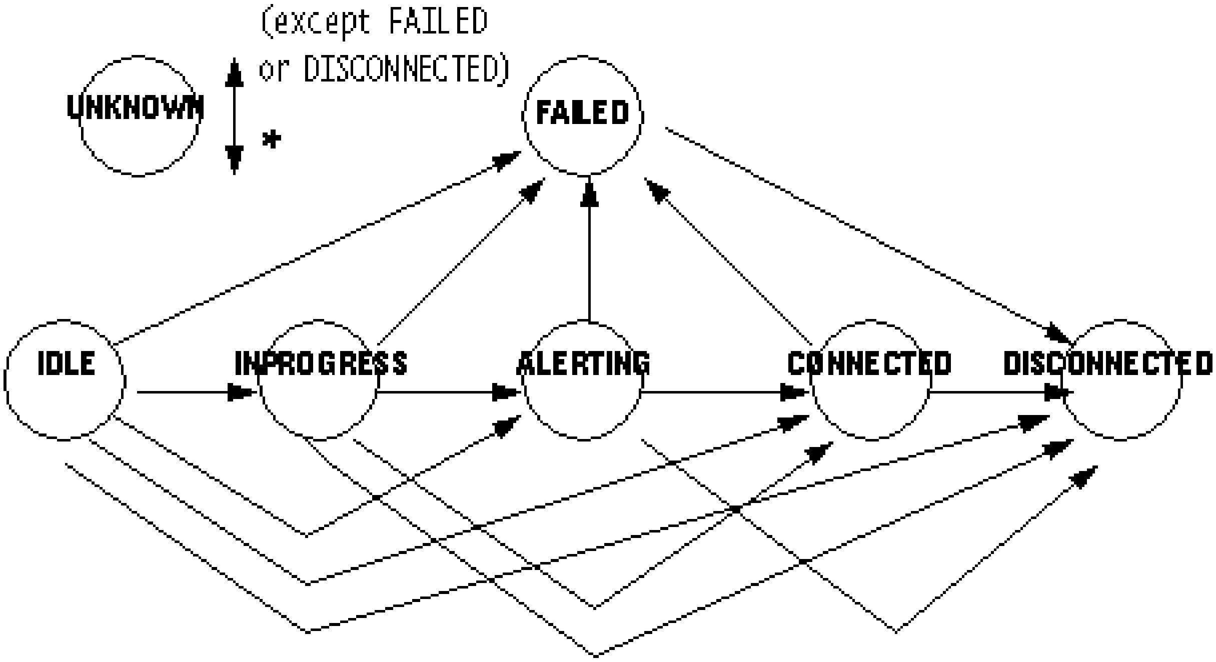
`Connection.UNKNOWN` This state implies that the implementation is unable to determine the current state of the Connection. Typically, methods are invalid on Connections which are in this state. Connections may move in and out of the `Connection.UNKNOWN` state at any time.

`Connection.FAILED` This state indicates that a Connection to that end of the call has failed for some reason. One reason why a Connection would be in the `Connection.FAILED` state is because the party was busy.

Connection State Transitions

With these loose, real-world meanings in the back of one's mind, the Connection class defines a finite-state diagram which describes the allowable Connection state transitions. This finite-state diagram must be guaranteed by the implementation. Each method which causes a change in a Connection state must be consistent with this state diagram. This finite state diagram is below:

Note there is a general left-to-right progression of the state transitions. A Connection object may transition into and out of the `Connection.UNKNOWN` state at any time (hence, the asterisk qualifier next to its bidirectional transition arrow).



The Connection.disconnect() Method

The primary method supported on the core package's Connection interface is the `Connection.disconnect()` method. This method drops an entire Connection from a telephone call. The result of this method is to move the Connection object into the `Connection.DISCONNECTED` state. See the specification of the `Connection.disconnect()` method on this page for more detailed information.

Observers and Events

All events pertaining to the Connection object are reported via the `CallObserver` interface on the Call object associated with this Connection. In the core package, events are reported to a CallObserver when a new Connection is created and whenever a Connection changes state. Observers are added to Call objects via the `Call.addObserver()` method and more indirectly via the `Address.addCallObserver()` and `Terminal.addCallObserver()` methods. See the specifications for the Call, Address, and Terminal interfaces for more information.

The following Connection-related events are defined in the core package. Each of these events extend the `ConnEv` interface (which, in turn, extends the `CallEv` interface).

`ConnCreatedEv` Indicates a new Connection has been created on a Call.

`ConnInProgressEv` Indicates the Connection has moved into the `Connection.INPROGRESS` state.

`ConnAlertingEv` Indicates the Connection has moved into the `Connection.ALERTING` state.

`ConnConnectedEv`	Indicates the Connection has moved into the `Connection.CONNECTED` state.
`ConnDisconnectedEv`	Indicates the Connection has moved into the `Connection.DISCONNECTED` state.
`ConnFailedEv`	Indicates the Connection has moved into the `Connection.FAILED` state.
`ConnUnknownEv`	Indicates the Connection has moved into the `Connection.UNKNOWN` state.

See Also:

CallObserver, ConnEv, ConnCreatedEv, ConnInProgressEv, ConnAlertingEv, ConnConnectedEv, ConnDisconnectedEv, ConnFailedEv, ConnUnknownEv

---

Variable Index

- **ALERTING**

  The `Connection.ALERTING` state implies that the Address is being notified of an incoming call.

- **CONNECTED**

  The `Connection.CONNECTED` state implies that a Connection and its Address is actively part of a telephone call.

- **DISCONNECTED**

  The `Connection.DISCONNECTED` state implies it is no longer part of the telephone call, although its references to Call and Address still remain valid.

- **FAILED**

  The `Connection.FAILED` state indicates that a Connection to that end of the call has failed for some reason.

- **IDLE**

  The `Connection.IDLE` state is the initial state for all new Connections.

- **INPROGRESS**

  The `Connection.INPROGRESS` state implies that the Connection, which represents the destination end of a telephone call, is in the process of contacting the destination side.

- **UNKNOWN**

  The `Connection.UNKNOWN` state implies that the implementation is unable to determine the current state of the Connection.

Method Index

- **disconnect**()

  Drops a Connection from an active telephone call.

- **getAddress**()

  Returns the Address object associated with this Connection.

- **getCall**()

  Returns the Call object associated with this Connection.

- **getCapabilities**()

  Returns the dynamic capabilities for the instance of the Connection object.

- **getConnectionCapabilities**(Terminal, Address)

  Gets the ConnectionCapabilities object with respect to a Terminal and an Address. **Deprecated.**

- **getState**()

  Returns the current state of the Connection.

getTerminalConnections()

Returns an array of TerminalConnection objects associated with this Connection.

Variables

IDLE

```
public static final int IDLE
```

The `Connection.IDLE` state is the initial state for all new Connections. Connections which are in the Connection.IDLE state are not actively part of a telephone call, yet their references to the Call and Address objects are valid. Connections typically do not stay in the `Connection.IDLE` state for long, quickly transitioning to other states.

INPROGRESS

```
public static final int INPROGRESS
```

The `Connection.INPROGRESS` state implies that the Connection, which represents the destination end of a telephone call, is in the process of contacting the destination side. Under certain circumstances, the Connection may not progress beyond this state. Extension packages elaborate further on this state in various situations.

ALERTING

```
public static final int ALERTING
```

The `Connection.ALERTING` state implies that the Address is being notified of an incoming call.

CONNECTED

```
public static final int CONNECTED
```

The `Connection.CONNECTED` state implies that a Connection and its Address is actively part of a telephone call. In common terms, two people talking to one another are represented by two Connections in the `Connection.CONNECTED` state.

DISCONNECTED

```
public static final int DISCONNECTED
```

The `Connection.DISCONNECTED` state implies it is no longer part of the telephone call, although its references to Call and Address still remain valid. A Connection in the `Connection.DISCONNECTED` state is interpreted as once previously belonging to this telephone call.

FAILED

```
public static final int FAILED
```

The `Connection.FAILED` state indicates that a Connection to that end of the call has failed for some reason. One reason why a Connection would be in the `Connection.FAILED` state is because the party was busy.

UNKNOWN

```
public static final int UNKNOWN
```

The `Connection.UNKNOWN` state implies that the implementation is unable to determine the current state of the Connection. Typically, method are invalid on Connections which are in the `Connection.UNKNOWN` state. Connections may move in and out of this state at any time.

Methods

getState

```
public abstract int getState()
```

Returns the current state of the Connection. The return value will be one of states defined above.

Returns:

The current state of the Connection.

getCall

```
public abstract Call getCall()
```

Returns the Call object associated with this Connection. This Call reference remains valid throughout the lifetime of the Connection object, despite the state of the Connection object. This Call reference does not change once the Connection object has been created.

Returns:

The call object associated with this Connection.

getAddress

```
public abstract Address getAddress()
```

Returns the Address object associated with this Connection. This Address object reference remains valid throughout the lifetime of the Connection object despite the state of the Connection object. This Address reference does not change once the Connection object has been created.

Returns:

The Address associated with this Connection.

getTerminalConnections

```
public abstract TerminalConnection[] getTerminalConnections()
```

Returns an array of TerminalConnection objects associated with this Connection. TerminalConnection objects represent the relationship between a Connection and a specific Terminal endpoint. There may be zero TerminalConnections associated with this Connection. In that case, this method returns null. Connection objects lose their reference to a TerminalConnection once the TerminalConnection moves into the `TerminalConnection.DROPPED` state.

Post-conditions:

1. Let TerminalConnection tc[] = this.getTerminalConnections()
2. tc == null or tc.length >= 1
3. For all i, tc[i].getState() != TerminalConnection.DROPPED

Returns:

An array of TerminalConnection objects associated with this Connection, null if there are no TerminalConnections.

disconnect

```
public abstract void disconnect() throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException, InvalidStateException
```

Drops a Connection from an active telephone call. The Connection's Address is no longer associated with the telephone call. This method does not necessarily drop the entire telephone call, only the particular Connection on the telephone call. This method provides the ability to disconnect a specific party from a telephone call, which is especially useful in telephone calls consisting of three or more parties. Invoking this method may result in the entire telephone call being dropped, which is a permitted outcome of this method. In that case, the appropriate events are delivered to the application, indicating that more than just a single Connection has been dropped from the telephone call.

Allowable Connection States

The Connection object must be in one of several states in order for this method to be successfully invoked. These allowable states are: `Connection.CONNECTED`, `Connection.ALERTING`, `Connection.INPROGRESS`, or `Connection.FAILED`. If the Connection is not in one of these allowable states when this method is invoked, this method throws InvalidStateException. Having the

Connection in one of the allowable states does not guarantee a successful invocation of this method.

Method Return Conditions

This method returns successfully only after the Connection has been disconnected from the telephone call and has transitioned into the `Connection.DISCONNECTED`. This method may return unsuccessfully by throwing one of the exceptions listed below. Note that this method waits (i.e. the invocating thread blocks) until either the Connection is actually disconnected from the telephone call or an error is detected and an exception thrown. Also, all of the TerminalConnections associated with this Connection are moved into the `TerminalConnection.DROPPED` state. As a result, they are no longer reported via the Connection by the `Connection.getTerminalConnections()` method.

As a result of this method returning successfully, one or more events are delivered to the application. These events are listed below:

1. A ConnDisconnectedEv event for this Connection.
2. A TermConnDroppedEv event for all TerminalConnections associated with this Connection.

Dropping Additional Connections

Additional Connections may be dropped indirectly as a result of this method. For example, dropping the destination Connection of a two-party Call may result in the entire telephone call being dropped. It is up to the implementation to determine which Connections are dropped as a result of this method. Implementations should not, however, drop additional Connections if it does not reflect the natural response of the underlying telephone hardware.

Dropping additional Connections implies that their TerminalConnections are dropped as well. Also, if all of the Connections on a telephone call are dropped as a result of this method, the Call will move into the `Call.INVALID` state. The following lists additional events which may be delivered to the application.

1. ConnDisconnectedEv/TermConnDroppedEv are delivered for all other Connections and TerminalConnections dropped indirectly as a result of this method.
2. CallInvalidEv if all of the Connections are dropped indirectly as a result of this method.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Connection.CONNECTED or Connection.ALERTING or Connection.INPROGRESS or Connection.FAILED
3. Let TerminalConnection tc[] = this.getTerminalConnections (see post- conditions)

Post-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Connection.DISCONNECTED
3. For all i, tc[i].getState() == TerminalConnection.DROPPED
4. this.getTerminalConnections() == null.
5. this is not an element of (this.getCall()).getConnections()
6. ConnDisconnectedEv is delivered for this Connection.
7. TermConnDroppedEv is delivered for all TerminalConnections associated with this Connection.
8. ConnDisconnectedEv/TermConnDroppedEv are delivered for all other Connections and TerminalConnections dropped indirectly as a result of this method.
9. CallInvalidEv if all of the Connections are dropped indirectly as a result of this method.

Throws: PrivilegeViolationException

The application does not have the authority or permission to disconnected the Connection. For example, the Address associated with this Connection may not be controllable in the Provider's domain.

Throws: ResourceUnavailableException

An internal resource required to drop a connection is not available.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: InvalidStateException

Some object required for the successful invocation of this method is not in the proper state as given by this method's pre-conditions. For example, the Provider may not be in the Provider.IN_SERVICE state or the Connection may not be in one of the allowable states.

See Also:

ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv

getCapabilities

```
 public abstract ConnectionCapabilities getCapabilities()
```

Returns the dynamic capabilities for the instance of the Connection object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon the current state of the call model or some implementation-specific knowledge. These indications do not guarantee that a particular method can be successfully invoked, however.

The dynamic Connection capabilities require no additional arguments.

Returns:

The dynamic Connection capabilities.

getConnectionCapabilities

```
 public abstract ConnectionCapabilities getConnectionCapabilities(Terminal terminal,
                                                                  Address address)
throws InvalidArgumentException, PlatformException
```

Note: getConnectionCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Connection.getCapabilities() method.*

Gets the ConnectionCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal parameter, the general/provider-wide Connection capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Connection.getCapabilities()` method returns the dynamic Connection capabilities. This method now should simply invoke the `Connection.getCapabilities()` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

address - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

All Packages   This Package   Previous   Next

All Packages This Package Previous Next

Interface javax.telephony.JtapiPeer

public interface **JtapiPeer**

Introduction

The `JtapiPeer` interface represents a vendor's particular implementation of the Java Telephony API. Each JTAPI implementation vendor must implement this interface. The JtapiPeer object returned by the `JtapiPeerFactory.getJtapiPeer()` method determines which Providers are made available to the application.

Obtaining a Provider

Applications use the `JtapiPeer.getProvider()` method on this interface to obtain new Provider objects. Each implementation may support one or more different "services" (e.g. for different types of underlying network substrate). A list of available services can be obtained via the `JtapiPeer.getServices()` method.

Applications may also supply optional arguments to the Provider through the `JtapiPeer.getProvider()` method. These arguments are appended to the `providerString` argument passed to the `JtapiPeer.getProvider()` method. The `providerString` argument has the following format:

< service name > ; arg1 = val1; arg2 = val2; ...

Where < service name > is not optional, and each optional argument pair which follows is separated by a semi-colon. The keys for these arguments is implementation specific, except for two standard-defined keys:

1. login: provides the login user name to the Provider.
2. passwd: provides a password to the Provider.

See Also:

JtapiPeerFactory

Method Index

getName()

Returns the name of this JtapiPeer object instance.

getProvider(String)

Returns an instance of a `Provider` object given a string argument which contains the desired service name.

getServices()

Returns the services that this implementation supports.

Methods

getName

```
 public abstract String getName()
```

Returns the name of this JtapiPeer object instance. This name is the same name used as an argument to *JtapiPeerFactory.getJtapiPeer()* method.

Returns:

The name of this JtapiPeer object instance.

getServices

```
 public abstract String[] getServices()
```

Returns the services that this implementation supports. This method returns `null` if no services as supported.

Returns:

The services that this implementation supports.

getProvider

```
 public abstract Provider getProvider(String providerString) throws
ProviderUnavailableException
```

Returns an instance of a `Provider` object given a string argument which contains the desired service name. Optional arguments may also be provided in this string, with the following format:

< service name > ; arg1 = val1; arg2 = val2; ...

Where < service name > is not optional, and each optional argument pair which follows is separated by a semi-colon. The keys for these arguments is implementation specific, except for two standard-defined keys:

1. login: provides the login user name to the Provider.
2. passwd: provides a password to the Provider.

If the argument is null, this method returns some default Provider as determined by the JtapiPeer object. The returned Provider is in the `Provider.OUT_OF_SERVICE` state.

Post-conditions:

1. this.getProvider().getState() == Provider.OUT_OF_SERVICE

Parameters:

providerString - The name of the desired service plus an optional arguments.

Returns:

An instance of the Provider object.

Throws: ProviderUnavailableException

Indicates a Provider corresponding to the given string is unavailable.

Interface javax.telephony.Provider

public interface **Provider**

Introduction

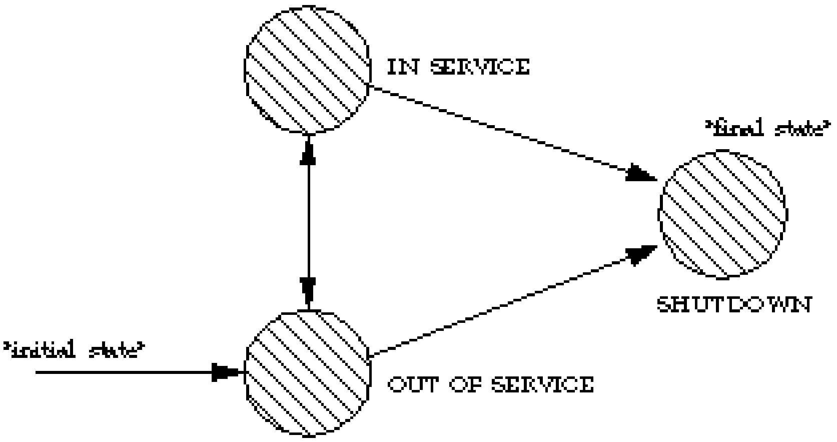
A Provider represents the telephony software-entity that interfaces with a telephony subsystem. The telephony subsystem could be a PBX connected to a server machine, a telephony/fax card in a desktop machine or a networking technology such as IP or ATM.

Provider States

The Provider may either be in one of the following states: `Provider.IN_SERVICE`, `Provider.OUT_OF_SERVICE`, or `Provider.SHUTDOWN`. The Provider state represents whether any action on that Provider may be valid. The following tables describes each state:

`Provider.IN_SERVICE`	This state indicates that the Provider is currently alive and available for use.
`Provider.OUT_OF_SERVICE`	This state indicates that a Provider is temporarily not available for use. Many methods in the Java Telephony API are invalid when the Provider is in this state. Providers may come back in service at any time, however, the application can take no direct action to cause this change.
`Provider.SHUTDOWN`:	This state indicates that a Provider is permanently no longer available for use. Most methods in the Java Telephony API are invalid when the Provider is in this state. Applications may use the `Provider.shutdown()` method on this interface to cause a Provider to move into the `Provider.SHUTDOWN` state.

The following diagram shows the allowable state transitions for the Provider as defined by the core package.



Obtaining a Provider

A Provider is created and returned by the `JtapiPeer.getProvider()` method which is given a string to describe the desired Provider. This method sets up any needed communication paths between the application and the Provider. The string given is one of the services listed in the `JtapiPeer.getServices()`. JtapiPeers particular implementation on a system and may be obtained via the `JtapiPeerFactory` class.

Observers and Events

Each time a state changes occurs on a Provider, the application is notified via an *event*. This event is reported via the `ProviderObserver` interface. Applications instantiate objects which implement this interface and use the `Provider.addObserver()` method to begin the delivery of events. All Provider events reported via this interface extend the `ProvEv` interface. Applications may then query the event object returned for the specific state change. In the core package API, the following events are sent to the ProviderObserver when the Provider changes state:

ProvInServiceEv, ProvOutOfServiceEv, and ProvShutdownEv. The ProvObservationEndedEv event is delivered to all observers when the Provider becomes unobservable and is the final event delivered to the observer.

Call Objects and Providers

The Provider maintains knowledge of the calls currently associated with it. Applications may obtain an array of these Calls via the `Provider.getCalls()` method. A Provider may have Calls associated with it which were created before it came into existence. It is the responsibility of the implementation of the Provider to model and report all existing telephone calls which were created prior to the Provider's lifetime. The Provider maintains references to all calls until they move into the `Call.INVALID` state.

Applications may create new Calls using the `Provider.createCall()` method. A new Call is returned in the `Call.IDLE` state. Applications may then use this idle Call to place new telephone calls. Once created, this new Call object is returned via the `Provider.getCalls()` method.

The Provider's domain

The term *Provider's domain* refers to the collection of Address and Terminal objects which are local to the Provider, and typically, can be controlled by the Provider. For example, the domain of a Provider of a desktop system with an ISDN card are the Address(es) and Terminal(s) which represent that ISDN endpoint. The domain of a Provider for a PBX may be the Addresses and Terminals in that PBX. The Provider implementation controls access to Addresses and Terminals by limiting the domain its presents to the application.

Address and Terminal Objects

An Address object represents what we commonly think of as a "telephone number." In more rare implementations where the underlying network is not a telephony network, this address may represent something else, such as an IP address. Regardless, it represents a *logical* endpoint of a telephone call. A Terminal represents a physical hardware endpoint connected to the telephone network. An example of a Terminal is a telephone set, but a Terminal does not have to take the form of this limited and traditional example. Addresses and Terminals are in a many-to-many relationship. An Address may contain multiple Terminals, and Terminals may contain multiple Addresses. See the specifications for the Address and Terminal objects for more information.

Unlike Call objects, applications may not create Terminal or Address objects. The Provider begins with knowledge of certain Terminal and Address objects defined as its local domain. This list is static once the Provider is created. The Addresses and Terminals in the Provider's domain are reported via the `Provider.getAddresses()` and `Provider.getTerminals()` methods, respectively.

Other Addresses and Terminals may be created sometime during the operation of the Provider when the Provider learns of new instances of these objects. These new objects, however, represent Addresses and Terminals outside the Provider's domain. For example, if the Provider's domain is a PBX, the Provider will know about all Addresses and Terminals in this PBX when the Provider first starts. Any Addresses and Terminals it subsequently learns about are outside this PBX. These Address and Terminal objects outside this PBX are not reported via the `Provider.getTerminals()` and `Provider.getAddresses()` methods. Address and Terminal objects outside of the Provider's domain are referred to as *remote*.

Capabilities: Static and Dynamic

The Provider interface supports methods to return *static* capabilities for each of the Java Telephony call model objects. Static capabilities provide applications with information concerning the ability of the implementation to perform certain methods. These static capabilities indicate whether a method is implemented for a particular type of object and does not depend upon the particular instance of the object nor the current state of the call model. Those methods for which the static capability returns false will throw a MethodNotSupportedException when invoked. The static capability methods supported on this interface are: `Provider.getProviderCapabilities()`, `Provider.getCallCapabilities()`, `Provider.getAddressCapabilities()`, `Provider.getTerminalCapabilities()`, `Provider.getConnectionCapabilities()`, and `Provider.getTerminalConnectionCapabilities()`.

Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation-specific knowledge. These indications do not guarantee that a particular method can be successfully invoked, however. This interface returns the dynamic capabilities for a Provider object via the `Provider.getCapabilities()` method. Note that this method is distinct from the static capability method `Provider.getProviderCapabilities()`.

Multiple Providers and Multiple Applications

It is not guaranteed or expected that objects (Call, Terminal, etc.) instantiated through one Provider will be usable with another Provider. Therefore, an application that uses two providers must keep all the objects relating to these providers separate. In the future, there may be a mechanism whereby a Provider may share objects with another Provider if they are speaking to the same telephony hardware, however, such capabilities are not available in this release.

Also, multiple applications may request and communicate with the same Provider implementation. Typically, since each application executes in its own object space, each will have its own instance of the Provider object. These two different Provider objects may, in fact, be proxies for a centralized Provider instance. Certain methods in JTAPI are specified to affect only the invoking applications and have not affect on others. The only example in the core package is the `Provider.shutdown()` method.

See Also:

JtapiPeer, JtapiPeerFactory, ProviderObserver

Variable Index

- **IN_SERVICE**

  The `Provider.IN_SERVICE` state indicates that a Provider is currently available for use.

- **OUT_OF_SERVICE**

  The `Provider.OUT_OF_SERVICE` state indicates that a Provider is temporarily not available for use.

- **SHUTDOWN**

  The `Provider.SHUTDOWN` state indicates that a Provider is permanently no longer available for use.

Method Index

- **addObserver**(ProviderObserver)

  Adds an observer to the Provider.

- **createCall**()

  Creates and returns a new instance of the Call object.

- **getAddress**(String)

  Returns an Address object which corresponds to the telephone number string provided.

- **getAddressCapabilities**()

  Returns the static capabilities of the Address object.

- **getAddressCapabilities**(Terminal)

  Gets the AddressCapabilities object with respect to a Terminal. **Deprecated.**

- **getAddresses**()

  Returns an array of Addresses associated with the Provider and within the Provider's domain.

- **getCallCapabilities**()

  Returns the static capabilities of the Call object.

- **getCallCapabilities**(Terminal, Address)

  Gets the CallCapabilities object with respect to a Terminal and an Address. **Deprecated.**

- **getCalls**()

  Returns an array of Call objects currently associated with the Provider.

- **getCapabilities**()

  Returns the dynamic capabilities for the instance of the Provider object.

- **getConnectionCapabilities**()

  Returns the static capabilities of the Connection object.

- **getConnectionCapabilities**(Terminal, Address)

Gets the ConnectionCapabilities object with respect to a Terminal and an Address. **Deprecated.**

- **getName**()

  Returns the unique string name of the Provider.

- **getObservers**()

  Returns a list of all ProviderObservers associated with this Provider object.

- **getProviderCapabilities**()

  Returns the static capabilities of the Provider object.

- **getProviderCapabilities**(Terminal)

  Returns the ProviderCapabilities object with respect to a Terminal. **Deprecated.**

- **getState**()

  Returns the current state of the Provider, either `Provider.IN_SERVICE`, `Provider.OUT_OF_SERVICE`, or `Provider.SHUTDOWN`.

- **getTerminal**(String)

  Returns an instance of the Terminal class which corresponds to the given name.

- **getTerminalCapabilities**()

  Returns the static capabilities of the Terminal object.

- **getTerminalCapabilities**(Terminal)

  Gets the TerminalCapabilities object with respect to a Terminal. **Deprecated.**

- **getTerminalConnectionCapabilities**()

  Returns the static capabilities of the TerminalConnection object.

- **getTerminalConnectionCapabilities**(Terminal)

  Gets the TerminalConnectionCapabilities of a Terminal. **Deprecated.**

- **getTerminals**()

  Returns an array of Terminals associated with the Provider and within the Provider's domain.

- **removeObserver**(ProviderObserver)

  Removes the given observer from the Provider.

- **shutdown**()

  Instructs the Provider to shut itself down and perform all necessary cleanup.

Variables

IN_SERVICE

```
public static final int IN_SERVICE
```

The `Provider.IN_SERVICE` state indicates that a Provider is currently available for use.

OUT_OF_SERVICE

```
public static final int OUT_OF_SERVICE
```

The `Provider.OUT_OF_SERVICE` state indicates that a Provider is temporarily not available for use. Many methods in the Java Telephony API are invalid when the Provider is in this state. Providers may come back in service at any time, however, the application can take no direct action to cause this change.

SHUTDOWN

```
public static final int SHUTDOWN
```

The `Provider.SHUTDOWN` state indicates that a Provider is permanently no longer available for use. Most methods in the Java Telephony API are invalid when the Provider is in this state.

Methods

getState

```
public abstract int getState()
```

Returns the current state of the Provider, either `Provider.IN_SERVICE`, `Provider.OUT_OF_SERVICE`, or `Provider.SHUTDOWN`.

Returns:

The current state of the provider.

getName

```
public abstract String getName()
```

Returns the unique string name of the Provider. Each different Provider must have a unique string associated with it. This is the same string which the application passed to the `JtapiPeer.getProvider()` method to create this Provider instance.

Returns:

The name of the Provider.

See Also:

JtapiPeer

getCalls

```
public abstract Call[] getCalls() throws ResourceUnavailableException
```

Returns an array of Call objects currently associated with the Provider. When a Call moves into the `Call.INVALID` state, the Provider loses its reference to this Call. Therefore, all Calls returned by this method must either be in the `Call.IDLE` or `Call.ACTIVE` state. This method returns null if the Provider has zero calls associated with it.

Post-conditions:

1. Let Calls calls[] = Provider.getCalls()
2. calls == null or calls.length >= 1
3. For all i, calls[i].getState() == Call.IDLE or Call.ACTIVE

Returns:

An array of Call objects currently associated with this Provider.

Throws: ResourceUnavailableException

Indicates the number of calls present in the Provider is too great to return as a static array.

getAddress

```
public abstract Address getAddress(String number) throws InvalidArgumentException
```

Returns an Address object which corresponds to the telephone number string provided. If the provided name does not correspond to an Address known by the Provider and within the Provider's domain, InvalidArgumentException is thrown. In other words, the Address must appear in the list generated by `Provider.getAddresses()`.

Pre-conditions:

1. Let Address address = this.getAddress(number);
2. address is an element of this.getAddresses();

Post-conditions:

1. Let Address address = this.getAddress(number);
2. address is an element of this.getAddresses();

Parameters:

number - The telephone address string.

Returns:

The Address object corresponding to the given telephone number.

Throws: InvalidArgumentException

The name, e.g. telephone number of the Address does not correspond to the name of any Address object known to the Provider or within the Provider's domain.

getAddresses

```
public abstract Address[] getAddresses() throws ResourceUnavailableException
```

Returns an array of Addresses associated with the Provider and within the Provider's domain. This list is static (i.e. is does not change) after the Provider is first created. If no Address objects are associated with this Provider, then this method returns null.

Post-conditions:

1. Let Address[] addresses = this.getAddresses()
2. addresses == null or addresses.length >= 1

Returns:

An array of Addresses known by this provider.

Throws: ResourceUnavailableException

Indicates the number of addresses present in the Provider is too great to return as a static array.

getTerminals

```
public abstract Terminal[] getTerminals() throws ResourceUnavailableException
```

Returns an array of Terminals associated with the Provider and within the Provider's domain. Each Terminal possesses a unique name, which is assigned to it by the JTAPI implementation. If there are no Terminals associated with this Provider, then this method returns null.

Post-conditions:

1. Let Terminal[] terminals = this.getTerminals()
2. terminals == null or terminals.length >= 1

Returns:

An array of Terminals in the Provider's local domain.

Throws: ResourceUnavailableException

Indicates the number of terminals present in the Provider is too great to return as a static array.

getTerminal

```
public abstract Terminal getTerminal(String name) throws InvalidArgumentException
```

Returns an instance of the Terminal class which corresponds to the given name. Each Terminal has a unique name associated with it, which is assigned to it by the JTAPI implementation. If no Terminal is available for the given name within the Provider's domain, this method throws the InvalidArgumentException. This Terminal must be in the array generated by `Provider.getTerminals().`

Pre-conditions:

1. Let Terminal terminal = this.getTerminal(name);
2. terminals is an element of this.getTerminals();

Post-conditions:

1. Let Terminal terminal = this.getTerminal(name);
2. terminal is an element of this.getTerminals();

Parameters:

name - The name of desired Terminal object.

Returns:

The Terminal object associated with the given name.

Throws: InvalidArgumentException

The name provided does not correspond to a name of any Terminal known to the Provider or within the Provider's domain.

shutdown

```
public abstract void shutdown()
```

Instructs the Provider to shut itself down and perform all necessary cleanup. Applications invoke this method when they no longer intend to use the Provider, most often right before they exit. This method is intended to allow the Provider to perform any necessary cleanup which would not be taken care of when the Java objects are garbage collected. This method causes the Provider to move into the `Provider.SHUTDOWN` state, in which it will stay indefinitely.

If the Provider is already in the `Provider.SHUTDOWN` state, this method does nothing. The invocation of this method should not affect other applications which are using the same implementation of the Provider object.

Post-conditions:

1. this.getState() == Provider.SHUTDOWN

createCall

```
public abstract Call createCall() throws ResourceUnavailableException,
InvalidStateException, PrivilegeViolationException, MethodNotSupportedException
```

Creates and returns a new instance of the Call object. This new call object is in the `Call.IDLE state` and has no Connections. An exception is generated if a new call cannot be created for various reasons. This Provider must be in the `Provider.IN_SERVICE` state, otherwise an InvalidStateException is thrown.

Pre-conditions:

1. this.getState() == Provider.IN_SERVICE

Post-conditions:

1. this.getState() == Provider.IN_SERVICE
2. Let Call call = this.createCall();
3. call.getState() == Call.IDLE.
4. call.getConnections() == null
5. call is an element of this.getCalls()

Returns:

The new Call object.

Throws: ResourceUnavailableException

An internal resource necessary to create a new Call object is unavailable.

Throws: InvalidStateException

The Provider is not in the `Provider.IN_SERVICE` state.

Throws: PrivilegeViolationException

The application does not have the proper authority to create a new telephone call object.

Throws: MethodNotSupportedException

The implementation does not support creating new Call objects.

addObserver

```
public abstract void addObserver(ProviderObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to the Provider. Provider-related events are reported via the `ProviderObserver` interface. The Provider object will report events to this interface for the lifetime of the Provider object or until the observer is removed with the `Provider.removeObserver()` method or until the Provider is no longer observable.

If the Provider becomes unobservable, a ProvObservationEndedEv is delivered to the application as is final event. No further events are delivered to the observer unless it is explicitly re-added by the application. When an observer receives a ProvObservationEndedEv it is no longer reported via the `Provider.getObservers()` method.

This method is valid anytime and has no pre-conditions. Application must have the ability to add observers to Providers so they can monitor the changes in state in the Provider.

If an application attempts to add an instance of an observer already present on this Provider, then repeated attempts to add the instance of the observer will silently fail, i.e. multiple instances of an observer are not added and no exception will be thrown.

Post-conditions:

1. observer is an element of this.getObservers()

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time.

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

getObservers

```
public abstract ProviderObserver[] getObservers()
```

Returns a list of all ProviderObservers associated with this Provider object. If no observers exist on this Provider, then this method returns null.

Post-conditions:

1. Let ProviderObserver[] observers = this.getObservers()
2. observers == null or observers.length >= 1

Returns:

An array of ProviderObserver objects associated with this Provider.

removeObserver

```
public abstract void removeObserver(ProviderObserver observer)
```

Removes the given observer from the Provider. The given observer will no longer receive events generated by this Provider object. The final event will be the ProvObservationEndedEv event and will no longer be listed by the `Provider.getObservers()` method.

Also, if an observer is not part of the Provider, then this method fails silently, i.e. no observer is removed an no exception is thrown.

Post-conditions:

1. observer is not an element of this.getObservers()
2. ProvObservationEndedEv is delivered to observer

Parameters:

observer - The observer which is being removed.

getProviderCapabilities

```
public abstract ProviderCapabilities getProviderCapabilities()
```

Returns the static capabilities of the Provider object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the Provider object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

NOTE: This method is different from the `Provider.getCapabilities()`, method which returns the dynamic capabilities of a

Provider object instance.

Returns:

The static capabilities of the Provider object.

getCallCapabilities

```
public abstract CallCapabilities getCallCapabilities()
```

Returns the static capabilities of the Call object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the Call object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

Returns:

The static capabilities of the Call object.

getAddressCapabilities

```
public abstract AddressCapabilities getAddressCapabilities()
```

Returns the static capabilities of the Address object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the Address object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

Returns:

The static capabilities of the Address object.

getTerminalCapabilities

```
public abstract TerminalCapabilities getTerminalCapabilities()
```

Returns the static capabilities of the Terminal object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the Terminal object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

Returns:

The static capabilities of the Address object.

getConnectionCapabilities

```
public abstract ConnectionCapabilities getConnectionCapabilities()
```

Returns the static capabilities of the Connection object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the Connection object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

Returns:

The static capabilities of the Connection object.

getTerminalConnectionCapabilities

```
public abstract TerminalConnectionCapabilities getTerminalConnectionCapabilities()
```

Returns the static capabilities of the TerminalConnection object. The value of these capabilities will not change over the lifetime of the Provider. They represent the static abilities of the implementation to perform certain methods on the TerminalConnection object. For all capability methods which return false, the invocation of that method will always throw MethodNotSupportedException.

Returns:

The static capabilities of the TerminalConnection object.

getCapabilities

```
public abstract ProviderCapabilities getCapabilities()
```

Returns the dynamic capabilities for the instance of the Provider object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation-specific knowledge. These indications do not guarantee that a particular method can be successfully invoked, however.

There are no arguments passed into this method for dynamic Provider capabilities

NOTE: This method is different from the `Provider.getProviderCapabilities()` method which returns the static capabilities for the Provider object.

Returns:

The dynamic Provider capabilities.

getProviderCapabilities

```
 public abstract ProviderCapabilities getProviderCapabilities(Terminal terminal)
throws InvalidArgumentException, PlatformException
```

Note: getProviderCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getProviderCapabilities() method.*

Returns the ProviderCapabilities object with respect to a Terminal. If null is passed as a Terminal parameter, the general/provider-wide Provider capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getProviderCapabilities()` method returns the static Provider capabilities. This method now should simply invoke the `Provider.getProviderCapabilities(void)` method.

Parameters:

terminal - This parameter is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

getCallCapabilities

```
 public abstract CallCapabilities getCallCapabilities(Terminal terminal,
                                                     Address address) throws
InvalidArgumentException, PlatformException
```

Note: getCallCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getCallCapabilities() method.*

Gets the CallCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal/Address parameter, the general/provider-wide Call capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getCallCapabilities()` method returns the static Call capabilities. This method now should simply invoke the `Provider.getCallCapabilities(void)` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

address - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

getConnectionCapabilities

```
 public abstract ConnectionCapabilities getConnectionCapabilities(Terminal terminal,
                                                                 Address address)
throws InvalidArgumentException, PlatformException
```

Note: getConnectionCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getConnectionCapabilities() method.*

Gets the ConnectionCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal/Address parameter, the general/ provider-wide Connection capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getConnectionCapabilities()` method returns the static Connection capabilities. This method now should simply invoke the `Provider.getConnectionCapabilities(void)` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

exception - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

getAddressCapabilities

```
 public abstract AddressCapabilities getAddressCapabilities(Terminal terminal)
throws InvalidArgumentException, PlatformException
```

Note: getAddressCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getAddressCapabilities() method.*

Gets the AddressCapabilities object with respect to a Terminal. If null is passed as a Terminal parameter, the general/provider-wide Address capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getAddressCapabilities()` method returns the static Address capabilities. This method now should simply invoke the `Provider.getAddressCapabilities(void)` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

getTerminalConnectionCapabilities

```
 public abstract TerminalConnectionCapabilities
getTerminalConnectionCapabilities(Terminal terminal) throws
InvalidArgumentException, PlatformException
```

Note: getTerminalConnectionCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getTerminalConnectionCapabilities() method.*

Gets the TerminalConnectionCapabilities of a Terminal. If null is passed as a Terminal parameter, the general/provider-wide TerminalConnection capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getTerminalConnectionCapabilities()` method returns the static TerminalConnection capabilities. This method now should simply invoke the `Provider.getTerminalConnectionCapabilities(void)` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later. are being queried

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

getTerminalCapabilities

```
 public abstract TerminalCapabilities getTerminalCapabilities(Terminal terminal)
throws InvalidArgumentException, PlatformException
```

Note: getTerminalCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Provider.getTerminalCapabilities() method.*

Gets the TerminalCapabilities object with respect to a Terminal. If null is passed as a Terminal parameter, the general/provider-wide Terminal capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Provider.getTerminalCapabilities()` method returns the static Terminal capabilities. This method now should simply invoke the `Provider.getTerminalCapabilities(void)` method.

Parameters:

terminal - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

All Packages This Package Previous Next

Interface javax.telephony.ProviderObserver

public interface **ProviderObserver**

Introduction

The `ProviderObserver` interface reports all changes which happen to the Provider object. These changes are reported as events to the `ProviderObserver.providerChangedEvent()` method. Applications must instantiate an object which implements this interface and then use the `Provider.addObserver()`) method to register the object to receive all future events associated with the Provider object.

The `ProviderObserver.providerChangedEvent()` method receives an array of events which all must extend the `ProvEv` interface. Since several changes may happen to a single JTAPI object at once, a list of events is needed to convey those changes which happen at the same time. Applications iterate through the array of events provided.

Provider State Changes

In the core package, an event is delivered whenever the Provider changes state. The event interfaces which correspond to these state changes for the core package are: `ProvInServiceEv`, `ProvOutOfServiceEv`, and `ProvShutdownEv`.

Provider Observation Ending

At various times, the underlying implementation may not be able to observe the Provider. In these instances, the ProviderObserver is sent an ProvObservationEndedEv event. This indicates that the application will not receive further events associated with the Provider object. This observer will no longer be reported via the `Provider.getObservers()` method.

See Also:

ProvEv, ProvInServiceEv, ProvOutOfServiceEv, ProvShutdownEv, ProvObservationEndedEv

Method Index

providerChangedEvent(ProvEv[])

Reports all events associated with the Provider object.

Methods

providerChangedEvent

```
 public abstract void providerChangedEvent(ProvEv eventList[])
```

Reports all events associated with the Provider object. This method passes an array of ProvEv objects as its arguments which correspond to the list of events representing the changes to the Provider object.

Parameters:

eventList - The list of Provider events.

Interface javax.telephony.Terminal

public interface **Terminal**

Introduction

A Terminal represents a physical hardware endpoint connected to the telephony domain. An example of a Terminal is a telephone set, but a Terminal does not have to take the form of this limited and traditional example. For example, computer workstations and hand-held devices are modeled as Terminals if they act as physical endpoints in a telephony network.

A Terminal object has a string *name* which is unique for all Terminal objects. The Terminal does not attempt to interpret this string in any way. This name is first assigned when the Terminal is created and does not change throughout the lifetime of the object. The method `Terminal.getName()` returns the name of the Terminal object. The name of the Terminal may not have any real-world interpretation. Typically, Terminals are chosen from a list of Terminals obtained from an Address object.

Terminal objects may be classified into two categories: *local* and *remote*. Local Terminal objects are those terminals which are part of the Provider's local domain. These Terminal objects are created by the implementation of the Provider object when it is first instantiated. All of the Provider's local terminals are reported via the `Provider.getTerminals()` method. Remote Terminal objects are those outside of the Provider's domain which the Provider learns about during its lifetime through various happenings (e.g. an incoming call from a currently unknown address). Remote Terminal objects are **not** reported via the `Provider.getTerminals()` method. Note that applications never explicitly create new Terminal objects.

Address and Terminal objects

Address and Terminal objects exist in a many-to-many relationship. An Address object may have zero or more Terminals associated with it. For each Terminal associated with an Address, that Terminal must also reflect its association with the Address. Since the implementation creates Address (and Terminal) objects, it is responsible for insuring the correctness of these relationships. The Terminals associated with an Address is given by the `Address.getTerminals()` method.

An association between an Address and Terminal object indicates that the Terminal contains the Address object as one of its telephone number addresses. In many instances, a telephone set (represented by a Terminal object) has only one telephone number (represented by an Address object) associated with it. In more complex configurations, telephone sets may have several telephone numbers associated with them. Likewise, a telephone number may appear on more than one telephone set. For example, feature phones in PBX environments may exhibit this configuration.

Terminals and Call objects

Terminal objects represent the *physical* endpoints of a telephone call. With respect to a single Address endpoint on a Call, multiple physical Terminal endpoints may exist. Terminal objects are related to Call objects via the TerminalConnection object. TerminalConnection objects are associated with Call indirectly via Connections. A Terminal may be associated with a Call only if one of its Addresses is associated with the Call. The TerminalConnection object has a *state* which describes the current relationship between the Connection and the Terminal. Each Terminal object may be part of more than one telephone call, and in each case, is represented by a separate TerminalConnection objet. The `Terminal.getTerminalConnections()` method returns all TerminalConnection object currently associated with the Terminal.

A Terminal object is associated with a Connection until the TerminalConnection moves into the `TerminalConnection.DROPPED` state. At that time, the TerminalConnection is no longer reported via the `Terminal.getTerminalConnections()` method. Therefore, the `Terminal.getTerminalConnections()` method never reports a TerminalConnection in the `TerminalConnection.DROPPED` state.

Existing Telephone Calls

The Java Telephony API specification states that the implementation is responsible for reporting all existing telephone calls when a Provider is first created. This implies that an Terminal object must report information regarding existing telephone calls to that Terminal. In other words, Terminal objects must report all TerminalConnection objects which represent existing telephone calls.

Terminal Observers and Events

All changes in an Terminal object are reported via the TerminalObserver interface. Applications instantiate an object which implements this interface and begins this delivery of events to this object using the `Terminal.addObserver()` method. All Terminal-related events extend the `TermEv` interface provided in the core package. Applications receive events on an observer until the observer is removed via the `Terminal.removeObserver()` method or until the Terminal is no longer observable. In these instances, each TerminalObserver receives a TermObservationEndedEv as its final event.

Currently in the core package, the only Terminal-related event is TermObservationEndedEv.

Call Observers

At times, applications may want to monitor a particular Terminal for all Calls which come to that Terminal. For example, a desktop telephone application is only interested in telephone calls associated with a particular agent terminal. To achieve this sort of Terminal-based Call monitoring applications may *add* CallObservers to an Terminal via the `Terminal.addCallObserver()` method.

When a CallObserver is added to an Terminal, this observer instance is immediately added to all Calls at this Terminal and is added to all Calls which come to this Terminal in the future. These observers remain on the telephone call as long as the Terminal is associated with the telephone call.

The specification of `Terminal.addCallObserver()` contains more precise semantics.

See Also:

TerminalObserver, CallObserver

Method Index

- **addCallObserver**(CallObserver)

  Adds an observer to a Call object when this Terminal object first becomes part of that Call.
- **addObserver**(TerminalObserver)

  Adds an observer to the Terminal.
- **getAddresses**()

  Returns an array of Address objects associated with this Terminal object.
- **getCallObservers**()

  Returns a list of all CallObservers associated with this Terminal object.
- **getCapabilities**()

  Returns the dynamic capabilities for the instance of the Terminal object.
- **getName**()

  Returns the name of the Terminal.
- **getObservers**()

  Returns a list of all TerminalObservers associated with this Terminal object.
- **getProvider**()

  Returns the Provider associated with this Terminal.
- **getTerminalCapabilities**(Terminal, Address)

  Gets the TerminalCapabilities object with respect to a Terminal and an Address. **Deprecated.**
- **getTerminalConnections**()

  Returns an array of TerminalConnection objects associated with this Terminal.
- **removeCallObserver**(CallObserver)

  Removes the given CallObserver from the Terminal.
- **removeObserver**(TerminalObserver)

  Removes the given observer from the Terminal.

Methods

getName

```
public abstract String getName()
```

Returns the name of the Terminal. Each Terminal possesses a unique name. This name is assigned by the implementation and may or may not carry a real-world interpretation.

Returns:

The name of the Terminal.

getProvider

```
public abstract Provider getProvider()
```

Returns the Provider associated with this Terminal. This Provider object is valid throughout the lifetime of the Terminal and does not change once the Terminal is created.

Returns:

The Provider associated with this Terminal.

getAddresses

```
public abstract Address[] getAddresses()
```

Returns an array of Address objects associated with this Terminal object. The Terminal object must have at least one Address object associated with it. This list does not change throughout the lifetime of the Terminal object.

Post-conditions:

1. Let Address[] addrs = this.getAddresses()
2. addrs.length >= 1

Returns:

An array of Address objects associated with this Terminal.

getTerminalConnections

```
public abstract TerminalConnection[] getTerminalConnections()
```

Returns an array of TerminalConnection objects associated with this Terminal. Once a TerminalConnection is added to a Terminal, the Terminal maintains a reference until the TerminalConnection moves into the `TerminalConnection.DROPPED` state. Therefore, all TerminalConnections returned by this method will never be in the `TerminalConnection.DROPPED` state. If there are no TerminalConnections associated with this Terminal, this method returns null.

Post-conditions:

1. Let TerminalConnection tc[] = this.getTerminalConnections()
2. tc == null or tc.length >= 1
3. For all i, tc[i].getState() != TerminalConnection.DROPPED

Returns:

An array of TerminalConnection objects associated with this Terminal.

addObserver

```
 public abstract void addObserver(TerminalObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to the Terminal. The TerminalObserver reports all Terminal-related state changes as events. The Terminal object will report events to this TerminalObserver object for the lifetime of the Terminal object or until the observer is removed with the

`Terminal.removeObserver()` or until the Terminal is no longer observable. In these instances, a TermObservationEndedEv is delivered to the observer as its final event. The observer will receive no events after TermObservationEndedEv unless the observer is explicitly re-added via the `Terminal.addObserver()` method. Also, once an observer receives an TermObservationEndedEv, it will no longer be reported via the `Terminal.getObservers()`.

If an application attempts to add an instance of an observer already present on this Terminal, this attempt will silently fail, i.e. multiple instances of an observer are not added and no exception will be thrown.

Currently, only the TermObservationEndedEv event is defined by the core package and delivered to the TerminalObserver.

Post-conditions:

1. observer is an element of this.getObservers()

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

getObservers

```
public abstract TerminalObserver[] getObservers()
```

Returns a list of all TerminalObservers associated with this Terminal object. If there are no observers associated with this Terminal, this method returns null.

Post-conditions:

1. Let TerminalObserver[] obs = this.getObservers()
2. obs == null or obs.length >= 1

Returns:

An array of TerminalObserver objects associated with this Terminal.

removeObserver

```
public abstract void removeObserver(TerminalObserver observer)
```

Removes the given observer from the Terminal. If successful, the observer will no longer receive events generated by this Terminal object. As its final event, the TerminalObserver receives a TermObservationEndedEv.

If an observer is not part of the Terminal, then this method fails silently, i.e. no observer is removed an no exception is thrown.

Post-conditions:

1. A TermObservationEndedEv event is reported to the observer as its final event.
2. observer is not an element of this.getObservers()

Parameters:

observer - The observer which is being removed.

addCallObserver

```
public abstract void addCallObserver(CallObserver observer) throws
ResourceUnavailableException, MethodNotSupportedException
```

Adds an observer to a Call object when this Terminal object first becomes part of that Call. This method permits applications to select a Terminal object in which they are interested and automatically have the implementation attach an observer to all present and future Calls which come to this Terminal.

JTAPI v1.0 Semantics

In version 1.0 of the Java Telephony API specification, the application monitored the Terminal object for the TermCallAtTermEv event. This event indicated that a Call has come to this Terminal. Then, applications would manually add an observer to the Call. With this architecture, potentially dangerous race conditions existed while an application was adding an observer to the Call. As a result, this mechanism has been replaced in version 1.1.

JTAPI v1.1 Semantics

In version 1.1 of the specification, the TermCallAtTermEv event does not exist and this method replaces the functionality described above. Instead of monitoring for a TermCallAtTermEv event, the application simply uses the `Terminal.addCallObserver()` method, and observer will be added to new telephone calls at this Terminal automatically.

If an application attempts to add an instance of a call observer already present on this Terminal, these repeated attempts will silently fail, i.e. multiple instances of a call observer are not added and no exception will be thrown.

When a call observer is added to an Terminal with this method, the following behavior is exhibited by the implementation.

1. It is immediately associated with any existing calls at the Terminal and a snapshot of those calls are reported to the call observer. For each of these calls, the observer is reported via `Call.getObservers()`.
2. It is associated with all future calls which come to this Terminal. For each new call, the observer is reported via `Call.getObservers().`

Note that the definition of the term ".. when a call is at a Terminal" means that the Call contains a Connection which contains a TerminalConnection with this Terminal as its Terminal.

Call Observer Lifetime

For all call observers which are present on Calls because of this method, the following behavior is exhibited with respect to the lifetime of the call.

1. The call observer receives events until the Call is no longer at this Terminal. In this case, the call observer will be re-applied to the Call if the Call returns to this Terminal at some point in the future.
2. The call observer is removed with `Call.removeObserver()`. Note that this only affects the instance of the call observer for that particular call. If the Call subsequently leaves and returns to the Terminal, the observer will be re-applied.
3. The Call can no longer be monitored by the implementation.
4. The Call moves into the `Call.INVALID` state.

When the CallObserver leaves the Call because of one of the reasons above, it receives a CallObservationEndedEv as its final event.

Call Observer on Multiple Addresses and Terminals

It is possible for an application to add CallObservers to more than one Address and Terminal (using `Address.addCallObserver()` and `Terminal.addCallObserver()`, respectively). The rules outlined above still apply, with the following additions:

1. A CallObserver is not added to a Call more than once, even if it has been added to more than one Address/Terminal which are present on the Call.
2. The CallObserver leaves the call only if ALL of the Addresses and Terminals on which the Call Observer was added leave the Call. If one of those Addresses/Terminals becomes part of the Call again, the call observer is re-applied to the Call.

Post-Conditions:

1. observer is an element of this.getCallObservers()
2. observer is an element of Call.getObservers() for each Call associated with the Connections from this.getConnections().
3. An array of snapshot events are reported to the observer for existing calls associated with this Terminal.

Parameters:

observer - The observer being added.

Throws: MethodNotSupportedException

The observer cannot be added at this time

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

See Also:

Call

getCallObservers

```
public abstract CallObserver[] getCallObservers()
```

Returns a list of all CallObservers associated with this Terminal object. That is, it returns a list of CallObserver objects which have been added via the `Terminal.addCallObserver()` method. If there are no Call observers associated with this Terminal object, this method returns null.

Post-conditions:

1. Let CallObserver[] obs = this.getCallObservers()
2. obs == null or obs.length >= 1

Returns:

An array of CallObserver objects associated with this Address.

removeCallObserver

```
public abstract void removeCallObserver(CallObserver observer)
```

Removes the given CallObserver from the Terminal. In other words, it removes a CallObserver which was added via the `Terminal.addCallObserver()` method. If successful, the observer will no longer be added to new Calls which are presented to this Terminal, however it does not affect CallObservers which have already been added at a Call.

Also, if the CallObserver is not part of the Terminal, then this method fails silently, i.e. no observer is removed and no exception is thrown.

Post-conditions:

1. observer is not an element of this.getCallObservers()

Parameters:

observer - The CallObserver which is being removed.

getCapabilities

```
public abstract TerminalCapabilities getCapabilities()
```

Returns the dynamic capabilities for the instance of the Terminal object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementation's knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation-specific knowledge. These indications do not guarantee that a particular method will be successful when invoked, however.

The dynamic Terminal capabilities require no additional arguments.

Returns:

The dynamic Terminal capabilities.

getTerminalCapabilities

```
 public abstract TerminalCapabilities getTerminalCapabilities(Terminal terminal,
                                                              Address address)
throws InvalidArgumentException, PlatformException
```

Note: getTerminalCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the Terminal.getCapabilities() method.*

Gets the TerminalCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal parameter, the general/provider- wide Terminal capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `Terminal.getCapabilities()` method returns the dynamic Terminal capabilities. This method now should simply invoke the `Terminal.getCapabilities()` method.

Parameters:

address - This argument is ignored in JTAPI v1.2 and later.

terminal - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

All Packages This Package Previous Next

Interface javax.telephony.TerminalConnection

public interface **TerminalConnection**

Introduction

The TerminalConnection represents the relationship between a Connection and a Terminal. A TerminalConnection object must always be associated with a Connection object and a Terminal object. The Connection and the Terminal objects associated with the TerminalConnection do not change throughout the lifetime of the TerminalConnection. Applications obtain the Connection and Terminal associated with the TerminalConnection via the `TerminalConnection.getConnection()` and `TerminalConnection.getTerminal()` methods, respectively.

Because a TerminalConnection is associated with a Connection, it there is also associated with some Call. The TerminalConnection describes the specific relationship between a physical Terminal endpoint with respect to an Address on a Call. TerminalConnections provide a *physical* view of a Call. For a particular Address endpoint on a Call, there may be zero or more Terminals at which the Call terminates. The TerminalConnection describes each specific Terminal on the Call associated with a particular Address endpoint on the Call. Many simple applications may not care about which specific Terminals are on the Call at a particular Address endpoint. In these cases, the logical view provided by Connections are sufficient.

Requirements for TerminalConnections

In order for a Terminal to be on a Call and associated with a Connection, the Terminal must be associated with the Address object endpoint of the Connection. That is, for each TerminalConnection on a Connection, the Connection's Address must be associated with the TerminalConnection's Terminal. The following predicates illustrates this necessary relationship:

1. Let address = connection.getAddress();
2. Let tc[] = connection.getTerminalConnections();
3. For all i in tc[], let terminal[i] = tc[i].getTerminal();
4. Assert for all i: address is an element of terminal[i].getAddresses();
5. Assert for all i: terminal[i] is an element of address.getTerminals();

TerminalConnection States

The TerminalConnection has a *state* which describes the current relationship between a Terminal and a Connection. TerminalConnection states are distinct from Connection states. Connection states describe the relationship between an entire Address endpoint and a Call, whereas the TerminalConnection state describes the relationship between one of the Terminals at the endpoint Address on the Call with respect to its Connection. Different Terminals on a Call which are associated with the same Connection may be in different states. Furthermore, the state of the TerminalConnection has a dependency and specific relationship to the state of its Connection, as described later on.

The TerminalConnection interface in the core package has six states defined in real-world terms below:

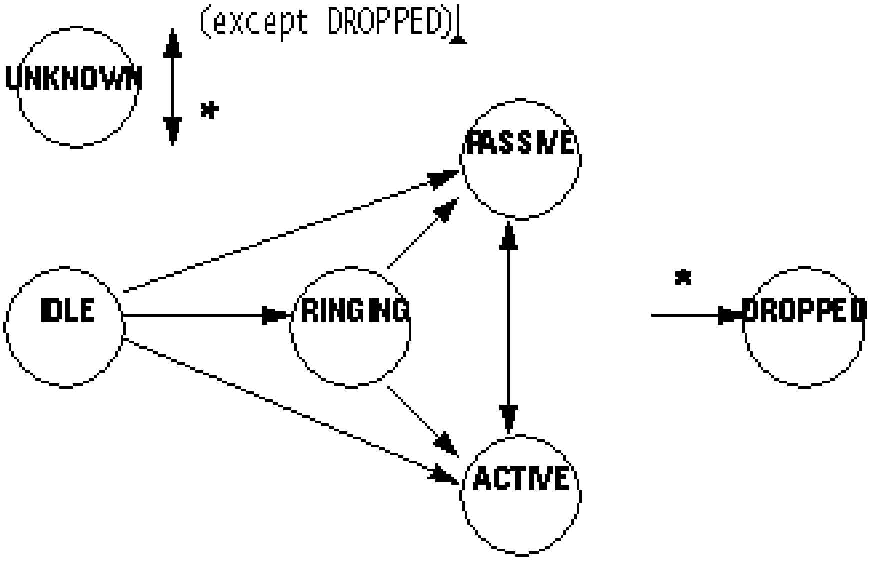
`TerminalConnection.IDLE`	This state is the initial state for all TerminalConnections. TerminalConnection objects do not stay in this state for long. They typically transition into another state quickly.
`TerminalConnection.RINGING`	This state indicates the a Terminal is ringing, indicating that the Terminal has an incoming Call.
`TerminalConnection.PASSIVE`	This state indicates that a Terminal is part of a telephone call but not in an active fashion. This may imply that a resource of the Terminal is being used and may limit actions on the Terminal.
`TerminalConnection.ACTIVE`	This state indicates that a Terminal is actively part of a telephone call. This usually implies that the party speaking on that Terminal is part of the telephone call.
`TerminalConnection.DROPPED`	This state indicates that a particular Terminal has permanently left the telephone call.
`TerminalConnection.UNKNOWN`	This state indicates that the implementation is unable to determine the state of the TerminalConnection. TerminalConnections may transition into and out of this state at any time.

When a TerminalConnection moves into the `TerminalConnection.DROPPED` state, it is no longer associated with its Connection and Terminal. That is, both of these objects lose their references to the TerminalConnection. However, the TerminalConnection still maintains its references to the Connection and Terminal object for application reference. That is, when a TerminalConnection moves into the

TerminalConnection.DROPPED state, the methods TerminalConnection.getConnection() and TerminalConnection.getTerminal() still return valid objects.

TerminalConnection state transitions

Similar to the Connection, there is a finite-state diagram which describes the allowable state transitions of a TerminalConnection. The implementation must guarantee these state transitions. The specifications of methods which affect the state of the TerminalConnections also obey these state transitions. This state diagram is below:



Note the TerminalConnection may transition into the TerminalConnection.DROPPED state from any state, and into and out of the TerminalConnection.UNKNOWN state from any state.

Relationship between Connections and TerminalConnections

As mentioned previously, the state of the Connection determines the following about TerminalConnections:

- Whether TerminalConnections may exist on a Connection.
- The allowable states of the TerminalConnections if they exist.

These properties about Connections and TerminalConnections are guaranteed by the implementation. This relationship is further illustrated in the description of such methods as Call.connect(), Connection.disconnected(), and TerminalConnection.answer(). The following chart defines the specific relationship between Connection states and TerminalConnections.

If the Connection is in state...	*... then the TerminalConnection is*
Connection.IDLE	No TerminalConnections may exist on this Connection, that is, the Connection.getTerminalConnections() method returns null.
Connection.INPROGRESS	No TerminalConnections may exist on this Connection, that is, the Connection.getTerminalConnections() method returns null.
Connection.ALERTING	Zero or more TerminalConnections may exist on this Connection, and each must be in the TerminalConnection.RINGING state.
Connection.CONNECTED	Zero or more TerminalConnections may exist on this Connection, and each must be in the TerminalConnection.PASSIVE or the TerminalConnection.ACTIVE state. Note that when TerminalConnections must into the TerminalConnection.DROPPED state they are no longer associated with the Connection.

`Connection.DISCONNECTED`	No TerminalConnections may exist on this Connection, that is, the `Connection.getTerminalConnections()` method returns null. Note that all TerminalConnections previously associated with this Connection will move into the `TerminalConnection.DROPPED` state.
`Connection.FAILED`	No TerminalConnections may exist on this Connection, that is, the `Connection.getTerminalConnections()` method returns null. Note that all TerminalConnections previously associated with this Connection will move into the `TerminalConnection.DROPPED` state.
`Connection.UNKNOWN`	Zero or more TerminalConnections may exist on this Connection, and each must be in the `TerminalConnection.UNKNOWN` state.

The TerminalConnection.answer() Method

The primary method supported on the core package's TerminalConnection interface is the `TerminalConnection.answer()` method. This method answers a telephone call at a Terminal. This method moves the TerminalConnection into the `TerminalConnection.ACTIVE` state upon success. The TerminalConnection must be in the `TerminalConnection.RINGING` state when this method is invoked.

Observers and Events

All events pertaining to the TerminalConnection object are reported via the `CallObserver` interface on the Call object associated with this TerminalConnection. In the core package, events are reported to a CallObserver when a new TerminalConnection is created and whenever a TerminalConnection changes state. Observers are added to Call objects via the `Call.addObserver()` method and more indirectly via the `Address.addCallObserver()` and `Terminal.addCallObserver()` methods. See the specifications for the Call, Address, and Terminal interfaces for more information.

The following TerminalConnection-related events are defined in the core package. Each of these events extend the `TermConnEv` interface (which, in turn, extends the `CallEv` interface).

`TermConnCreatedEv`	Indicates a new TerminalConnection has been created on a Connection.
`TermConnRingingEv`	Indicates the TerminalConnection has moved into the `TerminalConnection.RINGING` state.
`TermConnActiveEv`	Indicates the TerminalConnection has moved into the `TerminalConnection.ACTIVE` state.
`TermConnPassiveEv`	Indicates the TerminalConnection has moved into the `TerminalConnection.PASSIVE` state.
`TermConnDroppedEv`	Indicates the TerminalConnection has moved into the `TerminalConnection.DROPPED` state.
`TermConnUnknownEv`	Indicates the TerminalConnection has moved into the `TerminalConnection.UNKNOWN` state.

See Also:

CallObserver, TerminalObserver, TermConnEv, CallEv, TermConnRingingEv, TermConnActiveEv, TermConnPassiveEv, TermConnDroppedEv, TermConnUnknownEv

---

Variable Index

- **ACTIVE**

  The `TerminalConnection.ACTIVE` state indicates that a Terminal is actively part of a telephone call.
- **DROPPED**

  The `TerminalConnection.DROPPED` state indicates that a particular Terminal has permanently left the telephone call.
- **IDLE**

  The `TerminalConnection.IDLE` state is the initial state for all TerminalConnection objects.
- **PASSIVE**

  The `TerminalConnection.PASSIVE` state indicates that a Terminal is part of a telephone call but not in an active fashion.
- **RINGING**

  The `TerminalConnection.RINGING` state indicates the a Terminal is ringing, indicating that the Terminal has an incoming Call.
- **UNKNOWN**

The `TerminalConnection.UNKNOWN` state indicates that the implementation is unable to determine the state of the TerminalConnection.

Method Index

- **answer**()

  Answers an incoming telephone call on this TerminalConnection.
- **getCapabilities**()

  Returns the dynamic capabilities for the instance of the TerminalConnection object.
- **getConnection**()

  Returns the Connection object associated with this TerminalConnection.
- **getState**()

  Returns the state of the TerminalConnection object.
- **getTerminal**()

  Returns the Terminal associated with this TerminalConnection object.
- **getTerminalConnectionCapabilities**(Terminal, Address)

  Gets the TerminalConnectionCapabilities object with respect to a Terminal and an Address. **Deprecated.**

Variables

IDLE

```
public static final int IDLE
```

The `TerminalConnection.IDLE` state is the initial state for all TerminalConnection objects.

RINGING

```
public static final int RINGING
```

The `TerminalConnection.RINGING` state indicates the a Terminal is ringing, indicating that the Terminal has an incoming Call.

PASSIVE

```
public static final int PASSIVE
```

The `TerminalConnection.PASSIVE` state indicates that a Terminal is part of a telephone call but not in an active fashion. This may imply that a resource of the Terminal is being used and may limit actions on the Terminal.

ACTIVE

```
public static final int ACTIVE
```

The `TerminalConnection.ACTIVE` state indicates that a Terminal is actively part of a telephone call. This usually implies that the party speaking on that Terminal is party of the telephone call.

DROPPED

```
public static final int DROPPED
```

The `TerminalConnection.DROPPED` state indicates that a particular Terminal has permanently left the telephone call.

UNKNOWN

```
public static final int UNKNOWN
```

The `TerminalConnection.UNKNOWN` state indicates that the implementation is unable to determine the state of the TerminalConnection.

Methods

getState

```
public abstract int getState()
```

Returns the state of the TerminalConnection object.

Returns:

The current state of the TerminalConnection object.

getTerminal

```
public abstract Terminal getTerminal()
```

Returns the Terminal associated with this TerminalConnection object. A TerminalConnection's reference to its Terminal remains valid for the lifetime of the object, even if the Terminal loses its references to this TerminalConnection object. Also, this reference does not change once the TerminalConnection object has been created.

Returns:

The Terminal object associated with this TerminalConnection.

getConnection

```
public abstract Connection getConnection()
```

Returns the Connection object associated with this TerminalConnection. A TerminalConnection's reference to the Connection remains valid throughout the lifetime of the TerminalConnection. Also, this reference does not change once the TerminalConnection object has been created.

Returns:

The Connections associated with this TerminalConnection.

answer

```
public abstract void answer() throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException, InvalidStateException
```

Answers an incoming telephone call on this TerminalConnection. This method waits (i.e. the invoking thread blocks) until the telephone call has been answered at the endpoint before returning. When this method returns successfully, the state of this TerminalConnection object is `TerminalConnection.ACTIVE`.

Allowable TerminalConnection States

The TerminalConnection must be in the `TerminalConnection.RINGING` state when this method is invoked. According to the specification of the TerminalConnection object, this state implies the associated Connection object is also in the `Connection.ALERTING` state. There may be more than one TerminalConnection on the Connection which are in the `TerminalConnection.RINGING` state. In fact, if the Connection is in the `Connection.ALERTING` state, all of these TerminalConnections must be in the `TerminalConnection.RINGING` state. Any of these TerminalConnections may invoke this method to answer the telephone call.

Multiple TerminalConnections

The underlying telephone hardware determines the resulting state of other TerminalConnection objects after the telephone call has been answered by one of the TerminalConnections. The other TerminalConnection object may either move into the `TerminalConnection.PASSIVE` state or the `TerminalConnection.DROPPED` state. If a TerminalConnection moves into the

`TerminalConnection.PASSIVE` state, it remains part of the telephone call, but not actively so. It may have the ability to join the call in the future. If a TerminalConnection moves into the `TerminalConnection.DROPPED` state, it is removed from the telephone call and will never have the ability to join the call in the future. The appropriate events are delivered to the application indicates into which of these two states the other TerminalConnection objects have moved.

Events

The following events are reported to applications via the CallObserver interface as a result of the successful outcome of this method:

1. TermConnActiveEv for the TerminalConnection which invoked this method.
2. ConnConnectedEv for the Connection associated with the TerminalConnection.
3. TermConnPassiveEv or TermConnActiveEv for other TerminalConnections associated with the Connection.

Pre-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == TerminalConnection.RINGING
3. (this.getConnection()).getState() == Connection.ALERTING

Post-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == TerminalConnection.ACTIVE
3. (this.getConnection()).getState() == Connection.CONNECTED
4. TermConnActiveEv for the TerminalConnection which invoked this method.
5. ConnConnectedEv for the Connection associated with the TerminalConnection.
6. TermConnPassiveEv or TermConnActiveEv for other TerminalConnections associated with the Connection.

Throws: PrivilegeViolationException

The application did not have proper authority to answer the telephone call. For example, the Terminal associated with the TerminalConnection may not be in the Provider's local domain.

Throws: ResourceUnavailableException

The necessary resources to answer the telephone call were not available when the method was invoked.

Throws: MethodNotSupportedException

This method is currently not supported by this implementation.

Throws: InvalidStateException

An object was not in the proper state, violating the pre-conditions of this method. For example, the Provider was not in the Provider.IN_SERVICE state or the TerminalConnection was not in the TerminalConnection.RINGING state.

See Also:

TermConnActiveEv, TermConnPassiveEv, TermConnDroppedEv, ConnConnectedEv

getCapabilities

```
public abstract TerminalConnectionCapabilities getCapabilities()
```

Returns the dynamic capabilities for the instance of the TerminalConnection object. Dynamic capabilities tell the application which actions are possible at the time this method is invoked based upon the implementations knowledge of its ability to successfully perform the action. This determination may be based upon argument passed to this method, the current state of the call model, or some implementation- specific knowledge. These indications do not guarantee that a particular method will be successful when invoked, however.

The dynamic TerminalConnection capabilities require no additional arguments.

Returns:

The dynamic TerminalConnection capabilities.

getTerminalConnectionCapabilities

```
public abstract TerminalConnectionCapabilities
```

```
getTerminalConnectionCapabilities(Terminal terminal,

Address address) throws InvalidArgumentException, PlatformException
```

Note: getTerminalConnectionCapabilities() is deprecated. *Since JTAPI v1.2. This method has been replaced by the TerminalConnection.getCapabilities() method.*

Gets the TerminalConnectionCapabilities object with respect to a Terminal and an Address. If null is passed as a Terminal parameter, the general/ provider-wide Terminal Connection capabilities are returned.

Note: This method has been replaced in JTAPI v1.2. The `TerminalConnection.getCapabilities()` method returns the dynamic TerminalConnection capabilities. This method now should simply invoke the `TerminalConnection.getCapabilities()` method.

Parameters:

address - This argument is ignored in JTAPI v1.2 and later.

terminal - This argument is ignored in JTAPI v1.2 and later.

Throws: InvalidArgumentException

This exception is never thrown in JTAPI v1.2 and later.

Throws: PlatformException

A platform-specific exception occurred.

Interface javax.telephony.TerminalObserver

public interface **TerminalObserver**

Introduction

The `TerminalObserver` interface reports all changes which happen to the Terminal object. These changes are reported as events to the `TerminalObserver.terminalChangedEvent()` method. Applications must instantiate an object which implements this interface and then use the `Terminal.addObserver()`) method to register the object to receive all future events associated with the Terminal object.

The `TerminalObserver.terminalChangedEvent()` method receives an array of events which all must extend the `TermEv` interface. Since several changes may happen to a single JTAPI object at once, a list of events is needed to convey those changes which happen at the same time. Applications iterate through the array of events provided.

Terminal Observation Ending

At various times, the underlying implementation may not be able to observe the Terminal. In these instances, the TerminalObserver is sent an TermObservationEndedEv event. This indicates that the application will not receive further events associated with the Terminal object. This observer is no longer reported via the `Terminal.getObservers()` method.

See Also:

TermEv, TermObservationEndedEv

Method Index

● **terminalChangedEvent**(TermEv[])

Reports all events associated with the Terminal object.

Methods

● **terminalChangedEvent**

```
public abstract void terminalChangedEvent(TermEv eventList[])
```

Reports all events associated with the Terminal object. This method passes an array of TermEv objects as its arguments which correspond to the list of events representing the changes to the Terminal object.

Parameters:

eventList - The list of Terminal events.

Class javax.telephony.JtapiPeerFactory

```
java.lang.Object
   |
   +----javax.telephony.JtapiPeerFactory
```

public class **JtapiPeerFactory**

extends Object

Introduction

The `JtapiPeerFactory` class is a class by which applications obtain a Provider object. Applications use this class to first obtain a class which implements the `JtapiPeer` interface. The `JtapiPeer` interface represents a particular vendor's implementation of JTAPI. The term 'peer' is Java nomenclature for "a particular platform-specific implementation of a Java interface or API". This term has the same meaning for the Java Telephony API. Applications are not permitted to create an instance of the `JtapiPeerFactory` class. Through an installation procedure provided by each implementator, a `JtapiPeer` class is made available to an application environment. When applications have a JtapiPeer object for a particular platform-dependent implementation, they may obtain a Provider object via that interface. The details of that interface are discussed in the specification for the `JtapiPeer` interface.

Obtaining a JtapiPeer Object

Applications use the `JtapiPeerFactory.getJtapiPeer()` method to obtain a JtapiPeer object. The argument to this method is a classname which represents an object which implements the `JtapiPeer` interface. This object and the classname under which it can be found must be supplied by the vendor of the implementation. Note that this object is not a Provider, however, this interface is used to obtain Provider objects from that particular implementation.

The Java Telephony API places conventions on vendors on the classname they use for their JtapiPeer object. This class name *must* begin with the domain name assigned to the vendor in reverse order. Because the space of domain names is managed, this scheme ensures that collisions between two different vendor's implementations will not happen. For example, an implementation from Sun Microsystem's will have "com.sun" as the prefix to its JtapiPeer class. After the reversed domain name, vendors are free to choose any class hierarchy they desire.

Default JtapiPeer

Additionally, the vendor providing the JtapiPeer class may supply a a `DefaultJtapiPeer.class` class file. When placed in the classpath of applications, this class (which must implement the `JtapiPeer` interface) becomes the default JtapiPeer object returned by the `JtapiPeerFactory.getJtapiPeer()` method. By convention the default class name must be `DefaultJtapiPeer`.

In basic environments, applications and users do not want the burden of finding out the class name in order to use a particular implementation. Therefore, the `JtapiPeerFactory` class supports a mechanism for applications to obtain the default implementation for their system. If applications use a `null` argument to the `JtapiPeerFactory.getJtapiPeer()` method, they will be returned the default installed implementation on their system if it exists.

Note: It is the responsibility of implementation vendors to supply a version of a `DefaultJtapiPeer` or some means to alias their peer implementation along with a means to place that `DefaultJtapiPeer` class in the application classpath.

See Also:

JtapiPeer

Method Index

getJtapiPeer(String)

Returns an instance of a JtapiPeer object given a fully qualified classname of the class which implements the JtapiPeer object.

Methods

getJtapiPeer

```
 public static synchronized JtapiPeer getJtapiPeer(String jtapiPeerName) throws
JtapiPeerUnavailableException
```

Returns an instance of a JtapiPeer object given a fully qualified classname of the class which implements the JtapiPeer object.

If no classname is provided (null), a default class named `DefaultJtapiPeer` is chosen as the classname to load. If it does not exist or is not installed in the CLASSPATH as the default, a JtapiPeerUnavailableException exception is thrown.

Parameters:

jtapiPeerName - The classname of the JtapiPeer object class.

Returns:

An instance of the JtapiPeer object.

Throws: JtapiPeerUnavailableException

Indicates that the JtapiPeer specified by the classname is not available.

All Packages   This Package   Previous   Next

Class javax.telephony.InvalidArgumentException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.InvalidArgumentException
```

public class **InvalidArgumentException**

extends Exception

An InvalidArgumentException indicates an argument passed to the method is invalid.

Constructor Index

- **InvalidArgumentException**()

  Constructor with no string.
- **InvalidArgumentException**(String)

  Constructor which takes a string description.

Constructors

InvalidArgumentException

```
 public InvalidArgumentException()
```

Constructor with no string.

InvalidArgumentException

```
 public InvalidArgumentException(String s)
```

Constructor which takes a string description.

Class javax.telephony.InvalidPartyException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.InvalidPartyException
```

public class **InvalidPartyException**

extends Exception

An InvalidPartyException indicates that a party given as an argument to the method call was invalid. This may either be the originating party of a telephone call or the destination party of a telephone call.

Variable Index

- **DESTINATION_PARTY**

  Indicates that the destination party was invalid.
- **ORIGINATING_PARTY**

  Indicates that the originating party was invalid.
- **UNKNOWN_PARTY**

  Indicates that the party was unknown.

Constructor Index

- **InvalidPartyException**(int)

  Constructor with no string.
- **InvalidPartyException**(int, String)

  Constructor which takes a string description.

Method Index

- **getType**()

  Returns the type of party.

Variables

ORIGINATING_PARTY

```
public static final int ORIGINATING_PARTY
```

Indicates that the originating party was invalid.

DESTINATION_PARTY

```
public static final int DESTINATION_PARTY
```

Indicates that the destination party was invalid.

UNKNOWN_PARTY

```
public static final int UNKNOWN_PARTY
```

Indicates that the party was unknown.

Constructors

InvalidPartyException

```
public InvalidPartyException(int type)
```

Constructor with no string.

InvalidPartyException

```
public InvalidPartyException(int type,
                             String s)
```

Constructor which takes a string description.

Methods

getType

```
public int getType()
```

Returns the type of party.

Returns:

The type of party.

All Packages   This Package   Previous   Next

Class javax.telephony.InvalidStateException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.InvalidStateException
```

public class **InvalidStateException**

extends Exception

An InvalidStateException indicates the current state of an object involved in the method invocation does not meet the acceptable pre-conditions for the method. Each method which changes the call model typically has a set of states in which the object must be as a pre-condition for the method. Each method documents the pre-condition states for objects. Typically, this method will succeed in the future once the object in question has reached the proper state.

This exception provides the application with the object in question and the state it is currently in.

Variable Index

- **ADDRESS_OBJECT**

  The invalid object in question is the Address
- **CALL_OBJECT**

  The invalid object in question is the Call
- **CONNECTION_OBJECT**

  The invalid object in question is the Connection
- **PROVIDER_OBJECT**

  The invalid object in question is the Provider
- **TERMINAL_CONNECTION_OBJECT**

  The invalid object in question is the Terminal Connection
- **TERMINAL_OBJECT**

  The invalid object in question is the Terminal

Constructor Index

- **InvalidStateException**(Object, int, int)

  Constructor with no string.
- **InvalidStateException**(Object, int, int, String)

  Constructor which takes a string description.

Method Index

- **getObject**()

  Returns the object which has the incorrect state.

- **getObjectType**()

  Returns the type of object in question.

- **getState**()

  Returns the state of the object.

Variables

PROVIDER_OBJECT

```
public static final int PROVIDER_OBJECT
```

The invalid object in question is the Provider

CALL_OBJECT

```
public static final int CALL_OBJECT
```

The invalid object in question is the Call

CONNECTION_OBJECT

```
public static final int CONNECTION_OBJECT
```

The invalid object in question is the Connection

TERMINAL_OBJECT

```
public static final int TERMINAL_OBJECT
```

The invalid object in question is the Terminal

ADDRESS_OBJECT

```
public static final int ADDRESS_OBJECT
```

The invalid object in question is the Address

TERMINAL_CONNECTION_OBJECT

```
public static final int TERMINAL_CONNECTION_OBJECT
```

The invalid object in question is the Terminal Connection

Constructors

InvalidStateException

```
public InvalidStateException(Object object,
                                 int type,
                                 int state)
```

Constructor with no string.

InvalidStateException

```
public InvalidStateException(Object object,
                             int type,
                             int state,
                             String s)
```

Constructor which takes a string description.

Methods

getObjectType

```
public int getObjectType()
```

Returns the type of object in question.

Returns:

The type of object in question.

getObject

```
public Object getObject()
```

Returns the object which has the incorrect state.

Returns:

The object which is in the wrong state.

getState

```
public int getState()
```

Returns the state of the object.

Returns:

The state of the object.

Class javax.telephony.JtapiPeerUnavailableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.JtapiPeerUnavailableException
```

public class **JtapiPeerUnavailableException**

extends Exception

The JtapiPeerUnavailableException indicates that the JtapiPeer (i.e. a particular implementation of JTAPI is unavailable on the current system.

Constructor Index

- **JtapiPeerUnavailableException**()

  Constructor with no string.
- **JtapiPeerUnavailableException**(String)

  Constructor which takes a string description.

Constructors

JtapiPeerUnavailableException

```
 public JtapiPeerUnavailableException()
```

Constructor with no string.

JtapiPeerUnavailableException

```
 public JtapiPeerUnavailableException(String s)
```

Constructor which takes a string description.

Class javax.telephony.MethodNotSupportedException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.MethodNotSupportedException
```

public class **MethodNotSupportedException**

extends Exception

The MethodNotSupportedException indicates that the method which was invoked is not supported by the implementation.

Constructor Index

- **MethodNotSupportedException**()

  Constructor with no string.

- **MethodNotSupportedException**(String)

  Constructor which takes a string description.

Constructors

MethodNotSupportedException

```
public MethodNotSupportedException()
```

Constructor with no string.

MethodNotSupportedException

```
public MethodNotSupportedException(String s)
```

Constructor which takes a string description.

All Packages This Package Previous Next

Class javax.telephony.PlatformException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----javax.telephony.PlatformException
```

public class **PlatformException**

extends RuntimeException

A PlatformException indicates an implementation-specific exception. The specific exceptions which implementations throw is documented in their release notes.

JTAPI v1.1.1 NOTE: PlatformException extends Java's RuntimeException. This permits it to be thrown from a JTAPI method without being declared in its signature. Note that no JTAPI methods declare PlatformException to be thrown. This is a change from v1.1, but does not affect applications.

Since PlatformException typically denotes some form of unrecoverable platform-dependent error, invoking the method again typically does not yield success. These types of exceptions are often best dealt with at a higher level, in a top-level "try-catch" block where the entire application could be restarted.

Constructor Index

- **PlatformException**()

  Constructor with no string.
- **PlatformException**(String)

  Constructor which takes a string description.

Constructors

PlatformException

```
public PlatformException()
```

Constructor with no string.

PlatformException

```
public PlatformException(String s)
```

Constructor which takes a string description.

Class javax.telephony.PrivilegeViolationException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.PrivilegeViolationException
```

public class **PrivilegeViolationException**

extends Exception

A PrivilegeViolationException indicates that an action pertaining to a certain object failed because the application did not have the proper security permissions to execute that command.

This class stores the type of privilege not available which is obtained via the `PrivilegeViolationException.getType()` method.

Variable Index

- **DESTINATION_VIOLATION**

  A privilege violation occurred on the destination.
- **ORIGINATOR_VIOLATION**

  A privilege violation occurred on the originator.
- **UNKNOWN_VIOLATION**

  A privilege violation occurred at an unknown place.

Constructor Index

- **PrivilegeViolationException**(int)

  Constructor, takes a type but no string.
- **PrivilegeViolationException**(int, String)

  Constructor, takes a type and a string.

Method Index

- **getType**()

  Returns the type of privilege which is not available.

Variables

ORIGINATOR_VIOLATION

```
public static final int ORIGINATOR_VIOLATION
```

A privilege violation occurred on the originator.

DESTINATION_VIOLATION

```
public static final int DESTINATION_VIOLATION
```

A privilege violation occurred on the destination.

UNKNOWN_VIOLATION

```
public static final int UNKNOWN_VIOLATION
```

A privilege violation occurred at an unknown place.

Constructors

PrivilegeViolationException

```
public PrivilegeViolationException(int type)
```

Constructor, takes a type but no string.

PrivilegeViolationException

```
public PrivilegeViolationException(int type,
                                   String s)
```

Constructor, takes a type and a string.

Methods

getType

```
public int getType()
```

Returns the type of privilege which is not available.

Returns:

The type of privilege.

All Packages This Package Previous Next

Class javax.telephony.ProviderUnavailableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----javax.telephony.ProviderUnavailableException
```

public class **ProviderUnavailableException**

extends RuntimeException

The ProviderUnavailableException indicates that the Provider is currently not available to the application. This exception extends Java's RuntimeException, and therefore can be thrown on any JTAPI method. It is typically thrown in two situations: when `JtapiPeer.getProvider()` is called or on any method when the Provider is in a `Provider.SHUTDOWN` state. Because this method extends `RuntimeException`, it can be thrown from any method without being declared.

This exception is thrown on `JtapiPeer.getProvider()` when the requested Provider is not available to the application for a number of reasons, including when an invalid service string or optional argument was given. If this exception is thrown on a random JTAPI method, it indicates that the method call is invalid because the Provider is not in the "in service" state.

This exception stores the reason for the failure which may be obtained via the `ProviderUnavailableException.getCause()` method.

Variable Index

- **CAUSE_INVALID_ARGUMENT**

  Constant definition for an invalid optional argument given to `JtapiPeer.getProvider()`.
- **CAUSE_INVALID_SERVICE**

  Constant definition for an invalid service string given to `JtapiPeer.getProvider()`.
- **CAUSE_NOT_IN_SERVICE**

  Constant definition for the Provider not in the "in service" state.
- **CAUSE_UNKNOWN**

  Constant definition for an unknown cause.

Constructor Index

- **ProviderUnavailableException**()

  Constructor with no cause and string.
- **ProviderUnavailableException**(int)

  Constructor which takes a cause string.

- **ProviderUnavailableException**(int, String)

  Constructor which takes both a string and a cause.
- **ProviderUnavailableException**(String)

  Constructor which takes a string description.

Method Index

- **getCause**()

  Returns the cause for this exception.

Variables

CAUSE_UNKNOWN

```
public static final int CAUSE_UNKNOWN
```

Constant definition for an unknown cause.

CAUSE_NOT_IN_SERVICE

```
public static final int CAUSE_NOT_IN_SERVICE
```

Constant definition for the Provider not in the "in service" state.

CAUSE_INVALID_SERVICE

```
public static final int CAUSE_INVALID_SERVICE
```

Constant definition for an invalid service string given to `JtapiPeer.getProvider()`.

CAUSE_INVALID_ARGUMENT

```
public static final int CAUSE_INVALID_ARGUMENT
```

Constant definition for an invalid optional argument given to `JtapiPeer.getProvider()`.

Constructors

ProviderUnavailableException

```
public ProviderUnavailableException()
```

Constructor with no cause and string.

ProviderUnavailableException

```
public ProviderUnavailableException(int cause)
```

Constructor which takes a cause string.

ProviderUnavailableException

```
public ProviderUnavailableException(String s)
```

Constructor which takes a string description.

ProviderUnavailableException

```
public ProviderUnavailableException(int cause,
                                    String s)
```

Constructor which takes both a string and a cause.

Methods

getCause

```
public int getCause()
```

Returns the cause for this exception.

Returns:

The cause of this exception.

All Packages   This Package Previous Next

Class javax.telephony.ResourceUnavailableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.telephony.ResourceUnavailableException
```

public class **ResourceUnavailableException**

extends Exception

The ResourceUnavailableException indicates that a resource inside the system in not available to complete an operation. The type embodied in this exception further clarifies what is not available and is obtained via the `ResourceUnavailableException.getType()` method.

Variable Index

- **NO_DIALTONE**

  No dialtone detected.
- **OBSERVER_LIMIT_EXCEEDED**

  The number of observers existing already reached the limit.
- **ORIGINATOR_UNAVAILABLE**

  The originating device was not available for this action.
- **OUTSTANDING_METHOD_EXCEEDED**

  The internal resources to handle another method have been exceeded.
- **TRUNK_LIMIT_EXCEEDED**

  The number of trunks which are currently in use has been exceeded.
- **UNKNOWN**

  Indicates the specific reason is unspecified.
- **UNSPECIFIED_LIMIT_EXCEEDED**

  An internal resource, unspecified by the implementation, has been exceeded.
- **USER_RESPONSE**

  A user has not responded in the time allowed by an implementation.

Constructor Index

- **ResourceUnavailableException**(int)

  Constructor, takes a type but no string.

- **ResourceUnavailableException**(int, String)

  Constructor, takes a type and a string.

Method Index

- **getType**()

  Returns the type of resource which was unavailable.

Variables

UNKNOWN

```
public static final int UNKNOWN
```

Indicates the specific reason is unspecified.

ORIGINATOR_UNAVAILABLE

```
public static final int ORIGINATOR_UNAVAILABLE
```

The originating device was not available for this action.

OBSERVER_LIMIT_EXCEEDED

```
public static final int OBSERVER_LIMIT_EXCEEDED
```

The number of observers existing already reached the limit.

TRUNK_LIMIT_EXCEEDED

```
public static final int TRUNK_LIMIT_EXCEEDED
```

The number of trunks which are currently in use has been exceeded.

OUTSTANDING_METHOD_EXCEEDED

```
public static final int OUTSTANDING_METHOD_EXCEEDED
```

The internal resources to handle another method have been exceeded.

UNSPECIFIED_LIMIT_EXCEEDED

```
public static final int UNSPECIFIED_LIMIT_EXCEEDED
```

An internal resource, unspecified by the implementation, has been exceeded.

NO_DIALTONE

```
public static final int NO_DIALTONE
```

No dialtone detected.

USER_RESPONSE

```
public static final int USER_RESPONSE
```

A user has not responded in the time allowed by an implementation.

Constructors

ResourceUnavailableException

```
 public ResourceUnavailableException(int type)
```

Constructor, takes a type but no string.

ResourceUnavailableException

```
 public ResourceUnavailableException(int type,
                                     String s)
```

Constructor, takes a type and a string.

Methods

getType

```
 public int getType()
```

Returns the type of resource which was unavailable.

Returns:

The type of resource unavailable.

All Packages   This Package   Previous   Next

All Packages

package javax.telephony.callcenter

Interface Index

- ACDAddress
- ACDAddressObserver
- ACDConnection
- ACDManagerAddress
- ACDManagerConnection
- Agent
- AgentTerminal
- AgentTerminalObserver
- CallCenterAddress
- CallCenterCall
- CallCenterCallObserver
- CallCenterProvider
- CallCenterTrunk
- RouteAddress
- RouteCallback
- RouteSession

Interface javax.telephony.callcenter.ACDAddress

public interface **ACDAddress**
extends CallCenterAddress

Introduction

Automated Call Distribution (ACD) is a Call Center feature that provides a mechanism for receiving calls, queueing them, and distributing them to agent extensions within ACD Groups. An ACD Group comprises zero or more agent extensions, which are dynamically associated with ACD Groups through a login/logout process. The `ACDAddress` interface models an ACD Group for ACD systems.

A call placed to an `ACDAddress` represents a call which is being routed to an available agent logged into the ACD Group. In the case where no agent is available, the `ACDAddress` is queued for a group of agents who are logged in to that ACD Group, but unavailable to service that call. Calls are distributed to the agents in that group based on their availability and other factors determined by the implementation.

Calls may placed directly to an `ACDAddress` or the ACD machanism can be relied upon to select a destination `ACDAddress` by placing the call to an `ACDManagerAddress`.

The `ACDAddress` extends `CallCenterAddress` with the methods necessary to obtain ACD-specific information such as the Agent objects associated with the `ACDAddress` and a variety of call queue methods to get information on calls queued at this address.

ACDAddresses differ from Addresses

Some important differences between `ACDAddress` and Address are:

1. An `ACDAddress` cannot have `Terminal` objects associated with it.
2. An `ACDAddress` is not a logical endpoint of a call in the same sense as an `Address` Rather, it models a queuing process whereby the selection of a logical endpoint is deferred.
3. `ACDConnections` to an `ACDAddress` do not enter into a CONNECTED state.
4. It is not returned on `Provider.getAddresses()`, but is available through `CallCenterProvider.getACDAddresses()`

ACDAddresses and ACDConnections

A call presented to an `ACDAddress` is modeled by an `ACDConnection`. That `ACDConnection` may be between a `Call` object and the `ACDAddress` or it may exist between a `ACDManagerConnection` and the `ACDAddress`, depending on whether the call was placed directly to the `ACDAddress` or whether it arrived indirectly through the distribution mechanism from an `ACDManagerAddress`.

Observation at an ACDAddress

All events pertaining to the `ACDAddress` interface are reported via the `AddressObserver.addressChangedEvent()` method. In order to observe Agent state changes for `Agents` associated with an `ACDAddress`, an application must implement an `ACDAddressObserver` interface and associate it with the `ACDAddress` using the `addObserver()` method on an `ACDAddress` object.

See Also:

ACDConnection, CallCenterAddress, ACDAddressObserver, Address

Method Index

- **getACDManagerAddress**()

FOR JTAPI 1.2 Returns the `ACDManagerAddesses` (was a single `ACDManagerAddess`) associated at system administration time with this `ACDAddress`.

- **getLoggedOnAgents**()

  Returns the Agents logged into the ACDAddress.

- **getNumberQueued**()

  Returns the number of Calls queued at an ACDAddress.

- **getOldestCallQueued**()

  Rreturns the oldest Call queued to an ACDAddress.

- **getQueueWaitTime**()

  Returns the estimated wait time for new Calls queued at an ACDAddress.

- **getRelativeQueueLoad**()

  Returns the relative load of an ACDAddress queue.

Methods

getLoggedOnAgents

```
public abstract Agent[] getLoggedOnAgents() throws MethodNotSupportedException
```

Returns the Agents logged into the ACDAddress.

Returns:

An array of Agents associated with the ACDAddress.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getNumberQueued

```
public abstract int getNumberQueued() throws MethodNotSupportedException
```

Returns the number of Calls queued at an ACDAddress.

Returns:

The number of calls queued.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getOldestCallQueued

```
public abstract Call getOldestCallQueued() throws MethodNotSupportedException
```

Rreturns the oldest Call queued to an ACDAddress.

Returns:

The oldest Call queued.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getRelativeQueueLoad

```
public abstract int getRelativeQueueLoad() throws MethodNotSupportedException
```

Returns the relative load of an ACDAddress queue.

Returns:

The relative load of the ACDAddress queue.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getQueueWaitTime

```
 public abstract int getQueueWaitTime() throws MethodNotSupportedException
```

Returns the estimated wait time for new Calls queued at an ACDAddress.

Returns:

The estimated wait time for new calls at the ACDAddress.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getACDManagerAddress

```
 public abstract ACDManagerAddress[] getACDManagerAddress() throws
MethodNotSupportedException
```

FOR JTAPI 1.2 Returns the `ACDManagerAddesses` (was a single `ACDManagerAddess`) associated at system administration time with this `ACDAddress`. This method returns a null if no are associated with this `ACDAddress`.

Returns:

The ACDManagerAddress associated with this ACDAddress.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.ACDAddressObserver

public interface **ACDAddressObserver**

extends AddressObserver

The `ACDAddressObserver` interface reports all state changes in the `Agent` that is associated with the `ACDAddress` as events. Applications instantiate an object which implements this interface and use the `Address.addObserver()` to request delivery of events to this observer object. Events will be delivered to this interface only if the Provider is in the `Provider.IN_SERVICE` state. All events which are reported via this interface must extend the `ACDAddrEv` interface.

Events are reported via the `AddressObserver.addressChangedEvent()` method. This interface defines no additional methods and therefore serves as a way applications signal to the implementation that is desires call center package events.

Note that the state changes in the `Agent` are also reported via the `AgentTermianlObserver` for the `AgentTerminal` on which the `Agent` is logged in to.

See Also:

Address, AddressObserver, ACDAddress, AgentTerminalObserver, ACDAddrEv, ACDAddrBusyEv, ACDAddrLoggedOffEv, ACDAddrLoggedOnEv, ACDAddrNotReadyEv, ACDAddrReadyEv, ACDAddrUnknownEv, ACDAddrWorkNotReadyEv, ACDAddrWorkReadyEv

Interface javax.telephony.callcenter.ACDConnection

public interface **ACDConnection**

extends Connection

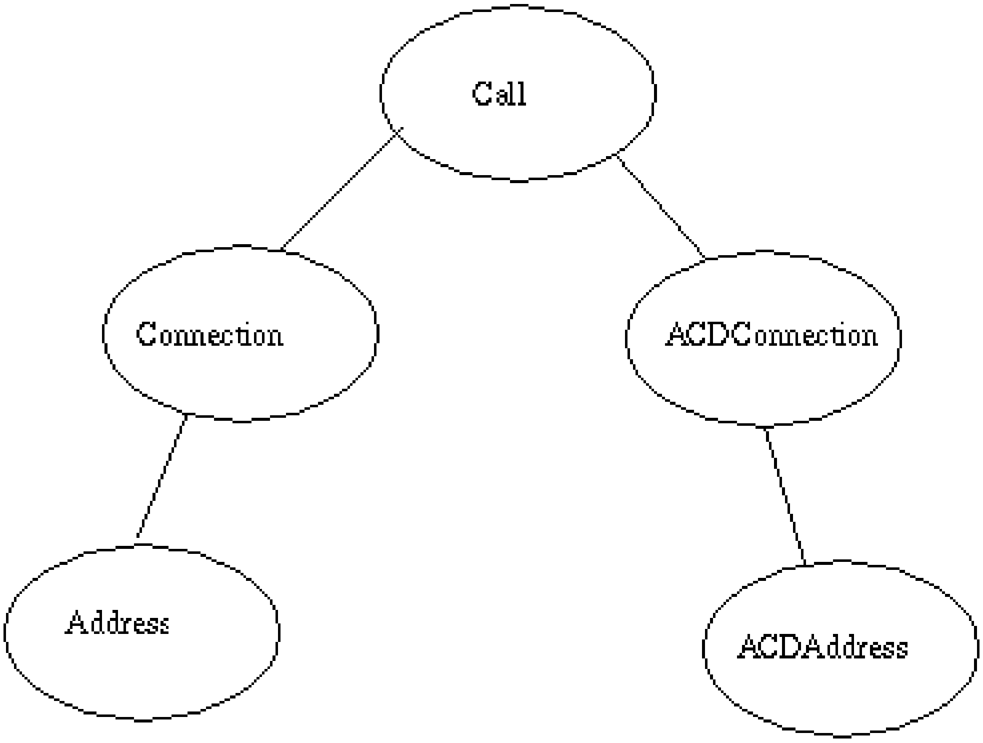
Introduction

An `ACDConnection` models either a direct relationship between a `Call` and an `ACDAddress` or an indirect relationship between a `Call` and an `ACDAddress` through an `ACDManagerAddress`.

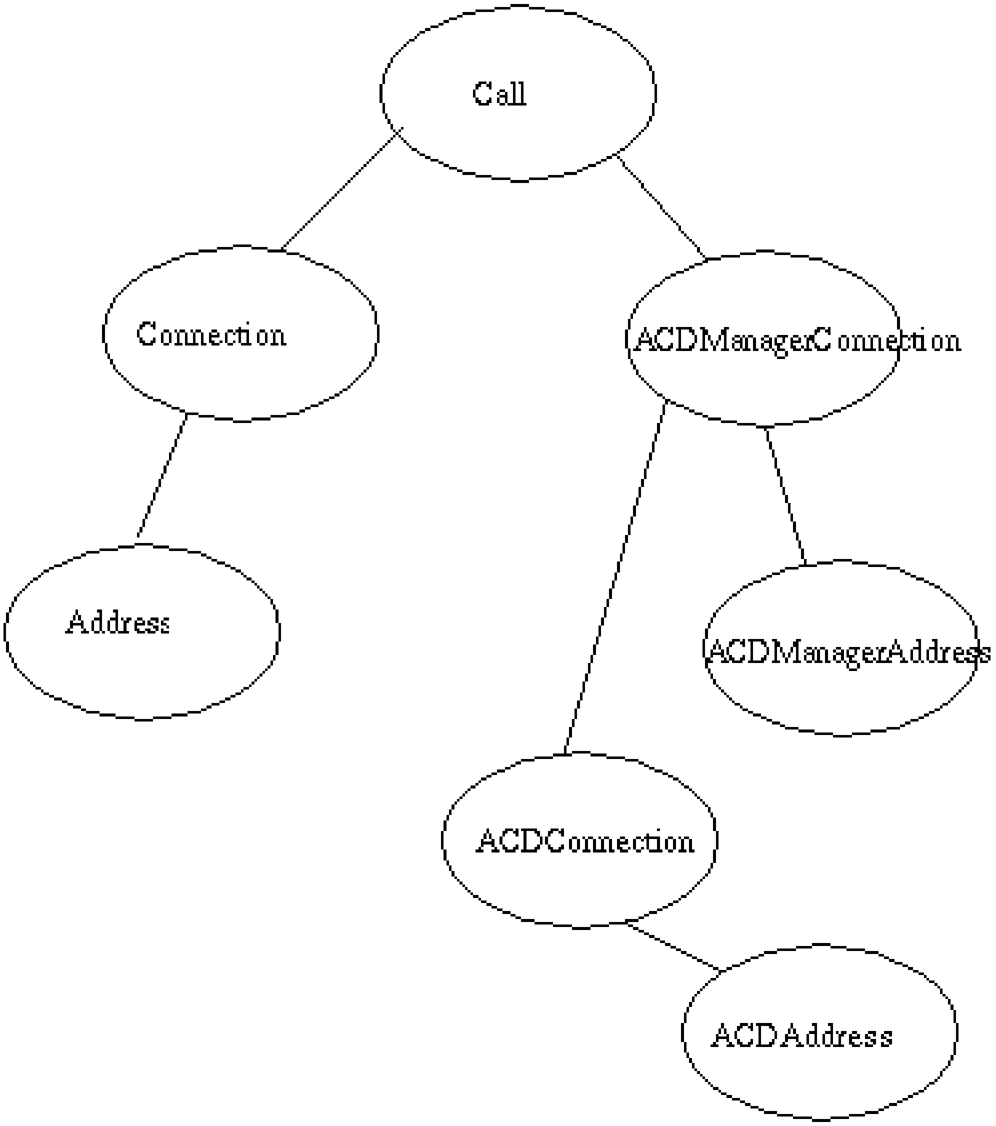
The direct relationship occurs when a `Call` arrives at an `ACDAddress`. The indirect relationship occurs when a `Call` arrives at an `ACDManagerAddress` and the implementation of the `ACDMangerAddress` determines that it must involve an `ACDAddress` in the `Call`.

The `ACDConnection` to an `ACDAddress` in either case, direct or indirect, models a Call that is being routed to an agent logged into the `ACDAddress`, or a call that is being queued for agents logged into the `ACDAddress`

ACDConnection as a Direct Connection



ACDConnection as an Indirect Connection



The `ACDConnection` is not a connection in the same sense as a `Connection`, because it never represents a call to an endpoint. Its purpose is to model a call that is being routed or queued by an ACD system.

ACDConnections and TerminalConnections

The `getTerminalConnection()` method on the core `Connection` interface, which `ACDConnection` extends, will always return `null` because `ACDAddresses` do not have `Terminals` associated with them.

ACDConnection States

The state of the `ACDConnection` is available through the `ACDConnection.getState()` method inherited from the core `Connection` interface. Each state is an integer constant defined in the core `Connection` interface. Their meaning in this interface are summarized below:

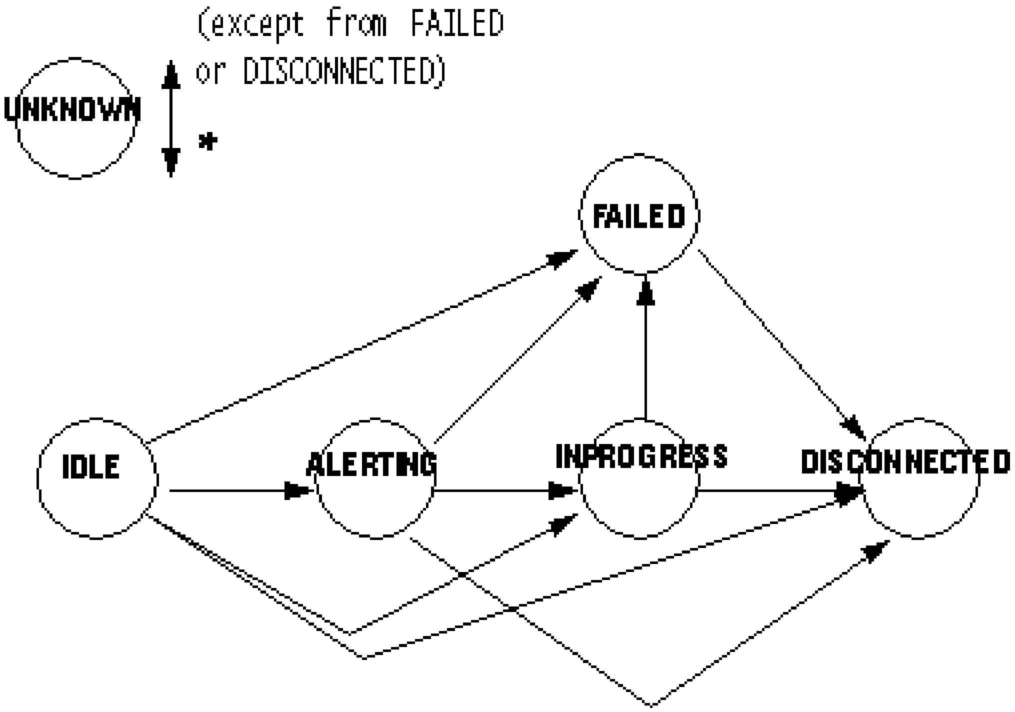
`ACDConnection.IDLE`	As in the core, this is the initial and transitory state for new `ACDConnection` objects.
`ACDConnection.INPROGRESS`	This state indicates that an `ACDConnection` is queued at a particular `ACDAddress`. This will result when there are no agents available to route the call to.

`ACDConnection.ALERTING`	This state indicates that the `ACDConnection` has been made to a particular `ACDAddress`. This state is only valid for `ACDConnections` that are not associated with an `ACDManagerConnection`.
`ACDConnection.DISCONNECTED`	This state has the same definition as in the core.
`ACDConnection.FAILED`	This state has the same definition as in the core.
`ACDConnection.UNKNOWN`	This state has the same definition as in the core.

ACDConnection State Transitions

The `ACDConnection` class defines the allowable `ACDConnection` state transitions. These finite-state transitions must be guaranteed by the implementation. Each method that causes a change in an `ACDConnection` state must be consistent with this state diagram.

Note there is a general left-to-right progression of the state transitions. A Connection object may transition into and out of the `ACDConnection.UNKNOWN` state at any time with the annotated exceptions (hence, the asterisk qualifier next to its bidirectional transition arrow).



See Also:

ACDAddress, ACDManagerAddress, ACDManagerConnection

Method Index

- **getACDManagerConnection**()

  Returns the `ACDManagerConnection` associated with this `ACDConnection`.

Methods

getACDManagerConnection

```
 public abstract ACDManagerConnection getACDManagerConnection() throws
MethodNotSupportedException
```

Returns the `ACDManagerConnection` associated with this `ACDConnection`. A null is returned if this `ACDConnection` is not in an indirect relationship between a Call, an `ACDAddress` and an `ACDManagerAddress`.

Returns:

The ACDManagerConnection associated with this ACDConnection.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

All Packages   This Package Previous Next

Interface javax.telephony.callcenter.ACDManagerAddress

public interface **ACDManagerAddress**
extends CallCenterAddress

Introduction

The `ACDManagerAddress` interface models an ACD management control point that manages one or more Agent Groups. Calls are presented to an `ACDManagerAddress` so that it can distribute those calls to agents logged into the Agent Groups managed by this special address.

ACDManagerAddresses and other ACD Objects

When a call is placed to an `ACDManagerAddress`, its implementation routes that call to one or more Agent Groups, which are modeled by `ACDAddresses`. When an `ACDAddress` receives the call it may present that call to an available Agent in its Agent Group, or it may queue the call until an Agent is available.

The ultimate destination of a call is a human at a phone terminal. This person may have "sessions" active at different Agent Groups, each of which presents her or him as an Agent in that Agent Group. An Agent is the recipient of a call in an Agent Group and Agents are modeled as `Agent` objects. When an Agent is selectd, the call is completed by a `Connection` between the `Call` object that originally involved the `ACDManagerAddress` and the `Address` associated with that `Agent`. The `ACDManagerConnection` and `ACDConnections` that modeled ACD call routing and queuing to that point are then placed in a DISCONNECTED state.

ACDManagerAddresses differ from Addresses

Some important differences between `ACDManagerAddress` and Address are:

1. An `ACDManagerAddress` cannot have any `Terminal` objects associated with it.
2. An `ACDManagerAddress` is not a logical endpoint of a call in the same sense as an `Address`, rather it models a distribution process whereby the selection of a logical endpoint is deferred.
3. `ACDManagerConnections` associated with an `ACDAddress` do not enter into the CONNECTED state.

See Also:

ACDManagerConnection, ACDAddress, ACDManagerConnection

Method Index

getACDAddresses()

Returns the `ACDAddess(es)` associated at system administration time with this `ACDManagerAddress`.

Methods

getACDAddresses

```
public abstract ACDAddress[] getACDAddresses() throws MethodNotSupportedException
```

Returns the `ACDAddess(es)` associated at system administration time with this `ACDManagerAddress`. This method returns a null if no `ACDAddress` is associated with this `ACDManagerAddress`.

It does not return the `ACDAddress(es)` connected to this `ACDManagerAddress` in a `Call`. That information can be obtained through the `getACDConnections()` method on `ACDManagerConnection`.

Returns:

The ACDAddresses associated with this ACDManagerAddress.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.ACDManagerConnection

public interface **ACDManagerConnection**

extends Connection

Introduction

The ACDManagerConnection models a call that is being offered for routing by an ACD. The ACD mechanism selects among Agent Groups as recipients of the call. A call can be offered to one or more Agent Groups. If none of those groups contain an agent that is able to service the call, the call will be queued at each of the groups. As soon as an agent from one of those groups can be given the call, the call is connected to that agent's Terminal and all the queued instances of that call are removed.

ACDManagerConnections and other ACD Objects

The ACDManagerConnection interface models the relationship between a Call and an ACDManagerAddress. A call placed to an ACDManagerAddress is one that is being offered for routing to Agent Groups. Agent Groups are modeled by ACDAddresses. When an implementation of an ACDManagerAddress routes a call to an ACDAddress, an ACDConnection is added to the routing ACDManagerConnection to model the offering or queuing nature of the call being extended to the ACDAddress

ACDManagerConnections and TerminalConnections

The getTerminalConnection() method on the Connection interface, that ACDManagerConnection extends, will always return null because ACDManagerAddresses do not have Terminals associated with them.

ACDManagerConnection States

The following are the possible core Connection states presented by this interface: IDLE, ALERTING, FAILED, DISCONNECTED.

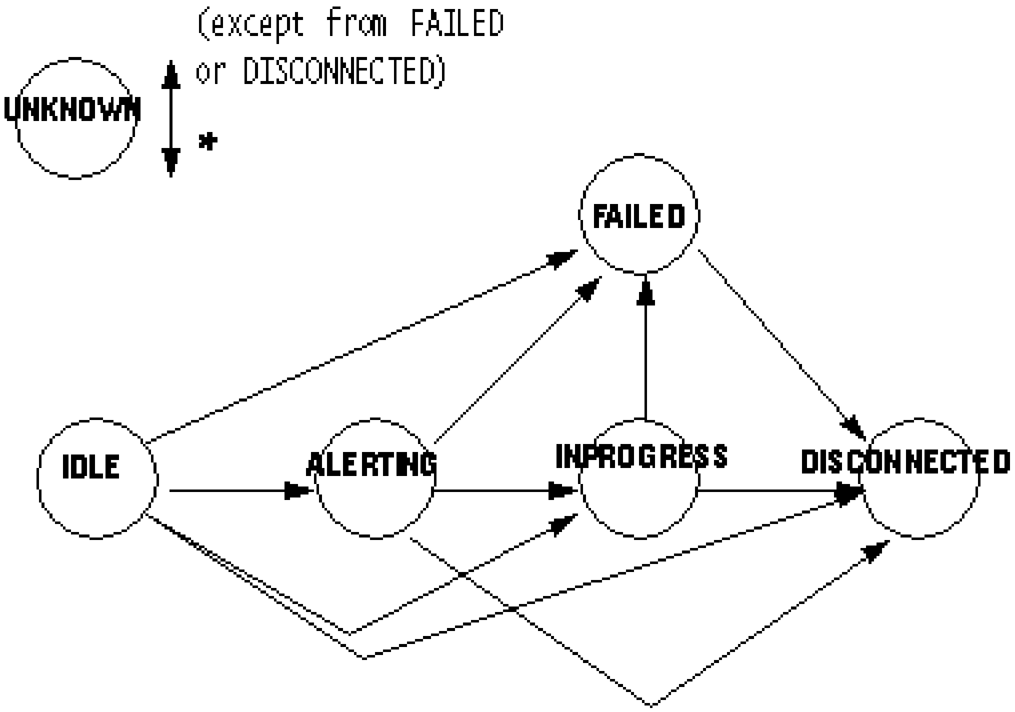
The state of the ACDManagerConnection is availble through the ACDManagerConnection.getState() inherited from the core Connection interface. Each state is an integer constant defined in the core Connection interface. Their meaning in this interface are summarized below:

ACDManagerConnection.IDLE	As in the core, this is the initial and transitory state for new ACDConnection objects.
ACDManagerConnection.ALERTING	This state indicates that the ACDManagerConnection has been made to a particular ACDManagerAddress.
ACDManagerConnection.DISCONNECTED	This state has the same definition as in the core.
ACDManagerConnection.FAILED	This state has the same definition as in the core.
ACDManagerConnection.UNKNOWN	This state has the same definition as in the core.

ACDManagerConnection State Transitions

The ACDManagerConnection class defines the allowable ACDManagerConnection state transitions. These finite-state transitions must be guaranteed by the implementation. Each method that causes a change in an ACDManagerConnection state must be consistent with this state diagram.

Note there is a general left-to-right progression of the state transitions. A Connection object may transition into and out of the ACDManagerConnection.UNKNOWN state at any time with the annotated exceptions (hence, the asterisk qualifier next to its bidirectional transition arrow).



See Also:

ACDAddress, ACDManagerAddress, ACDConnection

Method Index

- **getACDConnections**()

  Returns the ACDConnection objects associated with this ACDManagerConnection.

Methods

getACDConnections

```
 public abstract ACDConnection[] getACDConnections() throws
MethodNotSupportedException
```

Returns the ACDConnection objects associated with this ACDManagerConnection. A null will be returned if this ACDManagerConnection has no associated ACDConnections.

Returns:

The list of ACDConnection associated with this ACDManagerConnection.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.Agent

public interface **Agent**

Introduction

An `Agent` represents an individual capable of handling telephone calls for a particular Address. For example, an agent may be a customer service representative in a call center environment. An Agent is associated with a particular `AgentTerminal`, which represents the particular Terminal endpoint associated with the Agent. Each Agent is also *logged into* a particular `ACDAddress`. The act of logging into an address announces the availability of the Agent to handle Calls which come into that `ACDAddress`. Distinct Agents are used to represent the same individual who is logged into multiple `ACDAddresses` from the same `AgentTerminal`

Adding and Removing Agents

Agents are created and added to a particular `ACDAddress` via the `AgentTerminal.addAgent()` method. This method creates a new `Agent` associated with the `AgentTerminal` and the `ACDAddress` given as an argument.

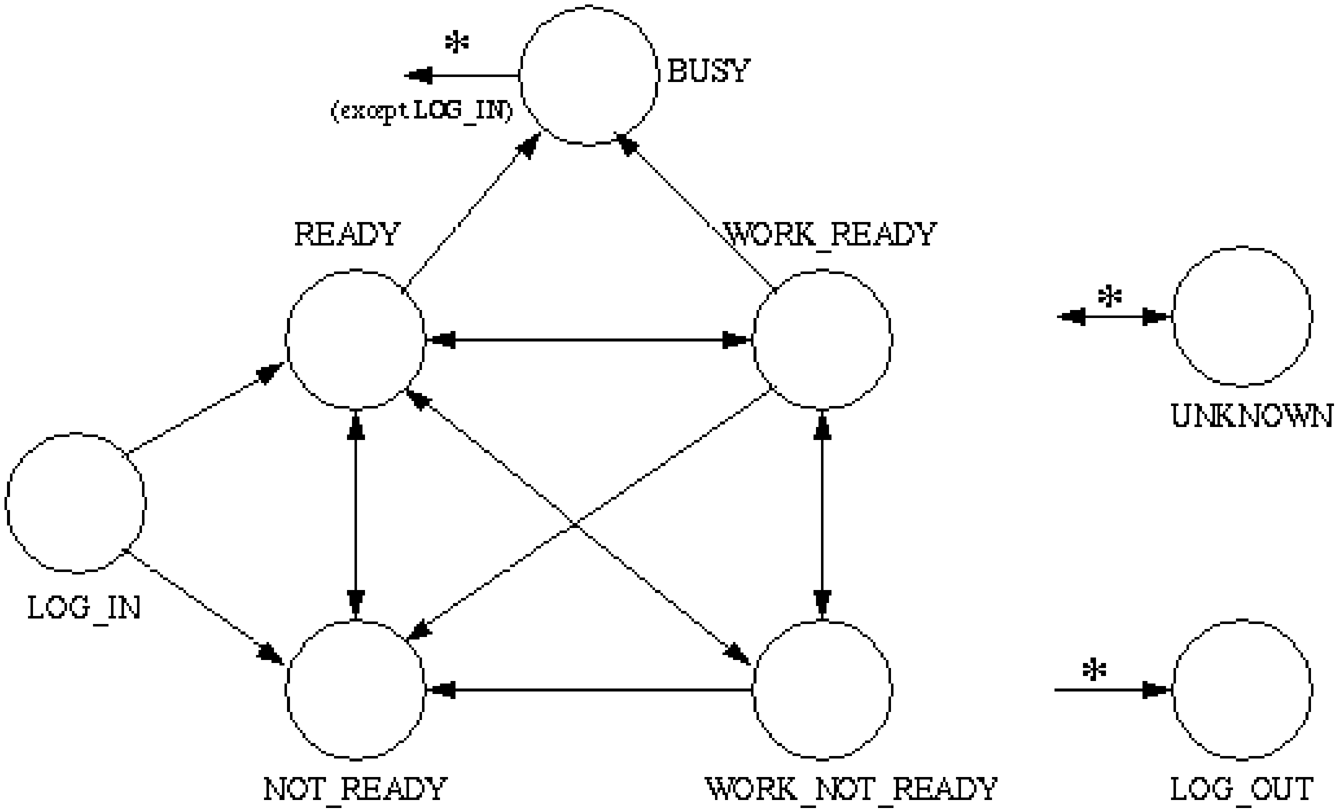
Agents are removed from an `AgentTerminal` via the `AgentTerminal.removeAgent()` method. This method also removes the Agent from its `ACDAddress`. The Agent is no longer available to receive telephone calls coming into its `ACDAddress`.

The Agent's State

The state of the Agent describes whether it is currently logged into an `ACDAddress` or its current ability to handle telephone calls. Applications obtain the state of the Agent via the `Agent.getState()` method. Applications may also directly change the state of the Agent via the `Agent.setState()` method. Each state is an integer constant defined in this interface and summarized below.

`Agent.LOG_IN`	This state indicates the Agent is logged into an `ACDAddress`.
`Agent.LOG_OUT`	This state indicates the Agent has logged out of an `ACDAddress`.
`Agent.NOT_READY`	This state indicates the Agent is not available to handle Calls because it is busy with other non-call servicing related tasks.
`Agent.READY`	This state indicates the Agent is available to service Calls.
`Agent.WORK_NOT_READY`	This state indicates the Agent is not available to service Calls because it is busy with other call-servicing related tasks.
`Agent.WORK_READY`	This state indicates the Agent is available to service Calls and is also performing other call-servicing related tasks.
`Agent.BUSY`	This state indicates the Agent is not available to service Calls because it is busy with another Call.
`Agent.UNKNOWN`	This state indicates the state of the Agent is currently not known.

The following diagram illustrates the valid state transitions for the `Agent`. The implementation must guarantee the Agent state adheres to these state transitions. If an applications requests an Agent state change which violates the transitions, the `setState()` method on this interface will throw `InvalidStateException`. The state of this object can be altered by invoking the the setState method.



Observers and Events

Application are notified when an Agent changes state via an event. Each Agent state has a corresponding event. Agent state changes are reported via two interfaces: `ACDAddressObserver` for the `ACDAddress` associated with this Agent, and `AgentTerminalObserver` for the `AgentTerminal` associated with this Agent. Both of these interfaces receive Agent state events.

See Also:

AgentTerminal, ACDAddress, AgentTerminalObserver, ACDAddressObserver

---

Variable Index

- **BUSY**

  The `Agent.BUSY` state indicates the Agent is not available to service Calls because it is busy with another Call.

- **LOG_IN**

  The `Agent.LOG_IN` state indicates that an Agent, which is associated with an `AgentTerminal` is logged into an `ACDAddress`.

- **LOG_OUT**

  The `Agent.LOG_OUT` state indicates the Agent has logged out of an `ACDAddress`.

- **NOT_READY**

  The `Agent.NOT_READY` state indicates the Agent is not available to handle Calls because it is busy with other non-call servicing related tasks.

- **READY**

  The `Agent.READY` state indicates the Agent is available to service Calls.

- **UNKNOWN**

  The `Agent.UNKNOWN` state indicates the state of the Agent is currently not known.

- **WORK_NOT_READY**

  The Agent.WORK_NOT_READY state indicates the Agent is not available to service Calls because it is busy with other call-servicing related tasks.
- **WORK_READY**

  The Agent.WORK_READY state indicates the Agent is available to service Calls and is also performing other call-servicing related tasks.

Method Index

- **getACDAddress**()

  Returns the ACDAddress which this Agent is logged into.
- **getAgentAddress**()

  Returns the Address associated with the AgentTerminal from which this Agent is logged in.
- **getAgentID**()

  Returns the Agent's string identification.
- **getAgentTerminal**()

  Returns the AgentTerminal associated with this Agent and which this Agent is logged into.
- **getState**()

  Returns the current Agent state.
- **setState**(int)

  Changes the state of the Agent.

Variables

UNKNOWN

```
public static final int UNKNOWN
```

The Agent.UNKNOWN state indicates the state of the Agent is currently not known.

LOG_IN

```
public static final int LOG_IN
```

The Agent.LOG_IN state indicates that an Agent, which is associated with an AgentTerminal is logged into an ACDAddress.

LOG_OUT

```
public static final int LOG_OUT
```

The Agent.LOG_OUT state indicates the Agent has logged out of an ACDAddress.

NOT_READY

```
public static final int NOT_READY
```

The Agent.NOT_READY state indicates the Agent is not available to handle Calls because it is busy with other non-call servicing related tasks.

READY

```
public static final int READY
```

The Agent.READY state indicates the Agent is available to service Calls.

WORK_NOT_READY

```
public static final int WORK_NOT_READY
```

The `Agent.WORK_NOT_READY` state indicates the Agent is not available to service Calls because it is busy with other call-servicing related tasks.

WORK_READY

```
public static final int WORK_READY
```

The `Agent.WORK_READY` state indicates the Agent is available to service Calls and is also performing other call-servicing related tasks.

BUSY

```
public static final int BUSY
```

The `Agent.BUSY` state indicates the Agent is not available to service Calls because it is busy with another Call.

Methods

setState

```
public abstract void setState(int state) throws InvalidArgumentException,
InvalidStateException
```

Changes the state of the Agent. The new desired state is given as a parameter to this method. The Agent's state must adhere to the state transition diagram given in this interface definition. If the given, new Agent state violates the transition diagram, this method throws `InvalidStateException`.

Pre-Conditions

1. this.getAgentTerminal().getProvider().getState()==Provider.IN_SERVICE
2. this.getState() == Agent.READY, Agent.NOT_READ, Agent.WORK_READY, or Agent.WORK_NOT_READY.

Post-Conditions

1. this.getAgentTerminal().getProvider().getState()==Provider.IN_SERVICE
2. this.getState() == state
3. The proper Agent state event is delivered to the application

Parameters:

state - The new, desired state of the Agent.

Throws: InvalidArgumentException

The state given as the argument is not a valid Agent state.

Throws: InvalidStateException

Either the provider is not in service or the Agent is not in a state in which the requested state change can be honored.

getState

```
public abstract int getState()
```

Returns the current Agent state. This method returns one of the integer constants defined by this interface.

Returns:

The current Agent state.

getAgentID

```
public abstract String getAgentID()
```

Returns the Agent's string identification. This identification is passed as an argument to the `AgentTerminal.addAgent()` method.

Returns:

the Agent's ID.

getACDAddress

```
public abstract ACDAddress getACDAddress()
```

Returns the `ACDAddress` which this Agent is logged into.

Returns:

The ACDAddress this Agent is logged into.

getAgentAddress

```
public abstract Address getAgentAddress()
```

Returns the Address associated with the `AgentTerminal` from which this Agent is logged in.

Returns:

The Agent's Address.

getAgentTerminal

```
public abstract AgentTerminal getAgentTerminal()
```

Returns the `AgentTerminal` associated with this Agent and which this Agent is logged into. If the state of the Agent is `Agent.LOG_OUT`, this method returns null.

Returns:

The AgentTerminal associated with this Agent.

Interface javax.telephony.callcenter.AgentTerminal

public interface **AgentTerminal**

extends Terminal

The `AgentTerminal` interface extends the core `Terminal` interface. This interface add methods to support ACD and Agent features.

Adding and Removing Agents

Applications may create and add new `Agents` associated with this `AgentTerminal` via the `AgentTerminal.addAgent()` method. This method creates and returns a new `Agent` which is associated with this `AgentTerminal` and an `ACDAddress` given as an argument. Agents model human individuals who are able to service telephone calls coming into an Address.

Agents may be removed from this `AgentTerminal` via the `AgentTerminal.removeAgent()` method.

See Also:

Terminal, Agent, ACDAddress, AgentTerminalObserver

Method Index

- **addAgent**(Address, ACDAddress, int, String, String)

  Creates a new `Agent` associated with this Terminal and is logged into the `ACDAddress` given as an argument.
- **getAgents**()

  Returns an array of Agents current associated with this Terminal.
- **removeAgent**(Agent)

  Removes a previously added `Agent` from this `AgentTerminal`.
- **setAgents**(Agent[])

  Sets the current list of Agents on this Terminal. **Deprecated.**

Methods

addAgent

```
 public abstract Agent addAgent(Address agentAddress,
                                ACDAddress acdAddress,
                                int initialState,
                                String agentID,
                                String password) throws InvalidArgumentException,
InvalidStateException, ResourceUnavailableException
```

Creates a new `Agent` associated with this Terminal and is logged into the `ACDAddress` given as an argument. This method returns the new `Agent` when it has been successfully created and logged into the `ACDAddress`.

Applications remove the new Agent via the `removeAgent()` method defined by this interface. Applications obtain all Agents associated with this Terminal via the `getAgents()` method defined on this interface.

Subsequent invocations of this methods with the same agentAddress and acdAddress parameters will simply return the Agent originally created.

Pre-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE
2. initialState == Agent.LOGIN, Agent.READY, or Agent.NOT_READY

Post-Conditions

1. Let agent be the Agent created an returned
2. this.getProvider().getState() == Provider.IN_SERVICE
3. agent is an element of this.getAgents()
4. agent.getState() == initialState
5. Either an AgentTermLoggedOnEv, AgentTermReadyEv, or AgentTermNotReadyEv is delivered for this Agent
6. Either an ACDAddrLoggedOnEv, ACDAddrReadyEv, or ACDAddrNotReadyEv is delivered for this Agent

Parameters:

agentAddress - The Address associated with this Terminal. Terminals may support more than one Address on which an Agent may be associated.

acdAddress - The Address which the Agent logs into.

initialState - The initial state of the Agent.

agentID - The Agent's string identification.

password - The string password which authorizes the application to log in as an Agent.

Returns:

An Agent representing the association between this AgentTerminal and the ACDAddress.

Throws: ResourceUnavailableException

An internal resource necessary for adding the Agent to this Terminal and ACDAddress is unavailable.

Throws: InvalidArgumentException

An argument provided is not valid either by not providing enough information for addAgent() or is inconsistent with another argument.

Throws: InvalidStateException

Either the provider is not in service or the AgentTerminal is not in a state in which it can be logged into the ACDAddress.

removeAgent

```
 public abstract void removeAgent(Agent agent) throws InvalidArgumentException,
InvalidStateException
```

Removes a previously added `Agent` from this `AgentTerminal`. This method returns when the Agent is logged out of the `ACDAddress` and the state of the Agent moves to `Agent.LOG_OUT`.

Pre-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE
2. agent is an element of this.getAgents()

Post-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE
2. agent is not an element of this.getAgents()
3. agent.getState() == Agent.LOG_OUT
4. AgentTermLoggedOffEv and ACDAddrLoggedOffEv is delivered for the Agent.

Parameters:

agent - The Agent to be removed and logged out.

Throws: InvalidArgumentException

The Agent given is not valid.

Throws: InvalidStateException

Either the provider is not in service or the AgentTerminal is not in a state in which it can be logged out of the ACDAddress.

See Also:

AgentTermLoggedOffEv, AgentTermLoggedOnEv

getAgents

```
public abstract Agent[] getAgents()
```

Returns an array of Agents current associated with this Terminal. This method returns `null` is no Agents exist. Agents are reported via this method once they are added via the `addAgent()` method until they are removed via the `removeAgent()` method defined on this interface.

Returns:

A list of Agents associated with this Terminal.

setAgents

```
public abstract void setAgents(Agent agents[]) throws MethodNotSupportedException
```

Note: setAgents() is deprecated. *Since JTAPI v1.2. Has been replaced with addAgent() and removeAgent()*

Sets the current list of Agents on this Terminal. This method may remove Agents previously on this Terminal. When this method is finished, the set of Agents on this Terminal will be the given array.

Note: This method has been removed for JTAPI v1.2 and later. It has been replaced with the `AgentTerminal.addAgent()` and `AgentTerminal.removeAgent()` methods. This method should now always throw `MethodNotSupportedException`.

Parameters:

The - array of Agents to be either added, removed, or changed.

Throws: MethodNotSupportedException

This exception should always be thrown for JTAPI v1.2 and later.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.AgentTerminalObserver

public interface **AgentTerminalObserver**

extends TerminalObserver

The `AgentTerminalObserver` interface reports all state changes in the `Agent` that is associated with the `AgentTerminal` as events. Applications instantiate an object which implements this interface and use the `Terminal.addObserver()` to request delivery of events to this observer object. Events will be delivered to this interface only if the Provider is in the `Provider.IN_SERVICE` state. All events which are reported via this interface must extend the `AgentTermEv` interface.

Events are reported via the `TerminalObserver.terminalChangedEvent()` method. This interface defines no additional methods and therefore serves as a way applications signal to the implementation that is desires call center package events.

Note that the state changes in the `Agent` are also reported via the `ACDAddressObserver` for the `ACDAddress` on which the `Agent` is logged in to.

See Also:

Terminal, TerminalObserver, AgentTerminal, ACDAddressObserver, AgentTermEv, AgentTermBusyEv, AgentTermLoggedOffEv, AgentTermLoggedOnEv, AgentTermNotReadyEv, AgentTermReadyEv, AgentTermUnknownEv, AgentTermWorkNotReadyEv, AgentTermWorkReadyEv

Interface javax.telephony.callcenter.CallCenterAddress

public interface **CallCenterAddress**

extends Address

The `CallCenterAddress` interface is the base Address interface for the call center package. This package defines two additional Address interfaces which both extend this interface: `ACDAddress` and `ACDManagerAddress`.

The `addCallObserver()` Method

This interface defines a version of the `addCallObserver()` method which overloaded the definition in the `Address` interface. This overloaded version accepts an additional boolean parameter which allows applications to monitor Calls which come to this Address for the lifetime of the Call, and not just while it is at this Address.

See Also:

Address, ACDAddress, ACDManagerAddress

Method Index

addCallObserver(CallObserver, boolean)

This method behaves similarly to `Address.addCallObserver()`, with the following exceptions:

- If *remain* is true, the observer remains on all Calls which come to this Address, for the lifetime of the Call.

Methods

addCallObserver

```
 public abstract void addCallObserver(CallObserver observer,
                                      boolean remain) throws
ResourceUnavailableException, PrivilegeViolationException,
MethodNotSupportedException
```

This method behaves similarly to `Address.addCallObserver()`, with the following exceptions:

- If *remain* is true, the observer remains on all Calls which come to this Address, for the lifetime of the Call.
- If *remain* is false, this method behaves exactly the same as `Address.addCallObserver()`

If an application attempts to add an instance of an observer to the Address more than once, whether via the method or via the `Address.addCallObserver()` method, only a single instance of an observer will be added. Repeated attempts to add an observer will silently fail, i.e. no exception is thrown.

Post-Conditions:

1. observer is an element of this.getCallObservers()
2. observer is an element of Call.getObservers() for each Call associated with the Connections from this.getConnections()
3. An array of snapshot events is reported to the observer for existing calls associated with this Address.

Parameters:

observer - The observer being added.

remain - If true, the observer remains on the Call for the lifetime of the Call. If false, the observer uses the default behavior.

Throws: MethodNotSupportedException

The Address is currently not observable.

Throws: ResourceUnavailableException

The resource limit for the numbers of observers has been exceeded.

Throws: PrivilegeViolationException

The application does not have the proper authority to perform this type of observation.

See Also:

Address

All Packages This Package Previous Next

All Packages This Package Previous Next

Interface javax.telephony.callcenter.CallCenterCall

public interface **CallCenterCall**

extends Call

Introduction

The `CallCenterCall` interface extends the core `Call` interface. This interface provides additional Call-related features for the call center package. Applications may query an object with the `instanceof` operator to check whether it supports this interface.

Predictive Dialing

Predictive dialing is a special means to place a telephone call. In a predictive telephone call, the destination end is created and placed on the Call first. Only when the destination Connection reaches a certain state, as designated by the application, the originating Connection is created and the originating end is placed on the Call. Predictive dialing is used, for example, when customers are called from a long list, and a customer service representative is placed on the Call only when a customer can be reached. Applications placed predictive telephone calls via the `connectPredictive()` method on this interface.

Application Data

Applications may associated an arbitrary piece of data with a Call. This data is seen by any application which has a handle to a Call. This mechanism is often used to store information specific to a Call, such as customer information. The `setApplicationData()` method defined on this sets the current data and the `getApplicationData()` method retrieves the current application-specific data.

Trunks

Zero or more trunks may be associated with the Call. Applications obtain the trunks associated with a Call via the `getTrunks()` method on this interface. A trunk is represented by the `CallCenterTrunk` interface.

Observers and Events

Events pertaining to the `CallCenterCall` interface are reported via the `CallCenterCallObserver` interface. The following are those events reported via this interface:

- Connection "in progress" state event
- Trunk state events
- Application data change events

See Also:

CallCenterCallObserver, CallCenterTrunk

Variable Index

ANSWERING_TREATMENT_CONNECT

Answering endpoint treatment: connect the Call when the endpoint is detected.

ANSWERING_TREATMENT_DROP

Answering endpoint treatment: drop the Call when the endpoint is detected.

ANSWERING_TREATMENT_NONE

Answering endpoint treatment: no treatment specified.

ANSWERING_TREATMENT_PROVIDER_DEFAULT

Answering endpoint treatment: follow the default treatment.

ENDPOINT_ANSWERING_MACHINE

Answering endpoint type: answering endpoint may be an answering machine.

ENDPOINT_ANY

Answering endpoint type: answering endpoint may be anything.

ENDPOINT_FAX_MACHINE

Answering endpoint type: answering endpoint may be a fax machine.

ENDPOINT_HUMAN_INTERVENTION

Answering endpoint type: answering endpoint may be a human.

MAX_RINGS

The `CallCenterCall.MIN_RINGS` constant defines the minimum number of rings which the application may specify for the destination end before a predictive telephone call is classified as "no answer".

MIN_RINGS

The `CallCenterCall.MIN_RINGS` constant defines the minimum number of rings which the application may specify for the destination end before a predictive telephone call is classified as "no answer".

Method Index

connectPredictive(Terminal, Address, String, int, int, int, int)

Places a predictive telephone call.

getApplicationData()

Returns the application-specific data associated with the Call.

getTrunks()

Returns an array of all `CallCenterTrunks` currently being used for this Call.

setApplicationData(Object)

This method associates application specific data with a Call.

Variables

MIN_RINGS

```
public static final int MIN_RINGS
```

The `CallCenterCall.MIN_RINGS` constant defines the minimum number of rings which the application may specify for the destination end before a predictive telephone call is classified as "no answer".

MAX_RINGS

```
public static final int MAX_RINGS
```

The `CallCenterCall.MIN_RINGS` constant defines the minimum number of rings which the application may specify for the destination end before a predictive telephone call is classified as "no answer".

ANSWERING_TREATMENT_PROVIDER_DEFAULT

```
public static final int ANSWERING_TREATMENT_PROVIDER_DEFAULT
```

Answering endpoint treatment: follow the default treatment. The answering endpoint treatment should follow the default treatment.

ANSWERING_TREATMENT_DROP

```
public static final int ANSWERING_TREATMENT_DROP
```

Answering endpoint treatment: drop the Call when the endpoint is detected.

ANSWERING_TREATMENT_CONNECT

```
public static final int ANSWERING_TREATMENT_CONNECT
```

Answering endpoint treatment: connect the Call when the endpoint is detected.

ANSWERING_TREATMENT_NONE

```
public static final int ANSWERING_TREATMENT_NONE
```

Answering endpoint treatment: no treatment specified.

ENDPOINT_ANSWERING_MACHINE

```
public static final int ENDPOINT_ANSWERING_MACHINE
```

Answering endpoint type: answering endpoint may be an answering machine.

ENDPOINT_FAX_MACHINE

```
public static final int ENDPOINT_FAX_MACHINE
```

Answering endpoint type: answering endpoint may be a fax machine.

ENDPOINT_HUMAN_INTERVENTION

```
public static final int ENDPOINT_HUMAN_INTERVENTION
```

Answering endpoint type: answering endpoint may be a human.

ENDPOINT_ANY

```
public static final int ENDPOINT_ANY
```

Answering endpoint type: answering endpoint may be anything.

Methods

connectPredictive

```
public abstract Connection[] connectPredictive(Terminal originatorTerminal,
                                               Address originatorAddress,
                                               String destination,
                                               int connectionState,
                                               int maxRings,
                                               int treatment,
                                               int endpointType) throws
ResourceUnavailableException, PrivilegeViolationException, InvalidPartyException,
InvalidArgumentException, InvalidStateException, MethodNotSupportedException
```

Places a predictive telephone call. A predictive telephone call is a telephone call placed to the destination end first, and connects the originating end only if the destination end reaches either the `Connection.CONNECTED` or `Connection.ALERTING` state. The destination Connection is created first, and the originating Connection is created only if the destination reaches the designated state. This method returns successfully when both Connections are created and returned.

The originating and destination end arguments

The first three arguments are identical to the arguments of the `Call.connect()` method. They represent the desired originating and destination ends of the Call. The originating Terminal may be `null`, however, for certain types of Addresses, such as ACD Addresses, this argument may be `null`. The destination address string given must be complete and valid.

The target destination Connection state

The application designates when the originating end of the Call is created and placed on the telephone Call, based upon the state of the destination Connection. The desired target state for the destination Connection is given as the *connectionState* argument to this method. The value must be either `Connection.CONNECTED` or `Connection.ALERTING`. If the destination Connection never reaches this state, this method throws an appropriate exception.

The maximum number of rings

The application may also designate the maximum number of rings allowed on the destination end before the Call is classified as a "no answer". The value must be between `CallCenterCall.MIN_RINGS` (2) and `CallCenterCall.MAX_RINGS` (15).

Answering treatment and endpoint type

The two final arguments specify how the Call is treated when the destination Connection reaches its target destination, and the allowed kinds of endpoints on the answering end. Each of these arguments must be one of the designated constants defined by this interface.

The returned Connections

The Connections created and returned by this method behave similarly to Connections which were returned from `Call.connect()`. The originating Connection moves into the `Connection.CONNECTED` state when an originating party is placed on the Call. If the target state for the destination Connection is `Connection.ALERTING`, it moves into the `Connection.CONNECTED` state when the called party answers the Call.

Pre-conditions

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. connectionState == Connection.CONNECTED or Connection.ALERTING
4. maxRings >= CallCenterCall.MIN_RINGS
5. maxRings <= CallCenterCall.MAX_RINGS

Post-conditions

1. Let connections[] = this.getConnections()
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.IDLE
4. connections.length == 2
5. connections[0].getState() == connectionState
6. connections[1].getState() == Connection.IDLE

Parameters:

originatorTerminal - The originating Terminal. If the originating Address is an ACD Address, for example, this value may be null.

originatorAddress - The originating Address of the telephone call.

dialedDigits - The complete and valid telephone address string.

connectionState - The target state for the destination Connection before the originating Connection is created. This must either be Connection.CONNECTED or Connection.ALERTING

maxRings - The maximum number of rings before classifying the Call as "no answer".

treatment - The treatment of the Call when the endpoint is detected.

endpointType - The permitted answering endpoint type.

Returns:

An array of the originating and destination Connection

Throws: ResourceUnavailableException

An internal resource necessary for placing the phone call is unavailable.

Throws: PrivilegeViolationException

The application does not have the proper authority to place a telephone call.

Throws: InvalidPartyException

Either the originator or the destination does not represent a valid party required to place a telephone call.

Throws: InvalidArgumentException

An argument provided is not valid either by not providing enough information or is inconsistent with another argument.

Throws: InvalidStateException

Either the Provider is not in service or the Call is not idle.

Throws: MethodNotSupportedException

The implementation does not support this method.

setApplicationData

```
 public abstract void setApplicationData(Object data) throws
ResourceUnavailableException, InvalidArgumentException, InvalidStateException,
MethodNotSupportedException
```

This method associates application specific data with a Call. The format of the data is application-specific. The application-specific data given in this method replaces any existing application data. If the argument given is `null`, the current application data (if any) is removed.

In the case that a Call is transfered or conferenced, the application data from the Call from which the conference or transfer is invoked will be retained.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE or Call.IDLE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE or Call.IDLE
3. this.getApplicationData() = data
4. A CallCentCallAppDataEv is delivered for this Call

Parameters:

data - The data to be associated with the call.

Throws: ResourceUnavailableException

An internal resource necessary for adding the data was unavailable. For example, the size of the Object was not supported by the implementation.

Throws: InvalidArgumentException

The argument provided is not valid. For example, the implementation does not support the specific object type.

Throws: InvalidStateException

Either the Provider was not in service or the Call was not active or idle.

Throws: MethodNotSupportedException

The implementation does not support this method.

See Also:

CallCentCallAppDataEv

getApplicationData

```
public abstract Object getApplicationData() throws MethodNotSupportedException
```

Returns the application-specific data associated with the Call. This method returns `null` is there is no associated data.

Returns:

s The application-specific data associated with this Call.

Throws: MethodNotSupportedException

The implementation does not support this method.

getTrunks

```
public abstract CallCenterTrunk[] getTrunks() throws MethodNotSupportedException
```

Returns an array of all `CallCenterTrunks` currently being used for this Call. If there are no trunks being used for this Call, this method returns null. Each trunk returns by this method will be in the `CallCenterTrunk.VALID` state.

Post-conditions:

1. Let CallCenterTrunk[] trunks = this.getTrunks()
2. trunks == null or trunks.length >= 1
3. trunks[i].getState() == CallCenterTrunk.VALID_TRUNK, for all i

Returns:

An array of trunks currently associated with this Call.

Throws: MethodNotSupportedException

The implementation does not support this method.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.CallCenterCallObserver

public interface **CallCenterCallObserver**

extends CallObserver

The `CallCenterCallObserver` interface extends the event reporting of of the core `CallObserver` to include call center related events. Applications instantiate an object which implements this interface and use the `Call.addObserver()` to request delivery of events to this observer object. Events will be delivered to this interface only if the Provider is in the `Provider.IN_SERVICE` state. All events which are reported via this interface must extend the `CallCentCallEv` interface.

Events are reported via the `CallObserver.callChangedEvent()` method. This interface defines no additional methods and therefore serves as a way applications signal to the implementation that is desires call center package events.

See Also:

Call, CallObserver, CallCentCallEv, CallCentTrunkValidEv, CallCentTrunkInvalidEv, CallCentCallAppDataEv, CallCentConnInProgressEv

All Packages This Package Previous Next

Interface javax.telephony.callcenter.CallCenterProvider

public interface **CallCenterProvider**

extends Provider

The `CallCenterProvider` interface extends the core `Provider` interface. This interface defines additional methods to query the Provider's local domain. This interface defines method to return the routeable addresses, the ACD addresses, and the ACD manager addresses in the Provider's domain.

See Also:

Provider, RouteAddress, ACDAddress, ACDManagerAddress

Method Index

- **getACDAddresses**()

  Returns an array of ACD Addresses associated with the Provider and within the Provider's domain.
- **getACDManagerAddresses**()

  Returns an array of ACD manager Addresses associated with the Provider and within the Provider's domain.
- **getRouteableAddresses**()

  Returns an array of routeable Addresses associated with the Provider and within the Provider's domain.

Methods

getRouteableAddresses

```
 public abstract RouteAddress[] getRouteableAddresses() throws
MethodNotSupportedException
```

Returns an array of routeable Addresses associated with the Provider and within the Provider's domain. This list is static (i.e. is does not change) after the Provider is first created. If no routeable Addresses are associated with this Provider, then this method returns null.

Post-conditions:

1. Let RouteAddress[] addresses = this.getRouteableAddresses()
2. addresses == null or addresses.length >= 1

Returns:

An array of RouteAddresses in the Provider's domain

Throws: MethodNotSupportedException

This method is not supported by the implementation.

getACDAddresses

```
 public abstract ACDAddress[] getACDAddresses() throws MethodNotSupportedException
```

Returns an array of ACD Addresses associated with the Provider and within the Provider's domain. This list is static (i.e. is does not change) after the Provider is first created. If no ACD Addresses are associated with this Provider, then this method returns null.

Post-conditions:

1. Let ACDAddress[] addresses = this.getACDAddresses()
2. addresses == null or addresses.length >= 1

Returns:

An array of ACDAddresses in the Provider's domain

Throws: MethodNotSupportedException

This method is not supported by the implementation.

● **getACDManagerAddresses**

```
 public abstract ACDManagerAddress[] getACDManagerAddresses() throws
MethodNotSupportedException
```

Returns an array of ACD manager Addresses associated with the Provider and within the Provider's domain. This list is static (i.e. is does not change) after the Provider is first created. If no ACD manager Addresses are associated with this Provider, then this method returns null.

Post-conditions:

1. Let ACDManagerAddress[] addresses = this.getACDManagerAddresses()
2. addresses == null or addresses.length >= 1

Returns:

An array of ACDManagerAddresses in the Provider's domain

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Interface javax.telephony.callcenter.CallCenterTrunk

public interface **CallCenterTrunk**

Introduction

The `CallCenterTrunk` interface represents a trunk on the underlying telephony hardware. Each trunk has four attributes: its *name*, its *state*, its *type*, and its *associated Call*.

The Trunk Name

The first attribute of a trunk is its string name. This name is assigned to the trunk by the underlying telephony hardware. Applications obtains the name of the trunk via the `getName()` method on this interface. A trunk's name does not change throughout its lifetime.

The Trunk State

The second attribute of a trunk is its state. The state indicates whether the trunk is associated with a Call (i.e. valid) or not associated with a Call (i.e. invalid). Applications obtain the state of a trunk via the `getState()` method on this interface. The following chart summarizes the two trunk states:

`CallCenterTrunk.VALID_TRUNK`	The trunk is valid and associated with a Call.
`CallCenterTrunk.INVALID_TRUNK`	The trunk is not valid and not associated with a Call.

The Trunk Type

The third attribute of a trunk is its type. The type indicates whether the trunk is an "incoming", "outgoing", or "unknown". Applications obtain the trunk type via the `getType()` method on this interface. The following chart summarizes the three trunk types:

`CallCenterTrunk.INCOMING_TRUNK`	The trunk is an incoming trunk
`CallCenterTrunk.OUTGOING_TRUNK`	The trunk is an outgoing trunk
`CallCenterTrunk.UNKNOWN_TRUNK`	The trunk type is not known

The Associated Call

The fourth attribute of a trunk is the Call associated with it. The Call is assigned when the trunk is first created and remains the same throughout the lifetime of the trunk. Applications obtain the associated Call via the `getCall()` method on this interface.

Observers and Events

Applications receive an event whenever the state of the trunk changes. These events are reported via the `CallCenterCallObserver` interface. The `CallCentTrunkValidEv` event is delivered when the trunk moves to the `CallCenterTrunk.VALID_TRUNK` state and a `CallCentTrunkInvalidEv` event is devliered when the trunk moves to the `CallCenterTrunk.INALID_TRUNK` state.

See Also:

CallCenterCall, CallCenterCallObserver, CallCentTrunkValidEv, CallCentTrunkInvalidEv

Variable Index

- **INCOMING_TRUNK**

Trunk type: The trunk is incoming.

- **INVALID_TRUNK**

  Trunk state: The trunk is invalid.

- **OUTGOING_TRUNK**

  Trunk type: The trunk is outgoing.

- **UNKNOWN_TRUNK**

  Trunk type: The trunk is unknown.

- **VALID_TRUNK**

  Trunk state: The trunk is valid.

Method Index

- **getCall**()

  Returns the Call associated with this trunk.

- **getName**()

  Returns the name of the trunk.

- **getState**()

  Returns the current state of the Trunk.

- **getType**()

  Returns the type of trunk.

Variables

INVALID_TRUNK

```
public static final int INVALID_TRUNK
```

Trunk state: The trunk is invalid.

VALID_TRUNK

```
public static final int VALID_TRUNK
```

Trunk state: The trunk is valid.

INCOMING_TRUNK

```
public static final int INCOMING_TRUNK
```

Trunk type: The trunk is incoming.

OUTGOING_TRUNK

```
public static final int OUTGOING_TRUNK
```

Trunk type: The trunk is outgoing.

UNKNOWN_TRUNK

```
public static final int UNKNOWN_TRUNK
```

Trunk type: The trunk is unknown.

Methods

getName

```
public abstract String getName()
```

Returns the name of the trunk. This name is assigned by the underlying telephony hardware.

Returns:

The name of the trunk.

getState

```
public abstract int getState()
```

Returns the current state of the Trunk.

Returns:

The current state of the trunk.

getType

```
public abstract int getType()
```

Returns the type of trunk.

Returns:

The type of trunk.

getCall

```
public abstract Call getCall()
```

Returns the Call associated with this trunk. This Call reference remains valid throughout the lifetime of the trunk, despite the current state of the trunk.

Returns:

The Call associated with this trunk.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.RouteAddress

public interface **RouteAddress**

extends Address

The `RouteAddress` interface extends the core `Address` interface and add methods to allow applications the ability to select destinations for calls coming into this Address.

Applications may register to route calls for this Address via the `registerRouteCallback()` method defined on this interface. This method takes an instance of an object which implements the `RouteCallback` interface. For each Call which comes into this Address, a new `RouteSession` is created for each callback which is currently registered. The callbacks will receive routing requests via the callbacks.

Applications may register to route calls for all RouteAddresses via a special Address created by the Provider. This Address has the name `RouteAddress.ALL_ROUTE_ADDRESS` and may be obtained via the `Provider.getAddresses()` or `Provider.getAddress()` method. When applications invoke the `registerRouteCallback()` method on this special address, the callback will receive routing requests for all routeable Addresses in the Provider's domain.

Variable Index

ALL_ROUTE_ADDRESS

The string name of the special Address created by the Provider used by applications to register a callback for all routeable Addresses in the Provider's domain.

Method Index

cancelRouteCallback(RouteCallback)

Cancels a previously registered routing callback for this Address.

getActiveRouteSessions()

Returns an array of all active route sessions associated with this Address.

getRouteCallback()

Returns an array of all callbacks which are registered to route Calls for this Address.

registerRouteCallback(RouteCallback)

Registers a callback to route calls for this Address.

Variables

ALL_ROUTE_ADDRESS

```
public static final String ALL_ROUTE_ADDRESS
```

The string name of the special Address created by the Provider used by applications to register a callback for all routeable Addresses in the Provider's domain.

Methods

registerRouteCallback

```
 public abstract void registerRouteCallback(RouteCallback routeCallback) throws
ResourceUnavailableException, MethodNotSupportedException
```

Registers a callback to route calls for this Address. The callback given as an argument will be notified of all routing requests for Calls which come into this Address. Applications may register a callback for all routeable Addresses in the Provider's domain by invoke this method on a special Address with the name `RouteAddress.ALL_ROUTE_ADDRESS`.

Multiple callbacks may be registered on a single Address. This method throws `ResourceUnavailableException` if the maximum number of registered callbacks on the Address has been exceeded.

Pre-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE

Post-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE
2. routeCallback is an element of this.getRouteCallback()

Parameters:

routeCallback - The callback to be registered.

Throws: ResourceUnavailableException

Indicates the maximum number of registered callbacks for this Address has been exceeded.

Throws: MethodNotSupportedException

The implementation does not support this method.

cancelRouteCallback

```
 public abstract void cancelRouteCallback(RouteCallback routeCallback) throws
MethodNotSupportedException
```

Cancels a previously registered routing callback for this Address. If the given callback is currently no registered on this Address, this method fails silently, i.e. no callback is removed and no exception is thrown.

Pre-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE

Post-Conditions

1. this.getProvider().getState() == Provider.IN_SERVICE
2. routeCallback is not an element of this.getRouteCallback()

Parameters:

routeCallback - The callback to be cancelled.

Throws: MethodNotSupportedException

will be thrown if provider does not support this method.

getRouteCallback

```
 public abstract RouteCallback[] getRouteCallback() throws
MethodNotSupportedException
```

Returns an array of all callbacks which are registered to route Calls for this Address. This method returns `null` if there exists no registered callbacks.

Returns:

An array of register callbacks.

Throws: MethodNotSupportedException

The implementation does not support this method.

getActiveRouteSessions

```
 public abstract RouteSession[] getActiveRouteSessions() throws
MethodNotSupportedException
```

Returns an array of all active route sessions associated with this Address. This method returns `null` if there exists no active route sessions.

Returns:

An array of active route sessions associated with this Address.

Throws: MethodNotSupportedException

The implementation does not support this method.

All Packages   This Package Previous Next

Interface javax.telephony.callcenter.RouteCallback

public interface **RouteCallback**

The `RouteCallback` interface is used by applications to receive routing requests for a particular `RouteSession`. Applications instantiates an object which implements this interface and registers the callback for a particular routeable address via the `RouteAddress.registerRouteCallback()` method.

Applications override the individual methods defined by this interface, each of which corresponds to a different kind of routing request.

See Also:

RouteAddress, RouteSession

Method Index

- **reRouteEvent**(ReRouteEvent)

  This method indicates the application is being asked to select another route for a Call.
- **routeCallbackEndedEvent**(RouteCallbackEndedEvent)

  This method indicates that this callback will no longer receive routing requests or routing information and the callback has been terminated.
- **routeEndEvent**(RouteEndEvent)

  This method indicates that a routing session has ended.
- **routeEvent**(RouteEvent)

  This method indicates the application is being asked to route a Call.
- **routeUsedEvent**(RouteUsedEvent)

  This method indicates that a Call has successfully reached a final destination which the application has selected.

Methods

routeEvent

```
public abstract void routeEvent(RouteEvent event)
```

This method indicates the application is being asked to route a Call. The `RouteSession` associated with this `RouteCallback` has transitioned into the `RouteSession.ROUTE` state.

Parameters:

event - The RouteEvent object describing the routing request.

reRouteEvent

```
public abstract void reRouteEvent(ReRouteEvent event)
```

This method indicates the application is being asked to select another route for a Call. The `RouteSession` associated with this `RouteCallback` has transitioned into the `RouteSession.RE_ROUTE` state.

Parameters:

event - The ReRouteEvent object describing the routing request.

routeUsedEvent

```
public abstract void routeUsedEvent(RouteUsedEvent event)
```

This method indicates that a Call has successfully reached a final destination which the application has selected. The `RouteSession` associated with this `RouteCallback` has transitioned into the `RouteSession.ROUTE_USED` state.

Parameters:

event - The RouteUsedEvent object describing the route used.

routeEndEvent

```
public abstract void routeEndEvent(RouteEndEvent event)
```

This method indicates that a routing session has ended. The `RouteSession` associated with this `RouteCallback` has transitioned into the `RouteSession.ROUTE_END` state.

Parameters:

event - The RouteEndEvent object describing the ending of the routing session.

routeCallbackEndedEvent

```
public abstract void routeCallbackEndedEvent(RouteCallbackEndedEvent event)
```

This method indicates that this callback will no longer receive routing requests or routing information and the callback has been terminated. The `RouteSession` associated with this `RouteCallback` has transitioned into the `RouteSession.ROUTE_CALLBACK_ENDED` state.

Parameters:

event - The RouteCallbackEndedEvent object describing the ending of the routing callback.

All Packages   This Package   Previous   Next

Interface javax.telephony.callcenter.RouteSession

public interface **RouteSession**

Introduction

A RouteSession represents an outstanding route request of a Call. Each session is associated with a particular RouteAddress which represents the Address to which the Call was originally placed. This RouteAddress is obtained via the getRouteAddress() method defined on this interface. Applications must have previously asked to route calls to this Address.

Routing Callbacks

Each RouteSession may be associated with zero or more routing callbacks, as represented by the RouteCallback interface. Using the RouteCallback interface, applications may routing decisions for Calls. Applications register a callback via the RouteAddress.registerRouteCallback() method. Current callbacks registered on a RouteAddress are associated with all new RouteSessions created at that RouteAddress. A RouteSession, therefore, may have more than one callback associated with it. The first RouteCallback to respond with a routing request for a particular RouteSession wins, in the case multiple RouteCallbacks exist.

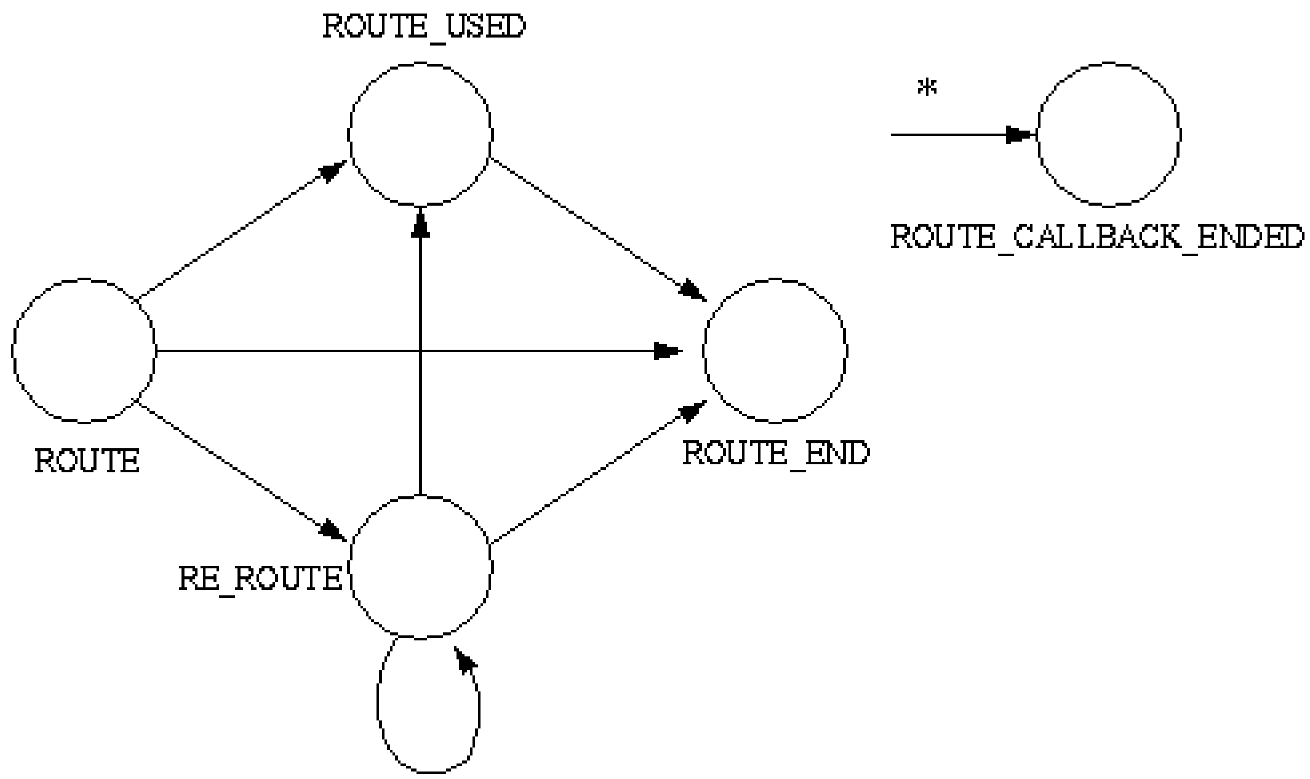
The Routing State

A RouteSession has a *state* which represents the current status of the Call with respect to the routing requests submitted by the application. These states are defined as static integer constants on this interface. Applications obtain the current state via the RouteSession.getState() method. The various states of a route session are summarized below.

RouteSession.ROUTE	This state indicates that an application has been requested to route a Call.
RouteSession.ROUTE_USED	This state indicates that a destination has been selected for a Call. This destination is one which the application had selected during its routing of the Call.
RouteSession.ROUTE_END	This state indicates that the routing of a Call has terminated.
RouteSession.RE_ROUTE	This state indicates that an application has been requested to select another destination for the Call.
RouteSession.ROUTE_CALLBACK_ENDED	This state indicates that all RouteCallback objects have been removed from this routing session. There are no more routing callbacks associated with this session. This is the final state for the RouteSession interface.

RouteSession State Transition Diagram

The states of the RouteSession must transition according to the finite state diagram below. The implementation must guarantee the state of a RouteSession adheres to these valid transitions.



Selecting the Routing for a Call

Applications use the `RouteSession.selectRoute()` method to select possible destinations for the Call associated with this routing session. The state of the `RouteSession` must either be `RouteSession.ROUTE` or `RouteSession.RE_ROUTE` in order for this method to be valid.

An array of destination address strings are given to this method. This list represents a priority-order list of possible destinations for the Call. The Call is routed to the first destination given (at index 0). If it fails, the second destination is attempted. This process is repeated until there are no more remaining destinations, or until a successful destination has been chosen. If a successful destination has been chosen, the state of the `RouteSession` moves into `RouteSession.ROUTE_USED`.

See Also:

RouteCallback, RouteAddress, RouteSessionEvent, RouteEvent, ReRouteEvent, RouteUsedEvent, RouteEndEvent, RouteCallbackEndedEvent

---

Variable Index

- **CAUSE_INVALID_DESTINATION**

  Cause code indicating that the routing session is being terminated because because the application supplied an invalid destination in the `RouteSession.routeSelect()` method.

- **CAUSE_NO_ERROR**

  Cause code indicating no error.

- **CAUSE_PARAMETER_NOT_SUPPORTED**

  Cause code indicating that the routing session is being terminated because the application supplied an unsupported parameter in the `RouteSession.routeSelect()` method.

- **CAUSE_ROUTING_TIMER_EXPIRED**

  Cause code indicating a routing timer has expired.

- **CAUSE_STATE_INCOMPATIBLE**

  Cause code indicating that the routing session is being terminated because the `Connection` state is incompatable with the `RouteSession`.

- **CAUSE_UNSPECIFIED_ERROR**

  Cause code indicating that the routing session is being terminated for unspecified reasons.

- **ERROR_RESOURCE_BUSY**

  Error code indicating the application is too busy to handle the routing request.

- **ERROR_RESOURCE_OUT_OF_SERVICE**

  Error code indicating the application or a database upon which it relies for routing is temporary out of service and cannot handle the routing request.

- **ERROR_UNKNOWN**

  Error code indicating the application is not giving a reason for ending the routing session.

- **RE_ROUTE**

  The `RouteSession.RE_ROUTE` state indicates that an application has been requested to select another destination for the Call.

- **ROUTE**

  The `RouteSession.ROUTE` state indicates that an application has been requested to route a Call.

- **ROUTE_CALLBACK_ENDED**

  The `RouteSession.ROUTE_CALLBACK_ENDED` state indicates that all `RouteCallback` objects have been removed from this routing session.

- **ROUTE_END**

  The `RouteSession.ROUTE_END` state indicates that the routing of a Call has terminated.

- **ROUTE_USED**

  The `RouteSession.ROUTE_USED` state indicates that a destination has been selected for a Call.

Method Index

- **endRoute**(int)

  Ends a routing session.

- **getCause**()

  Returns the cause indicating why this route session is in its current state.

- **getRouteAddress**()

  Returns the `RouteAddress` associated with this routing session and the one for which the application has registered to route Calls for.

- **getState**()

  Returns the current state of the route session.

- **selectRoute**(String[])

  Selects one or more possible destinations for the routing of the Call.

Variables

ROUTE

```
public static final int ROUTE
```

The `RouteSession.ROUTE` state indicates that an application has been requested to route a Call.

ROUTE_USED

```
public static final int ROUTE_USED
```

The `RouteSession.ROUTE_USED` state indicates that a destination has been selected for a Call. This destination is one which the application had selected during its routing of the Call.

ROUTE_END

```
public static final int ROUTE_END
```

The `RouteSession.ROUTE_END` state indicates that the routing of a Call has terminated.

RE_ROUTE

```
public static final int RE_ROUTE
```

The `RouteSession.RE_ROUTE` state indicates that an application has been requested to select another destination for the Call.

ROUTE_CALLBACK_ENDED

```
public static final int ROUTE_CALLBACK_ENDED
```

The `RouteSession.ROUTE_CALLBACK_ENDED` state indicates that all `RouteCallback` objects have been removed from this routing session. There are no more routing callbacks associated with this session. This is the final state for the `RouteSession` interface.

CAUSE_NO_ERROR

```
public static final int CAUSE_NO_ERROR
```

Cause code indicating no error.

CAUSE_ROUTING_TIMER_EXPIRED

```
public static final int CAUSE_ROUTING_TIMER_EXPIRED
```

Cause code indicating a routing timer has expired.

CAUSE_PARAMETER_NOT_SUPPORTED

```
public static final int CAUSE_PARAMETER_NOT_SUPPORTED
```

Cause code indicating that the routing session is being terminated because the application supplied an unsupported parameter in the `RouteSession.routeSelect()` method.

CAUSE_INVALID_DESTINATION

```
public static final int CAUSE_INVALID_DESTINATION
```

Cause code indicating that the routing session is being terminated because because the application supplied an invalid destination in the `RouteSession.routeSelect()` method.

CAUSE_STATE_INCOMPATIBLE

```
public static final int CAUSE_STATE_INCOMPATIBLE
```

Cause code indicating that the routing session is being terminated because the `Connection` state is incompatable with the `RouteSession`.

CAUSE_UNSPECIFIED_ERROR

```
public static final int CAUSE_UNSPECIFIED_ERROR
```

Cause code indicating that the routing session is being terminated for unspecified reasons.

ERROR_UNKNOWN

```
public static final int ERROR_UNKNOWN
```

Error code indicating the application is not giving a reason for ending the routing session. This value may be passed as an argument to the `RouteSession.endRoute()` method.

ERROR_RESOURCE_BUSY

```
public static final int ERROR_RESOURCE_BUSY
```

Error code indicating the application is too busy to handle the routing request. This value may be passed as an argument to the `RouteSession.endRoute()` method.

ERROR_RESOURCE_OUT_OF_SERVICE

```
public static final int ERROR_RESOURCE_OUT_OF_SERVICE
```

Error code indicating the application or a database upon which it relies for routing is temporary out of service and cannot handle the routing request. This value may be passed as an argument to the `RouteSession.endRoute()` method.

Methods

getRouteAddress

```
public abstract RouteAddress getRouteAddress()
```

Returns the `RouteAddress` associated with this routing session and the one for which the application has registered to route Calls for.

Returns:

The RouteAddress associated with this session.

selectRoute

```
public abstract void selectRoute(String routeSelected[]) throws
MethodNotSupportedException
```

Selects one or more possible destinations for the routing of the Call. This method takes an array of string destination telephone address names, in priority order. The highest priority destination is the first element in the given array, and routing is attempted with this destination first. Successive given destination addresses are attempted until one is found which does not fail.

A `RouteUsedEvent` event is delivered to the application when a successful routing destination has been selected and the Call has been routed to that destination.

Pre-conditions:

1. this.getRouteAddress().getProvider().getState() == Provider.IN_SERVICE
2. this.getState() == RouteSession.ROUTE or RouteSession.RE_ROUTE

Post-Conditions

1. this.getRouteAddress().getProvider().getState() == Provider.IN_SERVICE
2. this.getState() == RouteSession.ROUTE_USED if Call was successfully routed.
3. RouteUsedEvent is delivered for this RouteSession if a successful destination was selected.

Parameters:

routeSelected - A list of possible destinations for the call.

Throws: MethodNotSupportedException

Routing is not supported by the implementation.

endRoute

```
public abstract void endRoute(int errorValue) throws MethodNotSupportedException
```

Ends a routing session. The application provides an integer error value argument giving the reason why it is terminating this routing session. These error values are defined by this interface.

If this method is successful, the state of this `RouteSession` moves into the `RouteSession.ROUTE_END` state and a `RouteEndEvent` is delivered.

Pre-Conditions

1. this.getRouteAddress().getProvider().getState() == Provider.IN_SERVICE

Post-Conditions

1. this.getRouteAddress().getProvider().getState() == Provider.IN_SERVICE
2. this.getState() == RouteSession.ROUTE_END
3. RouteEndEvent is delivered to the application for this RouteSession

Parameters:

errorValue - Indicates the reason why the application is terminating this routing session, as defined by the constants in this interface.

Throws: MethodNotSupportedException

The implementation does not support this method.

getState

```
public abstract int getState()
```

Returns the current state of the route session.

Returns:

The current state of the route session.

getCause

```
public abstract int getCause()
```

Returns the cause indicating why this route session is in its current state. These cause values are constant integer definitions defined by this interface.

Returns:

The cause of the current route session state.

All Packages This Package Previous Next

All Packages

package javax.telephony.callcenter.capabilities

Interface Index

- ACDAddressCapabilities
- ACDConnectionCapabilities
- ACDManagerAddressCapabilities
- ACDManagerConnectionCapabilities
- AgentTerminalCapabilities
- CallCenterAddressCapabilities
- CallCenterCallCapabilities
- CallCenterProviderCapabilities
- RouteAddressCapabilities

Interface javax.telephony.callcenter.capabilities.ACDAddressCapabilities

public interface **ACDAddressCapabilities**

extends AddressCapabilities

The ACDAddressCapabilities interface extends the AddressCapabilities interface to add capabilities methods for the ACDAddress interface. Applications query these methods to find out what actions are possible on the ACDAddress interface.

Method Index

- **canGetACDManagerAddress**()

  This method returns true if the method getACDManagerAddress on the ACDAddress interface is supported.

- **canGetLoggedOnAgents**()

  This method returns true if the method getLoggedOnAgents on the ACDAddress interface is supported.

- **canGetNumberQueued**()

  This method returns true if the method getNumberQueued on the ACDAddress interface is supported.

- **canGetOldestCallQueued**()

  This method returns true if the method getOldestCallQueued on the ACDAddress interface is supported.

- **canGetQueueWaitTime**()

  This method returns true if the method getQueueWaitTime on the ACDAddress interface is supported.

- **canGetRelativeQueueLoad**()

  This method returns true if the method getRelativeQueueLoad on the ACDAddress interface is supported.

Methods

canGetLoggedOnAgents

```
 public abstract boolean canGetLoggedOnAgents()
```

This method returns true if the method getLoggedOnAgents on the ACDAddress interface is supported.

Returns:

True if the method getLoggedOnAgents on the ACDAddress interface is supported.

canGetNumberQueued

```
 public abstract boolean canGetNumberQueued()
```

This method returns true if the method getNumberQueued on the ACDAddress interface is supported.

Returns:

True if the method getNumberQueued on the ACDAddress interface is supported.

canGetOldestCallQueued

```
 public abstract boolean canGetOldestCallQueued()
```

This method returns true if the method getOldestCallQueued on the ACDAddress interface is supported.

Returns:

True if the method getOldestCallQueued on the ACDAddress interface is supported.

canGetRelativeQueueLoad

```
public abstract boolean canGetRelativeQueueLoad()
```

This method returns true if the method getRelativeQueueLoad on the ACDAddress interface is supported.

Returns:

True if the method getRelativeQueueLoad on the ACDAddress interface is supported.

canGetQueueWaitTime

```
public abstract boolean canGetQueueWaitTime()
```

This method returns true if the method getQueueWaitTime on the ACDAddress interface is supported.

Returns:

True if the method getQueueWaitTime on the ACDAddress interface is supported.

canGetACDManagerAddress

```
public abstract boolean canGetACDManagerAddress()
```

This method returns true if the method getACDManagerAddress on the ACDAddress interface is supported.

Returns:

True if the method getACDManagerAddress on the ACDAddress interface is supported.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.capabilities.ACDConnectionCapabilities

public interface **ACDConnectionCapabilities**

extends ConnectionCapabilities

The ACDConnectionCapabilities interface extends the ConnectionCapabilities interface to add capabilities methods for the ACDConnection interface. Applications query these methods to find out what actions are possible on the ACDConnection interface.

Method Index

- **canGetACDManagerConnection**()

  This method returns true if the method getACDManagerConnection on the ACDConnection interface is supported.

Methods

canGetACDManagerConnection

```
public abstract boolean canGetACDManagerConnection()
```

This method returns true if the method getACDManagerConnection on the ACDConnection interface is supported.

Returns:

True if the method getACDManagerConnection on the ACDConnection interface is supported.

Interface javax.telephony.callcenter.capabilities.ACDManagerAddressCapabilities

public interface **ACDManagerAddressCapabilities**

extends AddressCapabilities

The ACDManagerAddressCapabilities interface extends the AddressCapabilities interface to add capabilities methods for the ACDManagerAddress interface. Applications query these methods to find out what actions are possible on the ACDAManagerddress interface.

Method Index

- **canGetACDAddresses**()

  This method returns true if the method getACDAddresses on the ACDManagerAddress interface is supported.

Methods

canGetACDAddresses

```
 public abstract boolean canGetACDAddresses()
```

This method returns true if the method getACDAddresses on the ACDManagerAddress interface is supported.

Returns:

True if the method getACDAddresses on the ACDManagerAddress interface is supported.

Interface javax.telephony.callcenter.capabilities.ACDManagerConnectionCapabilities

public interface **ACDManagerConnectionCapabilities**

extends ConnectionCapabilities

The ACDManagerConnectionCapabilities interface extends the ConnectionCapabilities interface to add capabilities methods for the ACDManagerConnection interface. Applications query these methods to find out what actions are possible on the ACDManagerConnection interface.

Method Index

- **canGetACDConnections**()

  This method returns true if the method getACDConnections on the ACDManagerConnection interface is supported.

Methods

canGetACDConnections

```
 public abstract boolean canGetACDConnections()
```

This method returns true if the method getACDConnections on the ACDManagerConnection interface is supported.

Returns:

True if the method getACDConnections on the ACDManagerConnection interface is supported.

Interface javax.telephony.callcenter.capabilities.AgentTerminalCapabilities

public interface **AgentTerminalCapabilities**

extends TerminalCapabilities

The AgentTerminalCapabilities interface extends the TerminalCapabilities interface to add capabilities methods for the AgentTerminal interface. Applications query these methods to find out what actions are possible on the AgentTerminal interface.

Method Index

- **canHandleAgents**()

  This method returns true if the methods addAgent, removeAgent and getAgents on the AgentTerminal interface are supported.

Methods

● **canHandleAgents**

```
public abstract boolean canHandleAgents()
```

This method returns true if the methods addAgent, removeAgent and getAgents on the AgentTerminal interface are supported.

Returns:

True if the methods to handle agents on the AgentTerminal interface are supported.

Interface javax.telephony.callcenter.capabilities.CallCenterAddressCapabilities

public interface **CallCenterAddressCapabilities**

extends AddressCapabilities

The CallCenterAddressCapabilities interface extends the AddressCapabilities interface to add capabilities methods for the CallCenterAddress interface. Applications query these methods to find out what actions are possible on the CallCenterAddress interface.

Method Index

canAddCallObserver(boolean)

This method returns true if the method addCallObserver with the remain flag on the CallCenterAddress interface is supported.

Methods

canAddCallObserver

```
public abstract boolean canAddCallObserver(boolean remain)
```

This method returns true if the method addCallObserver with the remain flag on the CallCenterAddress interface is supported.

Returns:

True if the method addCallObserver on the CallCenterAddress interface is supported.

Interface javax.telephony.callcenter.capabilities.CallCenterCallCapabilities

public interface **CallCenterCallCapabilities**

extends CallCapabilities

The CallCenterCallCapabilities interface extends the CallCapabilities interface to add capabilities methods for the CallCenterCall interface. Applications query these methods to find out what actions are possible on the CallCenterCall interface.

Method Index

canConnectPredictive()

This method returns true if the method connectPredictive on the CallCenterCall interface is supported.

canGetTrunks()

This method returns true if the method getTrunks on the CallCenterCall interface is supported.

canHandleApplicationData()

This method returns true if the methods setApplicationData and getApplicationData on the CallCenterCall interface are supported.

Methods

canConnectPredictive

```
public abstract boolean canConnectPredictive()
```

This method returns true if the method connectPredictive on the CallCenterCall interface is supported.

Returns:

True if the method connectPredictive on the CallCenterCall interface is supported.

canHandleApplicationData

```
public abstract boolean canHandleApplicationData()
```

This method returns true if the methods setApplicationData and getApplicationData on the CallCenterCall interface are supported.

Returns:

True if the methods to handle ApplicationData on the CallCenterCall interface are supported.

canGetTrunks

```
public abstract boolean canGetTrunks()
```

This method returns true if the method getTrunks on the CallCenterCall interface is supported.

Returns:

True if the method getTrunks on the CallCenterCall interface is supported.

Interface javax.telephony.callcenter.capabilities.CallCenterProviderCapabilities

public interface **CallCenterProviderCapabilities**

extends ProviderCapabilities

The CallCenterProviderCapabilities interface extends the ProviderCapabilities interface to add capabilities methods for the CallCenterProvider interface. Applications query these methods to find out what actions are possible on the CallCenterProvider interface.

Method Index

- **canGetACDAddresses**()

  This method returns true if the method getACDAddresses on the CallCenterProvider interface is supported.

- **canGetACDManagerAddresses**()

  This method returns true if the method getACDManagerAddresses on the CallCenterProvider interface is supported.

- **canGetRouteableAddresses**()

  This method returns true if the method getRouteableAddresses on the CallCenterProvider interface is supported.

Methods

canGetRouteableAddresses

```
public abstract boolean canGetRouteableAddresses()
```

This method returns true if the method getRouteableAddresses on the CallCenterProvider interface is supported.

Returns:

True if the method addCallObserver on the CallCenterProvider interface is supported.

canGetACDAddresses

```
public abstract boolean canGetACDAddresses()
```

This method returns true if the method getACDAddresses on the CallCenterProvider interface is supported.

Returns:

True if the method getACDAddresses on the CallCenterProvider interface is supported.

canGetACDManagerAddresses

```
public abstract boolean canGetACDManagerAddresses()
```

This method returns true if the method getACDManagerAddresses on the CallCenterProvider interface is supported.

Returns:

True if the method CallCenterProvodier.getACDManagerAddresses() is supported.

Interface javax.telephony.callcenter.capabilities.RouteAddressCapabilities

public interface **RouteAddressCapabilities**

extends AddressCapabilities

The RouteAddressCapabilities interface extends the AddressCapabilities interface to add capabilities methods for the RouteAddress interface. Applications query these methods to find out what actions are possible on the RouteAddress interface.

Method Index

● **canRouteCalls**()

This method returns true if the methods registerRouteCallback, cancelRouteCallback, getRouteCallback and getActiveRouteSessions on the RouteAddress interface are supported.

Methods

● canRouteCalls

```
public abstract boolean canRouteCalls()
```

This method returns true if the methods registerRouteCallback, cancelRouteCallback, getRouteCallback and getActiveRouteSessions on the RouteAddress interface are supported.

Returns:

True if the methods for routing on the RouteAddress interface are supported.

All Packages

package javax.telephony.callcenter.events

Interface Index

- ACDAddrBusyEv
- ACDAddrEv
- ACDAddrLoggedOffEv
- ACDAddrLoggedOnEv
- ACDAddrNotReadyEv
- ACDAddrReadyEv
- ACDAddrUnknownEv
- ACDAddrWorkNotReadyEv
- ACDAddrWorkReadyEv
- AgentTermBusyEv
- AgentTermEv
- AgentTermLoggedOffEv
- AgentTermLoggedOnEv
- AgentTermNotReadyEv
- AgentTermReadyEv
- AgentTermUnknownEv
- AgentTermWorkNotReadyEv
- AgentTermWorkReadyEv
- CallCentCallAppDataEv
- CallCentCallEv
- CallCentConnEv
- CallCentConnInProgressEv
- CallCentEv
- CallCentTrunkEv
- CallCentTrunkInvalidEv
- CallCentTrunkValidEv
- ReRouteEvent
- RouteCallbackEndedEvent
- RouteEndEvent
- RouteEvent
- RouteSessionEvent
- RouteUsedEvent

Interface javax.telephony.callcenter.events.ACDAddrBusyEv

public interface **ACDAddrBusyEv**

extends ACDAddrEv

The `ACDAddrBusyEv` interface indicates that an Agent has moved into the `Agent.BUSY` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrEv

public interface **ACDAddrEv**

extends CallCentEv, AddrEv

The `ACDAddrEv` interfaces is the base event for all events pertaining to the `ACDAddress` interface. This interface extends the `CallCentEv` interface and the core `AddrEv` interface. All event interfaces which extend this interface are reported via the `ACDAddressObserver` interface.

The call center package defines events interfaces which extend this interface to report state changes in `Agent's` which are associated with the `ACDAddress`. These events are: `ACDAddrBusyEv`, `ACDAddrLoggedOffEv`, `ACDAddrLoggedOnEv`, `ACDAddrNotReadyEv`, `ACDAddrUnknownEv`, `ACDAddrWorkNotReadyEv`, and `ACDAddrWorkReadyEv`.

See Also:

Terminal, TermEv, Agent, ACDAddress, CallCentEv, ACDAddrBusyEv, ACDAddrLoggedOnEv, ACDAddrLoggedOffEv, ACDAddrNotReadyEv, ACDAddrReadyEv, ACDAddrUnknownEv, ACDAddrWorkNotReadyEv, ACDAddrWorkReadyEv

Method Index

getAgent()

Returns the `Agent` associated with this event.

getAgentAddress()

Returns the Address associated with the `Agent's` Terminal. **Deprecated.**

getAgentTerminal()

Returns the Terminal associated with the `Agent`. **Deprecated.**

getState()

Returns the state of the `Agent`. **Deprecated.**

getTrunks()

Returns an array of all Trunks currently being used for this Call. **Deprecated.**

Methods

getAgent

```
public abstract Agent getAgent()
```

Returns the `Agent` associated with this event.

Returns:

The associated Agent.

getAgentTerminal

```
public abstract AgentTerminal getAgentTerminal()
```

Note: getAgentTerminal() is deprecated. *JTAPI v1.2. This method has been replaced by the getAgent() method.*

Returns the Terminal associated with the Agent.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the ACDAddrEv.getAgent() method to obtain the Agent and then use the Agent.getAgentTerminal() method.

Returns:

The Terminal associated with the Agent.

getState

```
public abstract int getState()
```

Note: getState() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the state of the Agent.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the ACDAddrEv.getAgent() method to obtain the Agent and then use the Agent.getState() method.

Returns:

The Agent's state.

getAgentAddress

```
public abstract Address getAgentAddress()
```

Note: getAgentAddress() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the Address associated with the Agent's Terminal.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the ACDAddrEv.getAgent() method to obtain the Agent and then use the Agent.getAgentAddress() method.

Returns:

The Address associated with the Agent's Terminal.

getTrunks

```
public abstract CallCenterTrunk[] getTrunks()
```

Note: getTrunks() is deprecated. *JTAPI v1.2. This method has been replaced by the getAgent() method.*

Returns an array of all Trunks currently being used for this Call. If there are no Trunks being used, this method returns null.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the ACDAddrEv.getAgent() method to obtain the Agent and then get the trunks via the CallCenterCall interface.

Returns:

An array of Trunks, null if there are none.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.ACDAddrLoggedOffEv

public interface **ACDAddrLoggedOffEv**

extends ACDAddrEv

The `ACDAddrLoggedOffEv` interface indicates that an Agent has moved into the `Agent.LOG_OFF` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

- **ID**

  Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrLoggedOnEv

public interface **ACDAddrLoggedOnEv**

extends ACDAddrEv

The `ACDAddrLoggedOnEv` interface indicates that an Agent has moved into the `Agent.LOG_ON` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

- **ID**

  Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrNotReadyEv

public interface **ACDAddrNotReadyEv**

extends ACDAddrEv

The `ACDAddrNotReadyEv` interface indicates that an Agent has moved into the `Agent.NOT_READY` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrReadyEv

public interface **ACDAddrReadyEv**

extends ACDAddrEv

The `ACDAddrReadyEv` interface indicates that an Agent has moved into the `Agent.READY` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrUnknownEv

public interface **ACDAddrUnknownEv**

extends ACDAddrEv

The `ACDAddrUnknownEv` interface indicates that an Agent has moved into the `Agent.UNKNOWN` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrWorkNotReadyEv

public interface **ACDAddrWorkNotReadyEv**

extends ACDAddrEv

The `ACDAddrWorkNotReadyEv` interface indicates that an Agent has moved into the `Agent.WORK_NOT_READY` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

- **ID**

  Event id

Variables

● ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ACDAddrWorkReadyEv

public interface **ACDAddrWorkReadyEv**

extends ACDAddrEv

The `ACDAddrWorkReadyEv` interface indicates that an Agent has moved into the `Agent.WORK_READY` state. This interface extends the `ACDAddrEv` interface and is reported via the `ACDAddressObserver` interface for the `ACDAddress` associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, ACDAddressObserver, ACDAddrEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermBusyEv

public interface **AgentTermBusyEv**

extends AgentTermEv

The `AgentTermBusyEv` interface indicates that an Agent has moved into the `Agent.BUSY` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

- **ID**

  Event id

Variables

● ID

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermEv

public interface **AgentTermEv**

extends CallCentEv, TermEv

The `AgentTermEv` interfaces is the base event for all events pertaining to the `AgentTerminal` interface. This interface extends the `CallCentEv` interface and the core `TermEv` interface. All event interfaces which extend this interface are reported via the `AgentTerminalObserver` interface.

The call center package defines events interfaces which extend this interface to report state changes in the `Agent`. These events are: `AgentTermBusyEv`, `AgentTermLoggedOffEv`, `AgentTermLoggedOnEv`, `AgentTermNotReadyEv`, `AgentTermUnknownEv`, `AgentTermWorkNotReadyEv`, and `AgentTermWorkReadyEv`.

See Also:

Terminal, TermEv, Agent, AgentTerminal, CallCentEv, AgentTermBusyEv, AgentTermLoggedOnEv, AgentTermLoggedOffEv, AgentTermNotReadyEv, AgentTermReadyEv, AgentTermUnknownEv, AgentTermWorkNotReadyEv, AgentTermWorkReadyEv

Method Index

- **getACDAddress**()

  Returns the `ACDAddress` the agent currently is or was logged into. **Deprecated.**
- **getAgent**()

  Returns the `Agent` associated with the `AgentTerminal`.
- **getAgentAddress**()

  Returns the Address associated with the `Agent's` Terminal. **Deprecated.**
- **getAgentID**()

  Returns the ID of the `Agent`. **Deprecated.**
- **getState**()

  Returns the state of the `Agent`. **Deprecated.**

Methods

getAgent

```
 public abstract Agent getAgent()
```

Returns the `Agent` associated with the `AgentTerminal`.

Returns:

The associated Agent.

getACDAddress

```
 public abstract ACDAddress getACDAddress()
```

Note: getACDAddress() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the `ACDAddress` the agent currently is or was logged into.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the `AgentTermEv.getAgent()` method to obtain the `Agent` and then use the `Agent.getACDAddress()` method.

Returns:

The ACDAddress currently or formerly associated with the Agent.

getAgentID

```
public abstract String getAgentID()
```

Note: getAgentID() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the ID of the `Agent`.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the `AgentTermEv.getAgent()` method to obtain the `Agent` and then use the `Agent.getAgentID()` method.

Returns:

The Agent ID.

getState

```
public abstract int getState()
```

Note: getState() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the state of the `Agent`.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the `AgentTermEv.getAgent()` method to obtain the `Agent` and then use the `Agent.getState()` method.

Returns:

The Agent's state.

getAgentAddress

```
public abstract Address getAgentAddress()
```

Note: getAgentAddress() is deprecated. *JTAPI v1.2 This method has been replaced by the getAgent() method.*

Returns the Address associated with the `Agent's` Terminal.

Note: This method has been replaced in JTAPI v1.2 and later. Applications should use the `AgentTermEv.getAgent()` method to obtain the `Agent` and then use the `Agent.getAgentAddress()` method.

Returns:

The Address associated with the Agent's Terminal.

All Packages   This Package   Previous   Next

Interface javax.telephony.callcenter.events.AgentTermLoggedOffEv

public interface **AgentTermLoggedOffEv**

extends AgentTermEv

The `AgentTermLoggedOffEv` interface indicates that an Agent has moved into the `Agent.LOG_OFF` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

● **ID**

Event id

Variables

● ID

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermLoggedOnEv

public interface **AgentTermLoggedOnEv**

extends AgentTermEv

The `AgentTermLoggedOnEv` interface indicates that an Agent has moved into the `Agent.LOG_IN` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

- **ID**

  Event id

Variables

● ID

```
 public static final int ID
```

Event id

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.AgentTermNotReadyEv

public interface **AgentTermNotReadyEv**

extends AgentTermEv

The `AgentTermNotReadyEv` interface indicates that an Agent has moved into the `Agent.NOT_READY` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.AgentTermReadyEv

public interface **AgentTermReadyEv**

extends AgentTermEv

The `AgentTermReadyEv` interface indicates that an Agent has moved into the `Agent.READY` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermUnknownEv

public interface **AgentTermUnknownEv**

extends AgentTermEv

The `AgentTermUnknownEv` interface indicates that an Agent has moved into the `Agent.UNKNOWN` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

- **ID**

  Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermWorkNotReadyEv

public interface **AgentTermWorkNotReadyEv**

extends AgentTermEv

The `AgentTermWorkNotReadyEv` interface indicates that an Agent has moved into the `Agent.WORK_NOT_READY` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.AgentTermWorkReadyEv

public interface **AgentTermWorkReadyEv**

extends AgentTermEv

The `AgentTermWorkReadyEv` interface indicates that an Agent has moved into the `Agent.WORK_READY` state. This interface extends the `AgentTermEv` interface and is reported via the `AgentTerminalObserver` interface for the Terminal associated with the Agent.

This interface defines no additional methods.

See Also:

Terminal, TerminalObserver, Agent, AgentTerminalObserver, AgentTermEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.CallCentCallAppDataEv

public interface **CallCentCallAppDataEv**

extends CallCentCallEv

The `CallCentCallAppDataEv` event interfaces indicates that the application data associated with the Call has changed. This interface extends the `CallCentCallEv` interface and is reported via the `CallCenterCallObserver` interface.

The `CallCentCallAppDataEv.getApplicationData()` method returns the new application data.

See Also:

Call, CallCenterCall, CallCenterCallObserver, CallCentCallEv

Variable Index

ID

Event id.

Method Index

getApplicationData()

Returns the new application data for this call.

Variables

ID

```
public static final int ID
```

Event id.

Methods

getApplicationData

```
public abstract Object getApplicationData()
```

Returns the new application data for this call. This method returns null if the application data has been cleared from the call.

Returns:

The data object, null if it has been cleared.

Interface javax.telephony.callcenter.events.CallCentCallEv

public interface **CallCentCallEv**

extends CallCentEv, CallEv

The `CallCentCallEv` interface is the base event interface for all call center package Call-related events. Each Call-related event defined in this package must extend this interface. This interface extends both the core `CallEv` and the `CallCentEv` interfaces. All events which extend this interface are reported via the `CallCenterCallObserver` interface.

Additional Call Information

This interface supports methods which return additional information regarding the telephone call. Specifically, it returns the *calling address*, *calling terminal*, *called address*, and *last redirected address* information. This information is returned by the `getCallingAddress()`, `getCallingTerminal()`, `getCalledAddress()`, and `getLastRedirectedAddress()` methods on this interface, respectively.

The call center package defines the following interfaces which extend this interface: `CallCentConnEv`, `CallCentAppDataEv`, and `CallCentTrunkEv`

See Also:

Call, Address, Terminal, CallEv, CallCenterCallObserver, CallCenterCall, CallCenterTrunk, CallCentEv, CallCentConnEv, CallCentTrunkEv, CallCentAppDataEv

Method Index

- **getCalledAddress**()

  Returns the called Address associated with this Call.
- **getCallingAddress**()

  Returns the calling Address associated with this call.
- **getCallingTerminal**()

  Returns the calling Terminal associated with this Call.
- **getLastRedirectedAddress**()

  Returns the last redirected Address associated with this Call.
- **getTrunks**()

  Returns an array of all Trunks currently being used for this Call. **Deprecated.**

Methods

getCallingAddress

```
 public abstract Address getCallingAddress()
```

Returns the calling Address associated with this call. The calling Address is defined as the Address which placed the telephone call.

If the calling address is unknown or not yet known, this method returns null.

Returns:

The calling Address.

getCallingTerminal

```
public abstract Terminal getCallingTerminal()
```

Returns the calling Terminal associated with this Call. The calling Terminal is defined as the Terminal which placed the telephone call.

If the calling Terminal is unknown or not yet know, this method returns null.

Returns:

The calling Terminal.

getCalledAddress

```
public abstract Address getCalledAddress()
```

Returns the called Address associated with this Call. The called Address is defined as the Address to which the call has been originally placed.

If the called address is unknown or not yet known, this method returns null.

Returns:

The called Address.

getLastRedirectedAddress

```
public abstract Address getLastRedirectedAddress()
```

Returns the last redirected Address associated with this Call. The last redirected Address is the Address at which the current telephone call was placed immediately before the current Address. This is common if a Call is forwarded to several Addresses before being answered.

If the the last redirected address is unknown or not yet known, this method returns null.

Returns:

The last redirected Address for this telephone Call.

getTrunks

```
public abstract CallCenterTrunk[] getTrunks()
```

Note: getTrunks() is deprecated. *JTAPI v1.2*

Returns an array of all Trunks currently being used for this Call. If there are no Trunks being used, this method returns null.

Note: This method has been replaced in JTAPI v1.2 and later with the `CallCenterCall.getTrunks()` method.

Returns:

An array of Trunks, null if there are none.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.CallCentConnEv

public interface **CallCentConnEv**

extends CallCentCallEv, ConnEv

The `CallCentConnEv` is the base event interface for all call center Connection events. Each Connection-related event in this package must extend this interface. This interface extends both the `ConnEv` interface and the `CallCentCallEv` interface. Events which extend this interface are reported via the `CallCenterCallObserver` interface.

Currently, this package defines the `CallCentConnInProgressEv` event interface which extends the core event to provide additional call center-related information when the Connection moves into this state.

See Also:

CallCenterCallObserver, CallCentConnInProgressEv

Interface javax.telephony.callcenter.events.CallCentConnInProgressEv

public interface **CallCentConnInProgressEv**

extends CallCentConnEv

The `CallCentConnInProgressEv` indicates that the Connection has moved into the `Connection.INPROGRESS` state. This interface extends the `CallCentConnEv` interface and is reported via the `CallCenterCallObserver` interface.

The call center package extends the core `ConnInProgressEv` event interface to provide additional information to the application for call center-specific purposed.

See Also:

ConnInProgressEv, CallCenterCallObserver, CallCentConnEv

Variable Index

● **ID**

Event id.

Variables

● **ID**

```
public static final int ID
```

Event id.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.CallCentEv

public interface **CallCentEv**

extends Ev

The `CallCentEv` interface is the base event interface for all call center package events. This interface extends the core `Ev` interface. All call center package events must extend this interface.

This interface contains a single method `getCallCenterCause()` which returns the call center package-specific cause value for the event.

The call center package defines the following interfaces which directly extend this interface: `ACDAddrEv`, `AgentTermEv`, and `CallCenterCallEv`.

See Also:

Ev, ACDAddrEv, AgentTermEv, CallCenterCallEv

Variable Index

CAUSE_NO_AVAILABLE_AGENTS

This cause indicates no agents were available to handle the call.

Method Index

getCallCenterCause()

Returns the call center package cause value for this event.

Variables

CAUSE_NO_AVAILABLE_AGENTS

```
public static final int CAUSE_NO_AVAILABLE_AGENTS
```

This cause indicates no agents were available to handle the call.

Methods

getCallCenterCause

```
public abstract int getCallCenterCause()
```

Returns the call center package cause value for this event. This method may also return the `Ev.CAUSE_NORMAL` constant or the

`Ev.CAUSE_UNKNOWN` constant.

Returns:

The call center package cause for the event.

Interface javax.telephony.callcenter.events.CallCentTrunkEv

public interface **CallCentTrunkEv**

extends CallCentCallEv

The `CallCentTrunkEv` interface is the base event interface for all `CallCenterTrunk`-related events in the call center package. Every `CallCenterTrunk`-related event must extend this interface. This event extends the `CallCentCallEv` interface and all events which extend this interface are reported via the `CallCenterCallObserver` interface.

This interface contains a single method, `getTrunk()`, which returns the `CallCenterTrunk` associated with this event.

The call center package defines two event interfaces which extends this interface. They are the `CallCentTrunkValidEv` interface and the `CallCentTrunkInvalidEv` interface. These interfaces report a state change in the `CallCenterTrunk`.

See Also:

Call, CallObserver, CallCenterCallObserver, CallCenterTrunk, CallCentCallEv, CallCentTrunkValidEv, CallCentTrunkInvalidEv

Method Index

● **getTrunk**()

Returns the `CallCenterTrunk` associated with this event.

Methods

● **getTrunk**

```
 public abstract CallCenterTrunk getTrunk()
```

Returns the `CallCenterTrunk` associated with this event.

Returns:

The associated CallCenterTrunk.

Interface javax.telephony.callcenter.events.CallCentTrunkInvalidEv

public interface **CallCentTrunkInvalidEv**

extends CallCentTrunkEv

The `CallCentTrunkInvalidEv` interface indicates that a Trunk has moved into the `CallCenterTrunk.INVALID_TRUNK` state. This interface extends the `CallCentTrunkEv` interface and is reported via the `CallCenterCallObserver` interface.

This interface defines no additional methods.

See Also:

Call, CallObserver, CallCenterTrunk, CallCenterObserver, CallCentTrunkEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.CallCentTrunkValidEv

public interface **CallCentTrunkValidEv**

extends CallCentTrunkEv

The `CallCentTrunkValidEv` interface indicates that a Trunk has moved into the `CallCenterTrunk.VALID_TRUNK` state. This interface extends the `CallCentTrunkEv` interface and is reported via the `CallCenterCallObserver` interface.

This interface defines no additional methods.

See Also:

Call, CallObserver, CallCenterTrunk, CallCenterObserver, CallCentTrunkEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcenter.events.ReRouteEvent

public interface **ReRouteEvent**

extends RouteSessionEvent

The `ReRouteEvent` event interface indicates the `RouteSession` interface has moved into the `RouteSession.RE_ROUTE` state and the Provider is requesting the application select another route for a Call. This interface extends the `RouteSessionEvent` interface and is reported via the `RouteCallback` interface.

See Also:

RouteSession, RouteCallback, RouteSessionEvent

Interface javax.telephony.callcenter.events.RouteCallbackEndedEvent

public interface **RouteCallbackEndedEvent**

The `RouteCallbackEndedEvent` event interface indicates the `RouteSession` interface has moved into the `RouteSession.ROUTE_CALLBACK_ENDED` state and the registration of a routing callback has ended. This event is reported via the `RouteCallback` interface.

See Also:

RouteSession, RouteCallback

Method Index

● **getRouteAddress**()

Returns the `RouteAddress` that is associated with the `RouteSession` associated with this event.

Methods

● **getRouteAddress**

```
 public abstract RouteAddress getRouteAddress()
```

Returns the `RouteAddress` that is associated with the `RouteSession` associated with this event.

Returns:

The RouteAddress associated with the RouteSession.

Interface javax.telephony.callcenter.events.RouteEndEvent

public interface **RouteEndEvent**

extends RouteSessionEvent

The `RouteEndEvent` event interface indicates the `RouteSession` interface has moved into the `RouteSession.ROUTE_END` state and the routing of the Call has terminated. This interface extends the `RouteSessionEvent` interface and is reported via the `RouteCallback` interface.

The reason for the termination of routing may be determined via the `RouteSessionEvent.getRouteSession().getCause()` method.

See Also:

RouteSession, RouteCallback, RouteSessionEvent

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.RouteEvent

public interface **RouteEvent**

extends RouteSessionEvent

The `RouteEvent` event interface indicates the `RouteSession` interface has moved into the `RouteSession.ROUTE` state and the Provider is requesting the application route a Call. This interface extends the `RouteSessionEvent` interface and is reported via the `RouteCallback` interface.

See Also:

RouteSession, RouteCallback, RouteSessionEvent

Variable Index

- **SELECT_ACD**

  Route Selection Algorithm: Select a route to an ACDAddress.
- **SELECT_EMERGENCY**

  Route Selection Algorithm: Select an emergency route.
- **SELECT_LEAST_COST**

  Route Selection Algorithm: Select a least cost route.
- **SELECT_NORMAL**

  Route Selection Algorithm: Select a normal route.
- **SELECT_USER_DEFINED**

  Route Selection Algorithm: Select a user defined route.

Method Index

- **getCallingAddress**()

  Returns the calling Address.
- **getCallingTerminal**()

  Returns the calling Terminal.
- **getCurrentRouteAddress**()

  Returns the originally requested destination for the call.
- **getRouteSelectAlgorithm**()

  Returns the route select algorithm being used.
- **getSetupInformation**()

  Returns the ISDN call setup message when available.

Variables

SELECT_NORMAL

```
public static final int SELECT_NORMAL
```

Route Selection Algorithm: Select a normal route.

SELECT_LEAST_COST

```
public static final int SELECT_LEAST_COST
```

Route Selection Algorithm: Select a least cost route.

SELECT_EMERGENCY

```
public static final int SELECT_EMERGENCY
```

Route Selection Algorithm: Select an emergency route.

SELECT_ACD

```
public static final int SELECT_ACD
```

Route Selection Algorithm: Select a route to an ACDAddress.

SELECT_USER_DEFINED

```
public static final int SELECT_USER_DEFINED
```

Route Selection Algorithm: Select a user defined route.

Methods

getCurrentRouteAddress

```
public abstract RouteAddress getCurrentRouteAddress()
```

Returns the originally requested destination for the call.

Returns:

The originally request destination for the call.

getCallingAddress

```
public abstract Address getCallingAddress()
```

Returns the calling Address.

Returns:

The calling Address.

getCallingTerminal

```
public abstract Terminal getCallingTerminal()
```

Returns the calling Terminal.

Returns:

The calling Terminal.

● getRouteSelectAlgorithm

```
public abstract int getRouteSelectAlgorithm()
```

Returns the route select algorithm being used.

Returns:

The route selection algorithm being used.

● getSetupInformation

```
public abstract String getSetupInformation()
```

Returns the ISDN call setup message when available.

Returns:

The ISDN call setup message.

Interface javax.telephony.callcenter.events.RouteSessionEvent

public interface **RouteSessionEvent**

The `RouteSessionEvent` interface is the base event interface for all events pertaining to the `RouteSession` interface. Events which extend this interface are not part of the JTAPI observer mechanism. All events which extend this interface are reported via the `RouteCallback` interface.

The call center package defines event interface which extend this interface to report changes in the state of the `RouteSession` interface. These interfaces are: `RouteUsedEvent`, `RouteEvent`, `RouteEndEvent`, `RouteCallbackEndedEvent`, and `ReRouteEvent`

See Also:

RouteSession, RouteCallback, RouteUsedEvent, RouteEvent, ReRouteEvent, RouteEndEvent, RouteCallbackEndedEvent

Method Index

- **getRouteSession**()

  Returns the `RouteSession` associated with this event.

Methods

getRouteSession

```
public abstract RouteSession getRouteSession()
```

Returns the `RouteSession` associated with this event.

Returns:

The RouteSession associated with this event.

All Packages This Package Previous Next

Interface javax.telephony.callcenter.events.RouteUsedEvent

public interface **RouteUsedEvent**

extends RouteSessionEvent

The `RouteUsedEvent` event interface indicates the `RouteSession` interface has moved into the `RouteSession.ROUTE_USED` state and the Call has terminated at a destination as a result of routing by the application. This interface extends the `RouteSessionEvent` interface and is reported via the `RouteCallback` interface.

See Also:

RouteSession, RouteCallback, RouteSessionEvent

Method Index

- **getCallingAddress**()

  Returns the calling Address.
- **getCallingTerminal**()

  Returns the calling Terminal.
- **getDomain**()

  Returns true if the call was routed out of the Provider's domain, false otherwise.
- **getRouteUsed**()

  FOR JTAPI 1.2 return an `Address` as the final destination (was the final destination Terminal.)

Methods

getRouteUsed

```
public abstract Address getRouteUsed()
```

FOR JTAPI 1.2 return an `Address` as the final destination (was the final destination Terminal.)

Returns:

The final destination Address.

getCallingTerminal

```
public abstract Terminal getCallingTerminal()
```

Returns the calling Terminal.

Returns:

The calling Terminal.

getCallingAddress

```
public abstract Address getCallingAddress()
```

Returns the calling Address.

Returns:

The calling Address.

getDomain

```
public abstract boolean getDomain()
```

Returns true if the call was routed out of the Provider's domain, false otherwise.

Returns:

True if the call was routed out of the Provider's domain, false otherwise.

All Packages   This Package Previous Next

All Packages

package javax.telephony.callcontrol

Interface Index

- CallControlAddress
- CallControlAddressObserver
- CallControlCall
- CallControlCallObserver
- CallControlConnection
- CallControlTerminal
- CallControlTerminalConnection
- CallControlTerminalObserver

Class Index

- CallControlForwarding

Interface javax.telephony.callcontrol.CallControlAddress

public interface **CallControlAddress**

extends Address

Introduction

The `CallControlAddress` interface extends the core `Address` interface. It provides additional features on the Address. Applications may query an Address object using the `instanceof` operator to see whether it supports this interface.

Address Forwarding

This interface supports methods which permit applications to modify and query the forwarding characteristics of an Address. The forwarding characteristics determine how incoming telephone calls to this Address should be handled, if any special handling is desired. A switching domain will honor those instructions to the extent that other possibly higher priority instructions allow.

Each Address may have zero or more *forwarding instructions*. Each instruction describes how the switching domain should handle incoming telephone calls to an Address under different circumstances. Examples of forwarding instructions are "forward all calls to x9999" or "forward all calls to x7777 when no one answers." Each forwarding instruction is represented by an instance of the `CallControlForwarding` class. A switching domain will honor forwarding instructions to the extent that other (possibly higher priority) instructions allow.

Applications assign a list of forwarding instructions via the the `CallControlAddress.setForwarding()` method. To obtain the current, effective forwarding instructions, applications invoke the `CallControlAddress.getForwarding()` method. To cancel all forwarding instructions, applications use the `CallControlAddress.cancelForwarding()` method.

Do Not Disturb and Message Waiting

The `CallControlAddress` interface defines two additional attributes: *do-not-disturb* and *message-waiting*.

The do-not-disturb feature gives the means to notify the telephony platform that an Address does not want to receive incoming telephone calls. That is, if this feature is activated, the underlying telephony platform will not alert this Address to incoming telephone calls. Applications use the `CallControlAddress.setDoNotDisturb()` method to activate or deactivate this feature and the `CallControlAddress.getDoNotDisturb()` method to return the current state of this attribute.

Note that the `CallControlTerminal` interface also has a do-not-disturb attribute. This gives the ability to control the do not disturb property at either the Address level (e.g. a phone number) or at the Terminal level (e.g. an individual phone.)

The message-waiting attribute indicates whether there are messages waiting for a human user of the Address. These messages may either be maintained by an application or some telephony platform. Applications inform the telephony hardware of the message waiting status, and typically the hardware displays a visible indicator (e.g. an LED) to users. Applications use the `CallControlAddress.setMessageWaiting()` method to activate or deactivate this feature and the `CallControlAddress.getMessageWaiting()` method to return the current state of this attribute.

Observers and Events

All events pertaining to the `CallControlAddress` interface are reported via the `AddressObserver.addressChangedEvent()` method. The application observer object must also implement the `CallControlCallObserver` interface to express interest in the call control package events.

The following are the events associated with this interface:

`CallCtlAddrDoNotDisturbEv`	Indicates the do-not-disturb feature of this Address has changed.
`CallCtlAddrForwardEv`	Indicates the forwarding characteristics of this Address has changed.
`CallCtlAddrMessageWaitingEv`	Indicates the message waiting characteristics of this Address has changed.

See Also:

CallControlTerminal, CallControlForwarding, CallControlAddressObserver, CallCtlAddrDoNotDisturbEv, CallCtlAddrForwardEv, CallCtlAddrMessageWaitingEv

Method Index

- **cancelForwarding**()

  Cancels all of the forwarding instructions on this Address.
- **getDoNotDisturb**()

  Returns true if the do-not-disturb feature is activated, false otherwise.
- **getForwarding**()

  Returns an array of forwarding instructions currently effective for this Address.
- **getMessageWaiting**()

  Returns true if the message waiting is activated, false otherwise.
- **setDoNotDisturb**(boolean)

  Specifies whether the do-not-disturb feature should be activated or deactivated for this Address.
- **setForwarding**(CallControlForwarding[])

  Sets the forwarding characteristics for this Address.
- **setMessageWaiting**(boolean)

  Specifies whether the message-waiting indicator should be activated or deactivated for this Address.

Methods

setForwarding

```
 public abstract void setForwarding(CallControlForwarding instructions[]) throws
MethodNotSupportedException, InvalidStateException, InvalidArgumentException
```

Sets the forwarding characteristics for this Address. This forwarding command supplants all previous forwarding instructions if it succedes, otherwise the forwarding state at the time of the command remains unchanged. This method takes an array of `CallControlForwarding` objects. Each object describes a different forwarding. This method waits until all forwarding instructions have been set or until an error occurs and an exception is thrown.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getForwarding() == instructions
3. CallCtlAddrForwardEv is delivered for this Address

Parameters:

instructions - An array of Address forwarding instructions

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

Throws: InvalidStateException

The Provider is not "in service".

Throws: InvalidArgumentException

An invalid set of forwarding instructions were given as a parameter.

See Also:

CallCtlAddrForwardEv

getForwarding

```
 public abstract CallControlForwarding[] getForwarding() throws
MethodNotSupportedException
```

Returns an array of forwarding instructions currently effective for this Address. If there are no effective forwarding instructions, this method returns null.

Returns:

An array of Address forwarding instructions, null if there are none.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

cancelForwarding

```
 public abstract void cancelForwarding() throws MethodNotSupportedException,
InvalidStateException
```

Cancels all of the forwarding instructions on this Address. When this method completes, the `CallControlAddress.getForwarding()` method will return null. This method waits until all forwarding instructions have been cancelled or until an error occurs and an exception is thrown.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getForwarding == null
3. CallCtlAddrForwardEv is delivered for this Address.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

Throws: InvalidStateException

The Provider is not "in service".

See Also:

CallCtlAddrForwardEv

getDoNotDisturb

```
 public abstract boolean getDoNotDisturb() throws MethodNotSupportedException
```

Returns true if the do-not-disturb feature is activated, false otherwise.

Returns:

True if do-not-disturb feature is activated, false if it is deactivated.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

setDoNotDisturb

```
 public abstract void setDoNotDisturb(boolean enable) throws
MethodNotSupportedException, InvalidStateException
```

Specifies whether the do-not-disturb feature should be activated or deactivated for this Address. This feature only affects whether or not calls will be accepted at this Address. Note that the do-not-disturb feature on all Terminals associated with this Address are independent of this Terminal's do-not-disturb setting. If 'enable' is true, do-not-disturb is activated if not already activated. If 'enable' is false, do-not-disturb is deactivated if not already deactivated.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getDoNotDisturb() == enable
3. CallCtlAddrDoNotDisturbEv is delivered for this Address

Parameters:

enable - True to activate do-not-disturb, false to deactivate.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

Throws: InvalidStateException

The Provider is not "in service".

See Also:

CallCtlAddrDoNotDisturbEv

getMessageWaiting

```
public abstract boolean getMessageWaiting() throws MethodNotSupportedException
```

Returns true if the message waiting is activated, false otherwise.

Returns:

True if message-waiting is activated, false if it is deactivated.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

setMessageWaiting

```
public abstract void setMessageWaiting(boolean enable) throws
MethodNotSupportedException, InvalidStateException
```

Specifies whether the message-waiting indicator should be activated or deactivated for this Address. If 'enable' is true, message-waiting is activated if not already activated. If 'enable' is false, message-waiting is deactivated if not already deactivated.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getMessageWaiting() == enable
3. CallCtlAddrMessageWaitingEv is delivered for this Address

Parameters:

enable - True to activate message-waiting, false to deactivate.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

Throws: InvalidStateException

The Provider is not "in service".

See Also:

CallCtlAddrMessageWaitingEv

All Packages   This Package   Previous   Next

Interface javax.telephony.callcontrol.CallControlAddressObserver

public interface **CallControlAddressObserver**

extends AddressObserver

The `CallControlAddressObserver` interface reports all events for the `CallControlAddress` interface. Applications implement this interface to receive `CallControlAddress`-related events. All events are reported via the `AddressObserver.addressChangedEvent()` method. This interface, therefore, allows applications to signal to the implementation that they are interested in call control package events. This interface defines no additional methods.

All events must extend the `CallCtlAddrEv` event interface, which in turn, extends the core `AddrEv` interface.

The following are those events which are associated with this interface:

`CallCtlAddrDoNotDisturbEv`	Indicates the do not disturb characteristics of this Address has changed.
`CallCtlAddrForwardEv`	Indicates the forwarding characteristics of this Address has changed.
`CallCtlAddrMessageWaitingEv`	Indicates the message waiting characteristics of this Address has changed.

See Also:

AddressObserver, AddrEv, CallControlAddress, CallCtlAddrEv, CallCtlAddrDoNotDisturbEv, CallCtlAddrForwardEv, CallCtlAddrMessageWaitingEv

All Packages   This Package   Previous   Next

Interface javax.telephony.callcontrol.CallControlCall

public interface **CallControlCall**

extends Call

Introduction

The `CallControlCall` interface extends the core `Call` interface. This interface provides additional methods on a Call.

Additional Call Information

This interface supports methods which return additional information regarding the Call. Specifically, it returns the *calling address*, *calling terminal*, *called address*, and *last redirected address* information.

The calling address, as returned by the `CallControlCall.getCallingAddress()` method is the Address which originally placed the Call. The calling terminal, as returned by the `CallControlCall.getCallingTerminal()` method is the Terminal which originally placed the Call. The called Address, as returned by the `CallControlCall.getCalledAddress()` method is the Address to which the Call was originally placed. The last redirected address, as returned by the `CallControlCall.getLastRedirectedAddress()` method is the Address to which this Call was placed before the current destination Address. For example, if a Call was forwarded from one Address to another, then the first Address is the last redirected Address for this call.

Each of these methods returns `null` if their values are unknown at the present time. During the lifetime of a Call, an implementation may learn this additional information, and return different values for some or all of these methods as a result.

Conferencing Telephone Calls

The conferencing feature supported by this interface permits two telephone calls to be "merged". That is, the two Calls are merged into a single Call with the union of all of the participants of the two Calls being placed on the single Call.

Applications invoke the `CallControlCall.conference()` method to perform the conferencing feature. This method is given the "second" Call as an argument. All participants are moved from the second Call to the Call on which the method is invoked. The second Call moves into the `Call.INVALID` state as a result.

In order for the conferencing feature to happen, there must be a common participant to both Calls, as represented by a single Terminal and two TerminalConnections, one on each of the two Calls. These two TerminalConnections are known as the *conference controllers*. In the real-world, one of the two telephone calls must be on hold with respect to the controlling Terminal, and hence, the TerminalConnection on the second Call must be in the `CallControlTerminalConnection.HELD` state. The two conference controlling TerminalConnections are merged into one as a result of this method.

Applications may control which TerminalConnection acts as the conference controller via the `CallControlCall.setConferenceController()` method. The `CallControlCall.getConferenceController()` method returns the current conference controller, `null` if there is none. If no conference controller is set, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked.

Transferring Telephone Calls

The transfer feature supported by this interface permits one Call to be "moved" to another Call. That is, all of the participants from one Call are moved to another Call, except for the transferring participant which drops off from both Calls.

Applications invoke the `CallControlCall.transfer()` method to perform the transfer feature. There are two overloaded versions of this method. The first method takes a second Call as an argument. This method acts similarly to `CallControlCall.conference()`, except the two TerminalConnections on each Call with a common Terminal are removed from both Calls. The second version takes a string telephone address as an argument. This method removes the transfer controller participant while placing the telephone call to the designated address. This latter version of the transfer feature is often known as a *single-step transfer*.

In order for the transfer feature to happen, there must be a participant which acts as the *transfer controller*. The transfer controller is a

TerminalConnection around which the transfer is performed. In the first version of the `CallControlCall.transfer()` method, the transfer controller must be present on each of the two Calls and share a common Terminal. In the second version, the transfer controller only applies to the Call object on which the method is invoked (since there is no second Call involved). In both cases, the transfer controller participant is no longer part of any Call once the transfer feature is complete.

Applications may control which TerminalConnection acts as the transfer controller via the `CallControlCall.setTransferController()` method. The `CallControlCall.getTransferController()` method returns the current transfer controller, `null` if there is none. If no transfer controller is set, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked.

Consultation Calls

Consultation Calls are special types of telephony calls created (often temporarily) for a specific purpose. Consultation calls are created if a user wants to "consult" with another party briefly while currently on a Call, or are created for the purpose of conferencing or transferring with a Call. Consequently, consultation calls are always associated with another existing Call.

Applications invoke the `CallControlCall.consult()` method to perform the consultation feature. The instance on which the method is invoke is always the "idle" Call on which the consultation takes place. There are two overloaded versions of this method. The first method takes a TerminalConnection and a string telephone address as arguments. This consultation telephone call is associated with the Call of the TerminalConnection argument. This method places a telephone call from the same originating endpoint specified by the TerminalConnection argument to the designated telephone address string. The second version of this method only takes a TerminalConnection as an argument, and permits applications to use the `CallControlConnection.addToAddress()` method to dial the destination address string.

Additional CallControlCall Methods

The `CallControlCall.addParty()` method adds a single party to a Call given some telephone address string. The `CallControlCall.drop()` disconnects all parties from the Call and moves it into the `Call.INVALID` state. The `CallControlCall.offHook()` method takes an originating Address and Terminal pair "off hook" and permits applications to dial destination address digits one-by-one.

Observers and Events

All events pertaining to the `CallControlCall` interface are reported via the `CallObserver.callChangedEvent()` method. The application observer object must also implement the `CallControlCallObserver` interface to express interest in the call control package events. Applications receive events pertaining to the `CallControlConnection` and `CallControlTerminalConnection` interfaces via this observer as well.

All `CallControlCall`-related events must extend the `CallCtlCallEv` interface. There are no specific events pertaining to the `CallControlCall` interface, however.

See Also:

Call, CallObserver, CallControlCallObserver, CallCtlCallEv

Method Index

- **addParty**(String)

  Adds an additional party to an existing Call.
- **conference**(Call)

  Merges two Calls together, resulting in the union of the participants of both Calls being placed on a single Call.
- **consult**(TerminalConnection)

  Creates a consultation between this Call and an active Call.
- **consult**(TerminalConnection, String)

  Creates a consultation between this Call and an active Call.
- **drop**()

Drops the entire Call.

getCalledAddress()

Returns the called Address associated with this Call.

getCallingAddress()

Returns the calling Address associated with this call.

getCallingTerminal()

Returns the calling Terminal associated with this Call.

getConferenceController()

Returns the TerminalConnection which currently acts as the conference controller.

getConferenceEnable()

Return true if conferencing is enabled, false otherwise.

getLastRedirectedAddress()

Returns the last redirected Address associated with this Call.

getTransferController()

Returns the TerminalConnection which currently acts as the transfer controller.

getTransferEnable()

Return true if transferring is enabled, false otherwise.

offHook(Address, Terminal)

Takes the originating end of a Call off-hook.

setConferenceController(TerminalConnection)

Sets the TerminalConnection which acts as the conference controller for the Call.

setConferenceEnable(boolean)

Controls whether the Call is permitted or able to perform the conferencing feature.

setTransferController(TerminalConnection)

Sets the TerminalConnection which acts as the transfer controller for the Call.

setTransferEnable(boolean)

Controls whether the Call is permitted or able to perform the transferring feature.

transfer(Call)

This method moves all participants from one Call to another, with the exception of a selected common participant.

transfer(String)

This overloaded version of this method transfers all participants currently on this Call, with the exception of the transfer controller participant, to another telephone address.

Methods

getCallingAddress

```
 public abstract Address getCallingAddress()
```

Returns the calling Address associated with this call. The calling Address is the Address which originally placed the telephone call. If the calling address is unknown or not yet known, this method returns null.

Returns:

The calling Address.

getCallingTerminal

```
public abstract Terminal getCallingTerminal()
```

Returns the calling Terminal associated with this Call. The calling Terminal is the Terminal which originally placed the telephone call. If the calling Terminal is unknown or not yet known, this method returns null.

Returns:

The calling Terminal.

getCalledAddress

```
public abstract Address getCalledAddress()
```

Returns the called Address associated with this Call. The called Address is the Address to which the call had been originally placed. If the called address is unknown or not yet known, this method returns null.

Returns:

The called Address.

getLastRedirectedAddress

```
public abstract Address getLastRedirectedAddress()
```

Returns the last redirected Address associated with this Call. The last redirected Address is the Address at which the current Call was placed immediately before the current Address. This is common if a Call is forwarded to several Addresses before being answered. If the last redirected address is unknown or not yet known, this method returns null.

Returns:

The last redirected Address for this telephone Call.

addParty

```
public abstract Connection addParty(String newParty) throws InvalidStateException,
InvalidPartyException, MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Adds an additional party to an existing Call. This is sometimes called a "single-step conference" because a party is conferenced into a Call directly. The telephone address string provided as the argument must be complete and valid.

States of the Existing Connections

The Call must have at least two Connections in the `CallControlConnection.ESTABLISHED` state. An additional restriction requires that at most one other Connection may be in either the `CallControlConnection.QUEUED`, `CallControlConnection.OFFERED`, or `CallControlConnection.ALERTING` state.

Some telephony platforms impose restrictions on the number of Connections in a particular state. For instance, it is common to restrict the number of "alerting" Connections to at most one. As a result, this method requires that at most one other Connections is in the "queued", "offering", or "alerting" state. (Note that the first two states correspond to the core Connection "in progress" state). Although some systems may not enforce this requirement, for consistency, JTAPI specifies implementations must uphold the conservative requirement.

The New Connection

This method creates and returns a new Connection representing the new party. This Connection must at least be in the `CallControlConnection.IDLE` state. Its state may have progressed beyond "idle" before this method returns, and should be reflected by an event. This new Connection will progress as any normal destination Connection on a Call. Typical scenarios for this Connection are described by the `Call.connect()` method.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. Let c[] = call.getConnections() where c.length >= 2
4. c[i].getCallControlState() == CallControlConnection.ESTABLISHED for at least two i
5. c[j].getCallControlState() == CallControlConnection.QUEUED, CallControlConnection.OFFERED, or

CallControlConnection.ALERTING for at most one c[j]

Post-conditions:

1. Let connection be the Connection created and returned
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. connection.getCallControlState() at least CallControlConnection.IDLE
5. ConnCreatedEv is delivered for connection

Parameters:

newParty - The telephone address of the party to be added.

Returns:

The new Connection associated with the added party.

Throws: InvalidStateException

Either the Provider is not "in service", the Call is not "active" or the proper conditions on the Connections does not exist. as designated by the pre-conditions for this method.

Throws: InvalidPartyException

The destination address string is not valid and/or complete.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv

drop

```
 public abstract void drop() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Drops the entire Call. This method is equivalent to using the `Connection.disconnect()` method on each Connection which is part of the Call. Typically, each Connection on the Call will move into the `CallControlConnection.DISCONNECTED` state, each TerminalConnection will move into the `CallControlTerminalConnection.DROPPED` state, and the Call will move into the `Call.INVALID` state.

There are some Connections for which the application does not possess the proper authority to disconnect. In this case, this method performs no action on these Connections. These Connections may disconnect naturally as a result of disconnecting other Connections, however. This method returns when it can successfully disconnect as many methods as it can. The application is notified via events whether the entire Call was successfully dropped.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. Let c[] = this.getConnections() before this method is invoked
3. If c[i].getCallControlState == CallControlConnection.DISCONNECTED, for all i, then this.getState() == Call.INVALID
4. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for all disconnected Connections
5. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all dropped TerminalConnections
6. CallInvalidEv is delivered if all Connections were disconnected

Throws: InvalidStateException

Either the Provider was not "in service" or the Call was not "active".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method and it can drop none of the Connections.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

offHook

```
 public abstract Connection offHook(Address origaddress,
                                    Terminal origterminal) throws
InvalidStateException, MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Takes the originating end of a Call off-hook. This method permits applications to simply take the originating terminal of a Call off-hook so that users may manually dial telephone number digits or applications may supply digits with the `CallControlConnection.addToAddress()` method. This is in contrast to the `Call.connect()` method which requires the complete destination address string.

This method takes the originating Address and Terminal as arguments. This Call must be in the `Call.IDLE` state. This method creates and returns a Connection to the originating Address in the `CallControlConnection.INITIATED` state. This method also creates a TerminalConnection in the `CallControlTerminalConnection.TALKING` state and associated with the new Connection and originating Terminal.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. Let connection be the Connection created and returned
4. Let terminalconnection be the TerminalConnection created
5. connection.getCallControlState() == CallControlConnection.INITIATED
6. Let tc[] = c.getTerminalConnections() where tc.length == 1
7. tc[0] == terminalconnection
8. tc[0].getCallControlState() == CallControlTerminalConnection.TALKING
9. tc[0] element of origterminal.getTerminalConnections()
10. connection element of origaddress.getConnections()
11. connection element of this.getConnections()
12. ConnCreatedEv is delivered for connection
13. TermConnCreatedEv is delivered for terminalconnection
14. CallActiveEv is delivered for this Call
15. CallCtlConnInitiatedEv/ConnConnectedEv is delivered for connection
16. CallCtlTermConnTalkingEv/TermConnActiveEv is delivered for terminalconnection

Parameters:

origaddress - The originating Address object.

origterminal - The originating Terminal object.

Returns:

The Connection associated with the originating end of the telephone call.

Throws: InvalidStateException

Either the Provider was not "in service" or the Call was not "idle".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, TermConnCreatedEv, CallActiveEv, CallCtlConnInitiatedEv, CallCtlTermConnTalkingEv

conference

```
 public abstract void conference(Call otherCall) throws InvalidStateException,
InvalidArgumentException, MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Merges two Calls together, resulting in the union of the participants of both Calls being placed on a single Call. This method takes a Call as an argument, referred to hereafter as the "second" Call. All of the participants from the second call are moved to the Call on which this method is invoked.

The Conference Controller

In order for the conferencing feature to happen, there must be a common participant to both Calls, as represented by a single Terminal and two TerminalConnections, one on each of the two Calls. These two TerminalConnections are known as the *conference controllers*. In the real-world, one of the two Calls must be on hold with respect to the controlling Terminal, and hence, the TerminalConnection on the second Call must be in the `CallControlTerminalConnection.HELD` state. The two conference controlling TerminalConnections are merged into one as a result of this method.

Applications may control which TerminalConnection acts as the conference controller via the `CallControlCall.setConferenceController()` method. The `CallControlCall.getConferenceController()` method returns the current conference controller, `null` if there is none. If no conference controller is set, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked.

The Telephone Call Argument

All of the participants from the second Call, passed as the argument to this method, are "moved" to the Call on which this method was invoked. That is, new Connections and TerminalConnections are created on this Call which are found on the second Call. Those Connections and TerminalConnections on the second Call are removed from the Call and the Call moves into the `Call.INVALID` state.

The conference controller TerminalConnections are merged into one on this Call. That is, the existing TerminalConnection controller on this Call is left unchanged, while the TerminalConnection on the second Call is removed from that Call.

Other Shared Participants

There may exist Address and Terminals which are part of both telephone call in addition to the designated conference controller. In these instances, those participants which are shared between both Calls are merged into one. That is, the Connections and TerminalConnections on this Call are left unchanged. The corresponding Connections and TerminalConnections on the second Call are removed from that Call.

Pre-conditions:

1. Let tc1 be the conference controller on this Call
2. Let connection1 = tc1.getConnection()
3. Let tc2 be the conference controller on otherCall
4. (this.getProvider()).getState() == Provider.IN_SERVICE

5. this.getState() == Call.ACTIVE
6. tc1.getTerminal() == tc2.getTerminal()
7. tc1.getCallControlState() == CallControlTerminalConnection.TALKING
8. tc2.getCallControlState() == CallControlTerminalConnection.HELD
9. this != otherCall

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. otherCall.getState() == INVALID
4. Let c[] be the Connections to be merged from otherCall
5. Let tc[] be the TerminalConnections to be merged from otherCall
6. Let new(c) be the set of new Connections created on this Call
7. Let new(tc) be the set of new TerminalConnections created on this Call
8. new(c) element of this.getConnections()
9. new(c).getCallState() == c.getCallState()
10. new(tc) element of (this.getConnections()).getTerminalConnections()
11. new(tc).getCallState() == tc.getCallState()
12. c[i].getCallControlState() == CallControlConnection.DISCONNECTED for all i
13. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED for all i
14. CallInvalidEv is delivered for otherCall
15. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for all c[i]
16. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]
17. ConnCreatedEv is delivered for all new(c)
18. TermConnCreatedEv is delivered for all new(tc)
19. Appropriate events are delivered for all new(c) and new(tc)

Parameters:

otherCall - The second Call which to merge with this Call object.

Throws: InvalidArgumentException

The Call object provided is not valid for the conference

Throws: InvalidStateException

Either the Provider is not "in service", the Call is not "active", or the conference controllers are not in the proper state.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, TermConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

transfer

```
 public abstract void transfer(Call otherCall) throws InvalidStateException,
InvalidArgumentException, InvalidPartyException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method moves all participants from one Call to another, with the exception of a selected common participant. This method takes a Call as an argument, referred to hereafter as the "second" Call. All of the participants, with the exception for the selected common participant, from the second call are moved to the Call on which this method is invoked.

The Transfer Controller

In order for the transfer feature to happen, there must be a participant which acts as the transfer controller. The transfer controller is a TerminalConnection around which the transfer is placed. This transfer controller must be present on each of the two Calls and share a common Terminal. The transfer controller participant is no longer part of any Call once this transfer feature is complete. The transfer controllers on each of the two Calls must be in either of the `CallControlTerminalConnection.TALKING` or `CallControlTerminalConnection.HELD` state.

Applications may control which TerminalConnection acts as the transfer controller via the `CallControlCall.setTransferController()` method. The `CallControlCall.getTransferController()` method returns the current transfer controller, `null` if there is none. If no transfer controller is set, the implementation chooses a suitable TerminalConnection when the transfer feature is invoked.

The Telephone Call Argument

All of the participants from the second Call, passed as the argument to this method, are "moved" to the Call on which this method is invoked, with the exception of the transfer controller. That is, new Connections and TerminalConnections are created on this Call which are found on the second Call. Those Connections and TerminalConnections on the second Call are removed from the Call and the Call moves into the `Call.INVALID` state.

The transfer controller TerminalConnections are dropped from both Calls. They move into the `CallControlTerminalConnection.DROPPED` state.

Other Shared Participants

There may exist Address and Terminals which are part of both Calls in addition to the designated transfer controller. In these instances, those participants which are shared between both Calls are merged into one. That is, the Connections and TerminalConnections on this Call are left unchanged. The corresponding Connections and TerminalConnections on the second Call are removed from that Call.

Pre-conditions:

1. Let tc1 be the transfer controller on this Call
2. Let tc2 be the transfer controller on otherCall
3. this != otherCall
4. (this.getProvider()).getState() == Provider.IN_SERVICE
5. this.getState() == Call.ACTIVE
6. otherCall.getState() == Call.ACTIVE
7. tc1.getCallControlState() == CallControlTerminalConnection.TALKING or CallControlTerminalConnection.HELD
8. tc2.getCallControlState() == CallControlTerminalConnection.TALKING or CallControlTerminalConnection.HELD

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. otherCall.getState() == Call.INVALID
4. tc1.getCallControlState() == CallControlTerminalConnection.DROPPED
5. tc2.getCallControlState() == CallControlTerminalConnection.DROPPED
6. Let c[] be the Connections to be transferred from otherCall
7. Let tc[] be the TerminalConnections to be transferred from otherCall
8. Let new(c) be the set of new Connections created on this Call
9. Let new(tc) be the set of new TerminalConnections created on this Call
10. new(c) element of this.getConnections()
11. new(c).getCallState() == c.getCallState()
12. new(tc) element of (this.getConnections()).getTerminalConnections()

13. new(tc).getCallState() == tc.getCallState()
14. c[i].getCallControlState() == CallControlConnection.DISCONNECTED for all i
15. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED for all i
16. CallInvalidEv is delivered for otherCall
17. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for tc1, tc2
18. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for all c[i]
19. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]
20. ConnCreatedEv is delivered for all new(c)
21. TermConnCreatedEv is delivered for all new(tc)
22. Appropriate events are delivered for all new(c) and new(tc)

Parameters:

otherCall - The other Call which to transfer to this Call.

Throws: InvalidArgumentException

The TerminalConnection controlling the transfer is not valid or does not exist or the other Call is not valid.

Throws: InvalidStateException

Either the Provider is not "in service", the Call are not "active", or the transfer controllers are not "talking" or "held".

Throws: InvalidPartyException

The other Call given as the argument is not a valid Call to conference with.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, TermConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

transfer

```
 public abstract Connection transfer(String address) throws
InvalidArgumentException, InvalidStateException, InvalidPartyException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

This overloaded version of this method transfers all participants currently on this Call, with the exception of the transfer controller participant, to another telephone address. This is often called a "single-step transfer" because the transfer feature places another telephone call and performs the transfer at one time. The telephone address string given as the argument to this method must be valid and complete.

The Transfer Controller

The transfer controller for this version of this method represents the participant on this Call around which the transfer is taking place and who drops off the Call once the transfer feature has completed. The transfer controller is a TerminalConnection which must be in the `CallControlTerminalConnection.TALKING` state.

Applications may control which TerminalConnection acts as the transfer controller via the `CallControlCall.setTransferController()` method. The `CallControlCall.getTransferController()` method returns the current transfer controller, `null` if there is none. If no transfer controller is set, the implementation chooses a suitable TerminalConnection when the transfer feature is invoked.

When the transfer feature is invoked, the transfer controller moves into the `CallControlTerminalConnection.DROPPED` state. If it is the only TerminalConnection associated with its Connection, then its Connection moves into the

`CallControlConnection.DISCONNECTED` state as well.

The New Connection

This method creates and returns a new Connection representing the party to which the Call was transferred. Note that this Connections may be `null` in the case the Call has been transferred outside of the Provider's domain and can no longer be tracked. This Connection must at least be in the `CallControlConnection.IDLE` state. Its state may have progressed beyond "idle" before this method returns, and should be reflected by an event. This new Connection will progress as any normal destination Connection on a telephone call. Typical scenarios for this Connection are described by the `Call.connect()` method.

Pre-conditions:

1. Let tc be the transfer controller on this Call
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. tc.getCallControlState() == CallControlTerminalConnection.TALKING

Post-conditions:

1. Let newconnection be the Connection created and returned
2. Let connection == tc.getConnection()
3. (this.getProvider()).getState() == Provider.IN_SERVICE
4. this.getState() == Call.ACTIVE
5. tc.getCallControlState() == CallControlTerminalConnection.DROPPED
6. If connection.getTerminalConnections().length == 1, then connection.getCallControlState() == CallControlConnection.DISCONNECTED
7. newconnection is an element of this.getConnections(), if not null.
8. newconnection.getCallControlState() at least CallControlConnection.IDLE, if not null.
9. ConnCreatedEv is delivered for newconnection
10. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for tc
11. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for connection if no other TerminalConnections

Parameters:

dialedDigits - The destination telephone address string to where the Call is being transferred.

Returns:

The new Connection associated with the destination or null.

Throws: InvalidArgumentException

The TerminalConnection provided as controlling the transfer is not valid or part of this Call.

Throws: InvalidStateException

Either the Provider is not "in service", the Call is not "active", or the transfer controller is not "talking".

Throws: InvalidPartyException

The destination address is not valid and/or complete.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

● **setConferenceController**

```
 public abstract void setConferenceController(TerminalConnection tc) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
ResourceUnavailableException
```

Sets the TerminalConnection which acts as the conference controller for the Call. The conference controller represents the participant in the Call around which a conference takes place.

Typically, when two Calls are conferenced together, a single participant is part of both Calls. This participant is represented by a TerminalConnection on each Call, each of which shares the same Terminal.

If the designated TerminalConnection is not part of this Call, an exception is thrown. If the TerminalConnection leaves the Call in the future, the implementation resets the conference controller to `null`.

Pre-conditions:

1. Let connections[] = this.getConnections()
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. tc element of connections[i].getTerminalConnections() for all i
5. tc.getCallControlState() != CallControlTerminalConnection.DROPPED

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. Let connections[] = this.getConnections()
4. tc element of connections[i].getTerminalConnections() for all i
5. tc.getCallControlState() != CallControlTerminalConnection.DROPPED
6. tc == this.getConferenceController()

Parameters:

tc - The TerminalConnection to use as the conference controller

Throws: InvalidArgumentException

The TerminalConnection provided is not associated with this Call.

Throws: InvalidStateException

Either the Provider is not "in service" or the Call is not "active".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

getConferenceController

```
 public abstract TerminalConnection getConferenceController()
```

Returns the TerminalConnection which currently acts as the conference controller. The conference controller represents the participant in the telephone around which a conference takes place.

When a Call is initially created, the conference controller is set to `null`. This method returns non-null only if the application has previously set the conference controller. If the current conference controller leaves the Call, the conference controller is reset to `null`.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() != Call.INVALID

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() != Call.INVALID

3. Let tc = this.getConferenceController()
4. Let connections[] = this.getConnections()
5. tc element of connections[i].getTerminalConnections() for all i, if tc is not null
6. tc.getCallControlState() != CallControlTerminalConnection.DROPPED, if tc is not null

Returns:

The current TerminalConnection acting as the conference controller, null if one is not set.

setTransferController

```
 public abstract void setTransferController(TerminalConnection tc) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
ResourceUnavailableException
```

Sets the TerminalConnection which acts as the transfer controller for the Call. The transfer controller represents the participant in the Call around which a transfer takes place.

If the designated TerminalConnection is not part of this Call, an exception is thrown. If the TerminalConnection leaves the Call in the future, the implementation resets the transfer controller to `null`.

Pre-conditions:

1. Let connections[] = this.getConnections()
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. tc element of connections[i].getTerminalConnections() for all i
5. tc.getCallControlState() != CallControlTerminalConnection.DROPPED

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. Let connections[] = this.getConnections()
4. tc element of connections[i].getTerminalConnections() for all i
5. tc.getCallControlState() != CallControlTerminalConnection.DROPPED
6. tc == this.getTransferController()

Parameters:

tc - The TerminalConnection to use as the transfer controller

Throws: InvalidArgumentException

The TerminalConnection provided is not associated with this Call.

Throws: InvalidStateException

Either the Provider is not "in service", Call was not "active", or the TerminalConnection argument was "dropped".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

getTransferController

```
 public abstract TerminalConnection getTransferController()
```

Returns the TerminalConnection which currently acts as the transfer controller. The transfer controller represents the participant in the Call around which a transfer takes place.

When a Call is initially created, the transfer controller is set to `null`. This method returns non-null only if the application has previously set the transfer controller. If the current transfer controller leaves the telephone call, the transfer controller is reset to `null`.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() != Call.INVALID

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() != Call.INVALID
3. Let tc = this.getTransferController()
4. Let connections[] = this.getConnections()
5. tc element of connections[i].getTerminalConnections() for all i, if tc is not null
6. tc.getCallControlState() != CallControlTerminalConnection.DROPPED, if tc is not null

Returns:

The current TerminalConnection acting as the transfer controller, null if one is not set.

setConferenceEnable

```
 public abstract void setConferenceEnable(boolean enable) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException
```

Controls whether the Call is permitted or able to perform the conferencing feature. The boolean argument provided indicates whether conferencing should be turned on (true) or off (false). This method throws an exception if the boolean argument is true and the implementation does not support the conferencing feature. This method must be invoked when the Call is in the `Call.IDLE` state.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. enable = this.getConferenceEnable()

Parameters:

enable - True turns conferencing on, false turns conferencing off.

Throws: InvalidArgumentException

Conferencing cannot be turned on as requested by a true argument.

Throws: InvalidStateException

Either the Provider is not "in service" or the Call is not "idle".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

getConferenceEnable

```
 public abstract boolean getConferenceEnable()
```

Return true if conferencing is enabled, false otherwise. Applications may use this method initially to obtain the default value set by the implementation and may attempt to change this value using the `CallControlCall.setConferenceEnable()` method.

Returns:

True if conferencing is enabled, false otherwise.

setTransferEnable

```
 public abstract void setTransferEnable(boolean enable) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException
```

Controls whether the Call is permitted or able to perform the transferring feature. The boolean argument provided indicates whether transferring should be turned on (true) or off (false). This method throws an exception if the boolean argument is true and the implementation does not support the transferring feature. This method must be invoked when the Call is in the `Call.IDLE` state.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. enable = this.getConferenceEnable()

Parameters:

enable - True turns transferring on, false turns transferring off.

Throws: InvalidArgumentException

Transferring cannot be turned on as requested by the true argument.

Throws: InvalidStateException

Either the Provider is not "in service" or the Call is not "idle".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

getTransferEnable

```
 public abstract boolean getTransferEnable()
```

Return true if transfering is enabled, false otherwise. Applications may use this method initially to obtain the default value set by the implementation and may attempt to change this value using the `CallControlCall.setTransferEnable()` method.

Returns:

True if transferring is enabled, false otherwise.

consult

```
 public abstract Connection[] consult(TerminalConnection tc,
                                      String dialedDigits) throws
InvalidStateException, InvalidArgumentException, MethodNotSupportedException,
ResourceUnavailableException, PrivilegeViolationException, InvalidPartyException
```

Creates a consultation between this Call and an active Call. A consultation Call is a new, idle Call which is associated with a particular existing Call and often created for a particular purpose. For example, the consultation Call may be used simply to "consult" with another party or to conference or transfer with its associated Call. This method establishes a special relationship between the two Calls which extends down to the telephony platform level. Most often, this feature is directly provided by the underlying telephony platform.

This Call Object

The instance on which this method is invoked is used as the Call on which the consultation takes place. This Call must be in `Call.IDLE` and is created with the `Provider.createCall()` method.

The TerminalConnection Argument

The TerminalConnection argument provides two pieces of information. The first piece of information is the other active Call to which this idle Call is associated. The Call associated with the TerminalConnection argument must be in the `Call.ACTIVE` state.

The second piece of information given by the TerminalConnection argument is the originating endpoint from which to place a telephone call on this idle Call. That is, the Address and Terminal associated with the TerminalConnection argument are used as the originating endpoint for the telephone call. The state of the TerminalConnection must be `CallControlTerminalConnection.TALKING` and this method first moves it into the `CallControlTerminalConnection.HELD` in order to place a telephone call on this idle Call.

The Destination Address String

A telephone call is placed to the destination telephone address string given as the second argument to this method. This address string must be valid and complete.

The Consultation Purpose

As mentioned above, the purpose of creating a consultation Call is often to perform a transfer or conference action on these two Calls. If the methods `CallControlCall.setConferenceEnable()` and `CallControlCall.setTransferEnable()` are supported as indicated by the `CallControlCallCapabilities` interface, applications must specify the purpose of the consultation Call by first telling the telephony platform if the intend to perform a transfer and/or conference action.

If the `CallControlCall.setConferenceEnable()` and the `CallControlCall.setTransferEnable()` methods are not supported as indicated by this interface's capabilities, applications permit the telephony platform to use the static, default values reported by the `CallControlCall.getConferenceEnable()` and the `CallControlCall.getTransferEnable()` methods.

The Telephone Call

A telephone call is placed on this Call and an array of two Connections are returned representing the originating and destination participants of the Call. The Call progresses in the same way as if the Call was placed using the `Call.connect()` method. The description of that method describes different scenarios under which the state of the call progresses.

Pre-conditions:

1. this.getProvider().getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. tc.getCallControlState() == CallControlTerminalConnection.TALKING
4. (tc.getCall()).getState() == Call.ACTIVE

Post-conditions:

1. Let connections[] be the two Connections created and returned
2. this.getProvider().getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. tc.getCallControlState() == CallControlTerminalConnection.HELD
5. (tc.getCall()).getState() == Call.ACTIVE
6. Let c[] = this.getConnections() such that c.length == 2
7. c[0].getCallControlState() at least CallControlConnection.IDLE
8. c[1].getCallControlState() at least CallControlConnection.IDLE
9. c == connection
10. CallActiveEv is delivered for this Call
11. ConnCreatedEv are delivered for both connections[i]
12. CallCtlTermConnHeldEv is delivered for tc

Parameters:

tc - The controlling TerminalConnection for the consultation call.

dialedDigits - The destination telephone address string to which a telephone call is being placed.

Returns:

The two new Connections in the Call.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is unavailable.

Throws: PrivilegeViolationException

The application does not have the proper authority to place a consultation telephone call.

Throws: InvalidArgumentException

The TerminalConnection given is not a valid originating endpoint for a Call.

Throws: InvalidPartyException

The destination address string is not valid and/or complete.

Throws: InvalidStateException

Either the Provider is not "in service", the Call is not "idle", the other Call is not "active", or the TerminalConnection is not "talking".

Throws: MethodNotSupportedException

The implementation does not support this method.

See Also:

CallActiveEv, ConnCreatedEv, CallCtlTermConnHeldEv

● consult

```
 public abstract Connection consult(TerminalConnection tc) throws
InvalidStateException, InvalidArgumentException, MethodNotSupportedException,
ResourceUnavailableException, PrivilegeViolationException
```

Creates a consultation between this Call and an active Call. A consultation call is a new, idle Call which is associated with a particular existing Call and often created for a particular purpose. For example, the consultation call may be used simply to "consult" with another party or to conference or transfer with its associated Call. This method establishes a special relationship between the two Calls which extends down to the telephony platform level. Most often, this feature is directly provided by the underlying telephony platform.

This overloaded version of this method has a single difference with the other version of the `CallControlCall.consult()` method. This method does not take a destination telephone address string as an argument.

This method creates and returns a single Connection which is in the `CallControlConnection.INITIATED` state. Applications may use the `CallControlConnection.addToAddress()` method to dial the destination address digits.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. tc.getCallControlState() == CallControlTerminalConnection.TALKING
4. (tc.getCall()).getState() == Call.ACTIVE

Post-conditions:

1. Let connection be the connection created and returned
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.IDLE
4. tc.getCallControlState() == CallControlTerminalConnection.HELD
5. (tc.getCall()).getState() == Call.ACTIVE
6. connection == c
7. Let c = this.getConnections() such that c.length == 1
8. c[0].getCallControlState() == CallControlConnection.INITIATED
9. ConnCreatedEv is delivered for connection
10. CallCtlConnInitiatedEv/ConnConnectedEv is delivered for connection
11. CallActiveEv is delivered for this Call

Parameters:

tc - The controlling TerminalConnection for the consultation call.

Returns:

The Connection in the "initiated" state.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is unavailable.

Throws: PrivilegeViolationException

The application does not have the proper authority to place a consultation telephone call.

Throws: InvalidArgumentException

The TerminalConnection given is not a valid originating endpoint for a Call.

Throws: InvalidStateException

Either the Provider is not "in service", the Call is not "idle", the other Call is not "active", or the TerminalConnection is not "talking".

Throws: MethodNotSupportedException

The implementation does not support this method.

See Also:

CallActiveEv, ConnCreatedEv, ConnConnectedEv, CallCtlConnInitiatedEv

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.CallControlCallObserver

public interface **CallControlCallObserver**

extends CallObserver

The `CallControlCallObserver` interface reports all events for the `CallControlCall` interface. It also reports events for the `CallControlConnection` and the `CallControlTerminalConnection` interfaces. Applications implement this interface to receive these events. All events are reported via the `CallObserver.callChangedEvent()` method. This interface, therefore, allows applications to signal to the implementation that they are interested in call control package events. This interface defines no additional methods.

All events must extend the `CallCtlCallEv` event interface, which in turn, extends the core `CallEv` interface.

The `CallControlConnection` state events defined in this package are: `CallCtlConnOfferedEv`, `CallCtlConnQueuedEv`, `CallCtlConnAlertingEv`, `CallCtlConnInitiatedEv`, `CallCtlConnDialingEv`, `CallCtlConnNetworkReachedEv`, `CallCtlConnNetworkAlertingEv`, `CallCtlConnFailedEv`, `CallCtlConnEstablishedEv`, `CallCtlConnUnknownEv`, and `CallCtlConnDisconnectedEv`.

The `CallControlTerminalConnection` state events defined in this package are: `CallCtlTermConnBridgedEv`, `CallCtlTermConnDroppedEv`, `CallCtlTermConnHeldEv`, `CallCtlTermConnInUseEv`, `CallCtlTermConnRingingEv`, `CallCtlTermConnTalkingEv`, and `CallCtlTermConnUnknownEv`.

See Also:

CallObserver, CallEv, Connection, TerminalConnection, CallCtlCallEv, CallCtlConnEv, CallCtlConnAlertingEv, CallCtlConnDialingEv, CallCtlConnDisconnectedEv, CallCtlConnEstablishedEv, CallCtlConnFailedEv, CallCtlConnInitiatedEv, CallCtlConnNetworkAlertingEv, CallCtlConnNetworkReachedEv, CallCtlConnOfferedEv, CallCtlConnQueuedEv, CallCtlConnUnknownEv, CallCtlTermConnEv, CallCtlTermConnRingingEv, CallCtlTermConnTalkingEv, CallCtlTermConnHeldEv, CallCtlTermConnBridgedEv, CallCtlTermConnInUseEv, CallCtlTermConnDroppedEv, CallCtlTermConnUnknownEv

Interface javax.telephony.callcontrol.CallControlConnection

public interface **CallControlConnection**

extends Connection

Introduction

The `CallControlConnection` interface extends the core `Connection` interface and provides additional features and greater detail about the Connection state. Applications may query a Connection object using the `instanceof` operator to see whether it supports this interface.

CallControlConnection State

This interface defines states for the Connection which provide greater detail beyond the states defined in the `Connection` interface. The states defined by this interface are related to the states defined in the core package in certain, specific ways, as defined below. Applications may obtain the `CallControlConnection` state via the `getCallControlState()` method defined on this interface. This method returns one of the integer state constants defined in this interface.

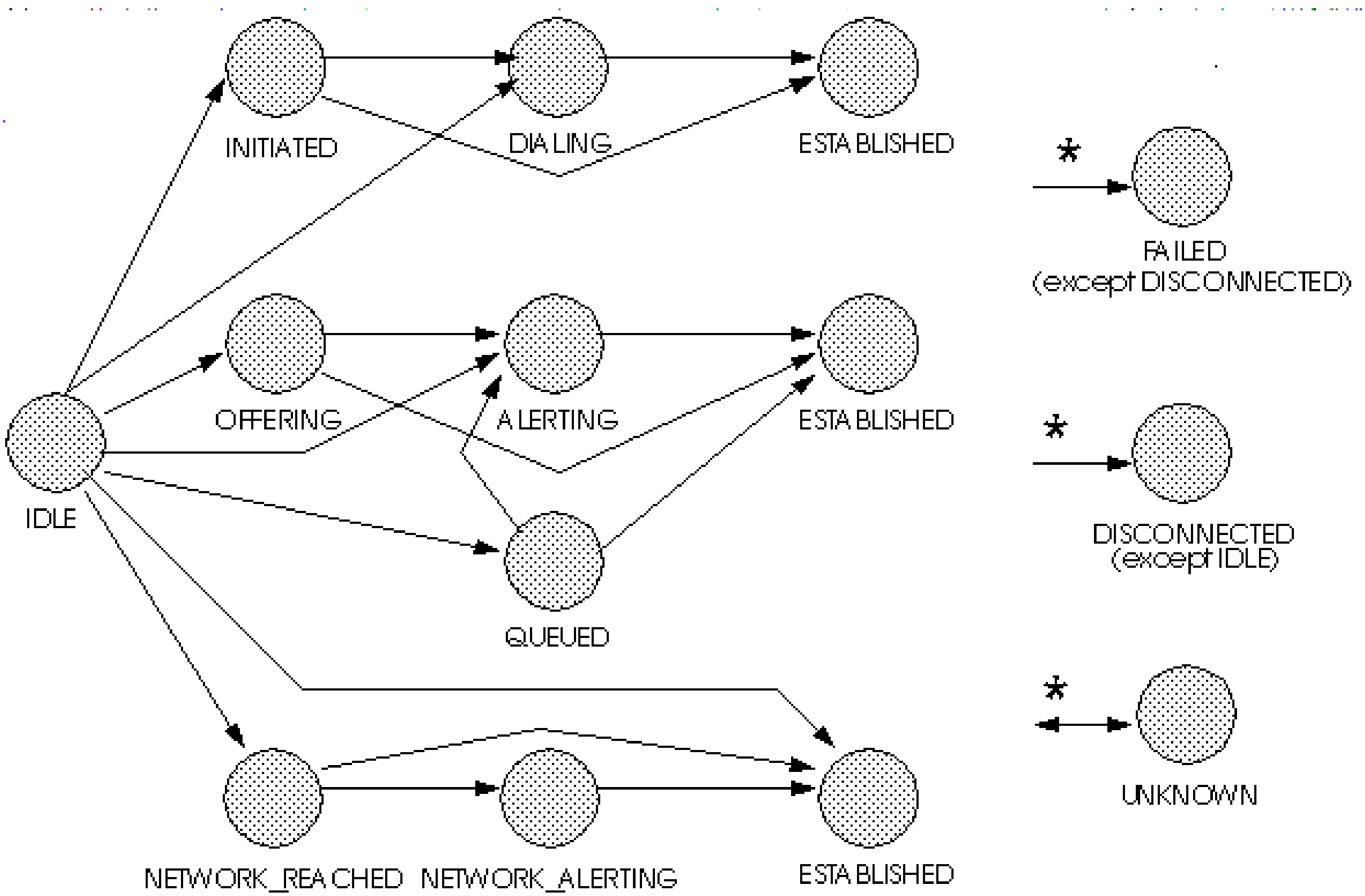
Below is a description of each `CallControlConnection` state in real-world terms. These real-world descriptions only serve to provide a more intuitive understanding of what is going on. Several methods in this specification state pre-conditions based upon the `CallControlConnection` state. Some of these states are identical to those defined in the core package.

`CallControlConnection.IDLE`	This state has the same definition as in the core package. It is the initial `CallControlConnection` state for all new Connections. Connections typically do not stay in this state for long, but quickly transition to another state.
`CallControlConnection.OFFERED`	This state indicates than an incoming call is being offered to the Address associated with the Connection. Typically, applications must either accept or reject this offered call before the called party is alerted to the incoming telephone call.
`CallControlConnection.QUEUED`	This state indicates that a Connection is queued at the particular Address associated with the Connection. For example, some telephony platforms permit the "queueing" of incoming telephone call to an Address when the Address is busy.
`CallControlConnection.NETWORK_REACHED`	This state indicates that an outgoing telephone call has reached the network. Applications may not receive further events about this leg of the telephone call, depending upon the ability of the telephone network to provide additional progress information. Applications must decide whether to treat this as a connected telephone call.
`CallControlConnection.NETWORK_ALERTING`	This state indicates that an outgoing telephone call is alerting at the destination end, which was previously only known to have reached the network. Typically, Connections transition into this state from the `CallControlConnection.NETWORK_REACHED` state. This state results from additional progress information being sent from the telephone network.
`CallControlConnection.ALERTING`	This state has the same definition as in the core package. It implies that the Address is being notified of an incoming call.
`CallControlConnection.INITIATED`	This state indicates the originating end of a telephone call has begun the process of placing a telephone call, but the dialing of the destination telephone address has not yet begun. Typically, a telephone associated with the Address has gone "off-hook".
`CallControlConnection.DIALING`	This state indicates the originating end of a telephone call has begun the process of dialing a destination telephone address, but has not yet completed dialing.
`CallControlConnection.ESTABLISHED`	This state is similar to that of `Connection.CONNECTED`. It indicates that the endpoint has reached its final, active state in the telephone call.
`CallControlConnection.DISCONNECTED`	This state has the same definition as in the core package. It implies the Connection object is no longer part of the telephone call.
`CallControlConnection.FAILED`	This state has the same definition as in the core package. It indicates that a particular leg of a telephone call has failed for some reason, perhaps because the party was busy.
`CallControlConnection.UNKNOWN`	This state has the same definition as in the core package. It indicates the implementation is unable to determine the current call control package state of the Connection object. Typically, methods are invalid on this object when it is in this state.

State Transitions

Similar to the `Connection` state transition diagram, the `CallControlConnection` state must transition according to rules illustrated in the finite state diagram below. An implementation must guarantee that `CallControlConnection` state must abide by this transition diagram.

Note there is a general left-to-right progression of the state transitions. The asterisk next to a state transition, as in the core package, implies a transition to/from another state, except where noted.



Core vs. CallControl Package States

There is a strong relationship between the `CallControlConnection` states and the `Connection` states. If an implementation supports the call control package, it must ensure this relationship is properly maintained.

Since the states defined in the `CallControlConnection` interface provide more detail to the states defined in the `Connection` interface, each state in the `Connection` interface corresponds to a state defined in the `CallControlConnection` interface. Conversely, each `CallControlConnection` state corresponds to exactly one `Connection` state. This arrangement permits applications to view either the core state or the call control state and still see a consistent view.

The following table outlines the relationship between the core package Connection states and the call control package Connection states.

If the call control package state is...	***then the core package state must be...***
`CallControlConnection.IDLE`	`Connection.IDLE`
`CallControlConnection.QUEUED`	`Connection.INPROGRESS`
`CallControlConnection.OFFERED`	`Connection.INPROGRESS`
`CallControlConnection.ALERTING`	`Connection.ALERTING`
`CallControlConnection.INITIATED`	`Connection.CONNECTED`
`CallControlConnection.DIALING`	`Connection.CONNECTED`
`CallControlConnection.NETWORK_REACHED`	`Connection.CONNECTED`
`CallControlConnection.NETWORK_ALERTING`	`Connection.CONNECTED`

CallControlConnection.ESTABLISHED	Connection.CONNECTED
CallControlConnection.DISCONNECTED	Connection.DISCONNECTED
CallControlConnection.FAILED	Connection.FAILED
CallControlConnection.UNKNOWN	Connection.UNKNOWN

Observers and Events

All events pertaining to the `CallControlConnection` interface are reported via the `CallObserver.callChangedEvent()` method. The application observer object must also implement the `CallControlCallObserver` interface to express interest in the call control package events.

Observers which are registered on a Call receive events when the `CallControlConnection` state changes. Note that when the `CallControlConnection` state changes, it sometimes results in the `Connection` state changing (according to the table above). In these instances, both the proper call control and core package events are delivered to the observer.

The `CallControlConnection` state events defined in this package are: `CallCtlConnOfferedEv`, `CallCtlConnQueuedEv`, `CallCtlConnAlertingEv`, `CallCtlConnInitiatedEv`, `CallCtlConnDialingEv`, `CallCtlConnNetworkReachedEv`, `CallCtlConnNetworkAlertingEv`, `CallCtlConnFailedEv`, `CallCtlConnEstablishedEv`, `CallCtlConnUnknownEv`, and `CallCtlConnDisconnectedEv`.

See Also:

Connection, CallObserver, CallControlCallObserver, CallCtlCallEv, CallCtlConnEv, CallCtlConnAlertingEv, CallCtlConnDialingEv, CallCtlConnDisconnectedEv, CallCtlConnEstablishedEv, CallCtlConnFailedEv, CallCtlConnInitiatedEv, CallCtlConnNetworkAlertingEv, CallCtlConnNetworkReachedEv, CallCtlConnOfferedEv, CallCtlConnQueuedEv, CallCtlConnUnknownEv

Variable Index

- **ALERTING**

  The `CallControlConnection.ALERTING` state has the same definition as in the core package.
- **DIALING**

  The `CallControlConnection.DIALING` state indicates the originating end of a telephone call has begun the process of dialing a destination telephone address, but has not yet completed.
- **DISCONNECTED**

  The `CallControlConnection.DISCONNECTED` state has the same definition as in the core package.
- **ESTABLISHED**

  The `CallControlConnection.ESTABLISHED` state is similar to that of `Connection.CONNECTED`.
- **FAILED**

  The `CallControlConnection.FAILED` state has the same definition as in the core package.
- **IDLE**

  The `CallControlConnection.IDLE` state has the same definition as in the core package.
- **INITIATED**

  The `CallControlConnection.INITIATED` state indicates the originating end of a telephone call has begun the process of placing a telephone call, but the dialing of the destination telephone address has not yet begun.
- **NETWORK_ALERTING**

  The `CallControlConnection.NETWORK_ALERTING` state indicates that an outgoing telephone call is alerting at the destination end, which was previously only known to have reached the network.
- **NETWORK_REACHED**

  The `CallControlConnection.NETWORK_REACHED` state indicates that an outgoing telephone call has reached the network.
- **OFFERED**

  The `CallControlConnection.OFFERED` state indicates than an incoming call is being offered to the Address associated with the Connection.
- **OFFERING**

The `CallControlConnection.OFFERING` state has been deprecated in JTAPI v1.2. **Deprecated.**

- **QUEUED**

  The `CallControlConnection.QUEUED` state indicates that a Connection is queued at the particular Address associated with the Connection.

- **UNKNOWN**

  The `CallControlConnection.UNKNOWN` state has the same definition as in the core package.

Method Index

- **accept**()

  Accepts a telephone call incoming to an Address.

- **addToAddress**(String)

  Appends additional address information onto an existing Connection.

- **getCallControlState**()

  Returns the current call control state of the Connection.

- **park**(String)

  Parks a Connection at a destination telephone address.

- **redirect**(String)

  Redirects an incoming telephone call at an Address to another telephone address.

- **reject**()

  Rejects a telephone call incoming to an Address.

Variables

IDLE

```
public static final int IDLE
```

The `CallControlConnection.IDLE` state has the same definition as in the core package. It is the initial `CallControlConnection` state for all new Connections. Connections typically do not stay in this state for long, quickly transitioning to another state.

OFFERED

```
public static final int OFFERED
```

The `CallControlConnection.OFFERED` state indicates than an incoming call is being offered to the Address associated with the Connection. Typically, applications must either accept or reject this offered call before the called party is alerted to the incoming telephone call.

QUEUED

```
public static final int QUEUED
```

The `CallControlConnection.QUEUED` state indicates that a Connection is queued at the particular Address associated with the Connection. A queued call is not active on a call. For example, some telephony platforms permit the "queueing" of incoming telephone call to an Address when the Address is busy.

ALERTING

```
public static final int ALERTING
```

The `CallControlConnection.ALERTING` state has the same definition as in the core package. It means that the Address is being notified of an incoming call.

INITIATED

```
public static final int INITIATED
```

The `CallControlConnection.INITIATED` state indicates the originating end of a telephone call has begun the process of placing a telephone call, but the dialing of the destination telephone address has not yet begun. Typically, a telephone associated with the Address has gone "off-hook".

DIALING

```
public static final int DIALING
```

The `CallControlConnection.DIALING` state indicates the originating end of a telephone call has begun the process of dialing a destination telephone address, but has not yet completed.

NETWORK_REACHED

```
public static final int NETWORK_REACHED
```

The `CallControlConnection.NETWORK_REACHED` state indicates that an outgoing telephone call has reached the network. Applications may not receive further events about this leg of the telephone call, depending upon the ability of the telephone network to provide additional progress information. Applications must decide whether to treat this as a connected telephone call.

NETWORK_ALERTING

```
public static final int NETWORK_ALERTING
```

The `CallControlConnection.NETWORK_ALERTING` state indicates that an outgoing telephone call is alerting at the destination end, which was previously only known to have reached the network. Typically, Connections transition into this state from the `CallControlConnection.NETWORK_REACHED` state. This state results from additional progress information being sent from a telephone network that was capable of transmitting that information.

ESTABLISHED

```
public static final int ESTABLISHED
```

The `CallControlConnection.ESTABLISHED` state is similar to that of `Connection.CONNECTED`. It indicates that the endpoint has reached its final, active state in the telephone call.

DISCONNECTED

```
public static final int DISCONNECTED
```

The `CallControlConnection.DISCONNECTED` state has the same definition as in the core package. It indicates that the Connection object is no longer part of the telephone call.

FAILED

```
public static final int FAILED
```

The `CallControlConnection.FAILED` state has the same definition as in the core package. It indicates that a Connection is no longer able to participate in a call. It is a final state in the life of a Connection. It indicates that a particular leg of a telephone call has failed for some reason, perhaps because the party was busy.

UNKNOWN

```
public static final int UNKNOWN
```

The `CallControlConnection.UNKNOWN` state has the same definition as in the core package. It indicates that the state of the Connection is not known to its Provider. Typically, methods are invalid on this object when it is in this state.

OFFERING

```
public static final int OFFERING
```

Note: OFFERING is deprecated. *Since JTAPI v1.2.*

The `CallControlConnection.OFFERING` state has been deprecated in JTAPI v1.2. It has the same meaning as the new `CallControlConnection.OFFERED` state. This constant has been replaced to be more tense-consistent with the event name.

Methods

getCallControlState

```
 public abstract int getCallControlState()
```

Returns the current call control state of the Connection. The return value will be one of integer state constants defined above.

Returns:

The current call control state of the Connection.

accept

```
 public abstract void accept() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Accepts a telephone call incoming to an Address. Telephone calls into an Address may first be *offered* to that address for acceptance before the standard notion of "alerting" takes place. This method is valid on a Connection in the `CallControlConnection.OFFERED` state. If successful, this method moves the Connection into the `CallControlConnection.ALERTING` state. This method waits until the telephone call has been accepted or an error occurs and an exception is thrown.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.OFFERED

Post-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.ALERTING
3. CallCtlConnAlertingEv/ConnAlertingEv is delivered for this Connection

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "offered".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnAlertingEv, CallCtlConnAlertingEv

reject

```
 public abstract void reject() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Rejects a telephone call incoming to an Address. Telephone calls into an Address may first be *offered* to that address for acceptance before the standard notion of "alerting" takes place. This method is valid on a Connection in the `CallControlConnection.OFFERED` state. If successful, this method moves the Connection into the `CallControlConnection.DISCONNECTED` state. This method waits until the telephone call has been rejected or an error occurs and an exception is thrown.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.OFFERED

Post-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.DISCONNECTED
3. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for this Connection

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "offered".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnDisconnectedEv, CallCtlConnDisconnectedEv

redirect

```
 public abstract Connection redirect(String destinationAddress) throws
InvalidStateException, InvalidPartyException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

Redirects an incoming telephone call at an Address to another telephone address. This method is very similar to the transfer feature, however, applications may invoke this method before first answering the telephone call. This method redirects the telephone call to another telephone address string provided as the argument to this method. This telephone address string must be valid and complete.

This Connection must either be in the `CallControlConnection.OFFERED` state or the `CallControlConnection.ALERTING` state. If successful, this method moves the Connection to the `CallControlConnection.DISCONNECTED` state. Additionally, any TerminalConnections associated with this Connection will move to the `CallControlTerminalConnection.DROPPED` state.

A new Connection is created and returned corresponding to the new destination leg of the telephone call. Note that this Connection may be `null` in the case the Call has been redirected outside of the Provider's domain and can no longer be tracked. The new Connection (if not null) must at least be in the `CallControlConnection.IDLE` state. The Connection may progress beyond this state before this method returns, which should be reflected by the proper events. This Connection behaves similarly to the destination Connection as described in `Call.connect()` and undergoes similar possible state change scenarios.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.OFFERED or CallControlConnection.ALERTING
3. destinationAddress must be a valid and complete telephone address

Post-conditions:

1. Let newconnection be the new Connection created and returned
2. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
3. this.getCallControlState() == CallControlConnection.DISCONNECTED
4. newconnection element of (this.getCall()).getConnections()
5. newconnection.getCallControlState() at least CallControlConnection.IDLE
6. Let tc[] = this.getTerminalConnections() before this method is invoked
7. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED, for all i.
8. CallCtlConnDisconnected/ConnDisconnectedEv is delivered for this Connection
9. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]
10. ConnCreatedEv is delivered for newconnection

Parameters:

desintationAddress - The Connection is redirected to this telephone address

Returns:

The Connection associated with the new leg of the Call.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "offered" or "alerting".

Throws: InvalidPartyException

The destination address to which this call is redirected is not valid and/or complete.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

addToAddress

```
 public abstract void addToAddress(String additionalAddress) throws
InvalidStateException, MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Appends additional address information onto an existing Connection. This method is used when part of a telephone address string has been dialed and additional addressing information is needed in order to complete the dialing process and place the telephone call. The additional addressing information is provided as the argument to this method.

This Connection must either be in the `CallControlConnection.DIALING` state or the `CallControlConnection.INITIATED` state. If successful, this moves the Connection into one of two states. If the information provided completes the addressing information, as determined by the telephony platform, the Connection moves into the `CallControlConnection.ESTABLISHED` state and the telephone call is placed. If additional addressing information is still required, the Connection moves into the `CallControlConnection.DIALING` state if not already there.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.DIALING or CallControlConnection.INITIATED

Post-conditions Outcome 1: The addressing information is complete.

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.ESTABLISHED
3. CallCtlConnEstablishedEv is delivered for this Connection

Post-conditions Outcome 2: The addressing information is not complete.

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.DIALING
3. CallCtlConnDialingEv is delivered for this Connection

Parameters:

additionalAddress - The additional addressing information.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "initiated" or "dialing".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

CallCtlConnDialingEv, CallCtlConnEstablishedEv

park

```
 public abstract Connection park(String destinationAddress) throws
InvalidStateException, MethodNotSupportedException, PrivilegeViolationException,
InvalidPartyException, ResourceUnavailableException
```

Parks a Connection at a destination telephone address. This method is similar to the transfer feature, except the Connection at the new destination Address is in a special *queued* state. Parking a Connection at a destination Address drops the Connection from the Call and creates and returns a new Connection at the specified destination address in the `CallControlConnection.QUEUED` state.

The new destination telephone address string is given as an argument to this method and must be a valid and complete telephone address. The `CallControlTerminal.pickup()` method permits applications to "unpark" the new Connection.

The Connection must be in the `CallControlConnection.ESTABLISHED` state. If this method is successful, this Connection moves to the `CallControlConnection.DISCONNECTED` state. All of its associated TerminalConnections move to the `CallControlTerminalConnection.DROPPED` state.

Pre-conditions:

1. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlConnection.ESTABLISHED
3. destinationAddress must be a valid and complete telephone address.

Post-conditions:

1. Let newconnection be the new Connection created and returned
2. ((this.getCall()).getProvider()).getState() == Provider.IN_SERVICE
3. newconnection element of (this.getCall()).getConnections()
4. newconnection.getCallControlState() == CallControlConnection.QUEUED
5. this.getCallControlState() == CallControlConnection.DISCONNECTED
6. Let tc[] = this.getTerminalConnections() before this method is invoked
7. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED, for all i
8. ConnCreatedEv is delivered for newconnection
9. CallCtlConnQueuedEv/ConnInProgressEv is delivered for newconnection
10. CallCtlConnDisconnected/ConnDisconnectedEv is delivered for this Connection
11. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]

Parameters:

destinationAddress - The telephone address string at which this connection is to be parked.

Returns:

The new Connection which is parked at the new destination Address.

Throws: InvalidStateException

Either the Provider was not "in service" or the Connection was not "established".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: InvalidPartyException

The party to which to party the Connection is invalid.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, ConnInProgressEv, ConnDisconnectedEv, TermConnDroppedEv, CallCtlConnQueuedEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

---

All Packages   This Package   Previous   Next

Interface javax.telephony.callcontrol.CallControlTerminal

public interface **CallControlTerminal**

extends Terminal

Introduction

The `CallControlTerminal` interface extends the core `Terminal` interface with features like the ability to pickup a call at a terminal and the ability to specify that this terminal should not be disturbed. Applications may query a Terminal object using the `instanceof` operator to see whether it supports this interface.

Do Not Disturb

The `CallControlTerminal` interface defines the do not disturb attribute. The do not disturb attribute indicates to the telephony platform that this Terminal does not want to be bothered with incoming telephone calls. That is, if this feature is activated, the underlying telephone platform will not ring this Terminal for incoming telephone calls. Applications use the `CallControlTerminal.setDoNotDisturb()` method to activate or deactivate this feature and the `CallControlTerminal.getDoNotDisturb()` method to return the current state of this attribute.

Note that the `CallControlAddress` interface also carries the do not disturb attribute. This attribute associated with each are maintained independently. Maintaining a separate do not distrub attribute at terminal and address allows for control over the do not disturb feature at either the terminal or address level.

Picking Up Telephone Calls

The pickup feature permits applications to answer telephone calls which are not ringing at a particular Terminal. This feature is often used when a call is "queued" at an Address or a telephone is ringing at a Terminal across a room. Both the `pickup()` and the `pickupFromGroup()` methods defined on this interface provide this pickup feature to the application.

The `CallControlTerminal.pickup()` method has three versions and permits applications to answer a Call at a designated Address or Terminal. The `CallControlTerminal.pickupFromGroup()` method permits applications to answer a Call at some other Terminal in the same "pickup group".

Observers and Events

All events pertaining to the `CallControlTerminal` interface are reported via the `TerminalObserver.terminalChangedEvent()` method. The application observer object must also implement the `CallControlCallObserver` interface to express interest in the call control package events.

The following are those events associated with this interface:

`CallCtlTermDoNotDisturbEv`	Indicates the do-not-disturb attribute of this Terminal has changed.

See Also:

CallControlAddress, CallControlTerminalObserver, CallCtlTermDoNotDisturbEv

Method Index

- **getDoNotDisturb**()

  Returns true if the do-not-disturb feature is activated, false otherwise.

- **pickup**(Address, Address)

This method "picks up" a Call at this Terminal.

- **pickup**(Connection, Address)

  This method "picks up" a Call at this Terminal.

- **pickup**(TerminalConnection, Address)

  This method "picks up" a Call at this Terminal.

- **pickupFromGroup**(Address)

  This method "picks up" a Call at this Terminal.

- **pickupFromGroup**(String, Address)

  This method "picks up" a Call at this Terminal.

- **setDoNotDisturb**(boolean)

  Specifies whether the do-not-disturb feature should be activated or deactivated for this Terminal.

Methods

getDoNotDisturb

```
 public abstract boolean getDoNotDisturb() throws MethodNotSupportedException
```

Returns true if the do-not-disturb feature is activated, false otherwise.

Returns:

True if do-not-disturb is activated, false if it is deactivated

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

setDoNotDisturb

```
 public abstract void setDoNotDisturb(boolean enable) throws
MethodNotSupportedException, InvalidStateException
```

Specifies whether the do-not-disturb feature should be activated or deactivated for this Terminal. This feature only affects whether or not calls will be accepted at this Terminal. The setting of this attribute does not affect the do-not-disturb attribute associated with all Addresses associated with this Terminal. If 'enable' is true, do-not-disturb is activated if not already so. If 'enable' is false, do-not-disturb is deactivated if not already so.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getDoNotDisturb() == enable
3. CallCtlTermDoNotDisturbEv is delivered for this Terminal

Parameters:

enable - True to activated do-not-disturb, false to deactivate.

Throws: MethodNotSupportedException

This method is not supported by the given implementation.

Throws: InvalidStateException

The Provider is not "in service".

See Also:

CallCtlTermDoNotDisturbEv

pickup

```
 public abstract TerminalConnection pickup(Connection pickupConnection,
                                           Address terminalAddress) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method "picks up" a Call at this Terminal. Picking up a Call is analogous to answering a Call at this Terminal (i.e. `TerminalConnection.answer()`), except the Call typically is not ringing at this Terminal. For example, this method is used to answer a "queued" Call or a Call which is ringing at another Terminal across the room.

This method takes a Connection and an Address as arguments. The Connection argument represents the destination end of the telephone call to be picked up. This Connection must be in either the `CallControlConnection.QUEUED` state or the `CallControlConnection.ALERTING` state. The Address argument chooses the Address associated with this Terminal on which to pick up the Call. A new TerminalConnection is created and returned which is in the `CallControlTerminalConnection.TALKING` state and associated with this Terminal.

The Address and Connection Arguments

The relationship between the Address and Connection arguments affects the resulting behavior of this method. There are two different situations: if the given Connection is associated with the given Address, and if the given Connection is not associated with the given Address (i.e. via the `Connection.getAddress()` method).

If the given Connection is associated with the given Address, this implies that the Connection was in the `CallControlConnection.QUEUED` state, or the Terminal did not ring for some reason even though the Connection is in the `CallControlConnection.ALERTING` state. In this case, the Connection moves to the `CallControlConnection.ESTABLISHED` state and the new TerminalConnection created is associated with the Connection.

If the given Connection is not associated with the given Address, this implies that the call is alerting at an entirely different endpoint from this Terminal. This scenario permits applications to pick up a telephone call which is ringing across the room. In this case, the Connection moves to the `CallControlConnection.DISCONNECTED` state. A new Connection is created and associated with the Address argument. It is in the `CallControlConnection.ESTABLISHED` state. The new TerminalConnection created is associated with this new Connection.

Pre-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. (pickupConnection.getCall()).getState() == Call.ACTIVE
3. pickupConnection.getCallControlState() == CallControlConnection.QUEUED or CallControlConnection.ALERTING
4. terminaladdress element of this.getAddresses()

Post-conditions Outcome 1: If pickupConnection is associated with terminaladdress (i.e. pickupConnection.getAddress() == terminaladdress)

1. Let tc be the new TerminalConnection created and returned
2. tc.getCallControlState() == CallControlTerminalConnection.TALKING
3. pickupConnection.getCallControlState() == CallControlConnection.ESTABLISHED
4. tc.getConnection() == pickupConnection
5. TermConnCreatedEv is delivered for tc
6. CallCtlTermConnTalkingEv/TermConnActiveEv is delivered for tc
7. CallCtlConnEstablishedEv/ConnConnectedEv is delivered for pickupConnection

Post-conditions Outcome 2: If pickupConnection is not associated with terminaladdress (i.e. pickupConnection.getAddress() != terminaladdress)

1. Let tc be the new TerminalConnection created and returned
2. Let connection be the new Connection created
3. Let call = pickupConnection.getCall()
4. tc.getCallControlState() == CallControlTerminalConnection.TALKING
5. connection.getCallControlState() == CallControlConnection.ESTABLISHED
6. connection.getAddress() == terminaladdress

7. connection.getCall() == call
8. tc.getConnection() == connection
9. pickupConnection.getCallControlState() == CallControlConnection.DISCONNECTED
10. TermConnCreatedEv is delivered for tc
11. CallCtlTermConnTalkingEv/TermConnActiveEv is delivered for tc
12. ConnCreatedEv is delivered for connection
13. CallCtlConnEstablishedEv/ConnConnectedEv is delivered for connection
14. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for pickupConnection

Parameters:

pickupConnection - The Connection to be picked up

terminalAddress - The Address associated with the Terminal

Returns:

The new TerminalConnection associated with the Terminal

Throws: InvalidArgumentException

Either the Connection or Address given as arguments is not valid.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "queued" or "alerting".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

ConnCreatedEv, TermConnCreatedEv, ConnConnectedEv, ConnDisconnectedEv, TermConnActiveEv, CallCtlTermConnTalkingEv, CallCtlConnEstablishedEv, CallCtlConnDisconnectedEv

pickup

```
 public abstract TerminalConnection pickup(TerminalConnection pickTermConn,
                                           Address terminalAddress) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method "picks up" a Call at this Terminal. Picking up a Call is analogous to answering a Call at this Terminal (i.e. `TerminalConnection.answer()`), except the Call typically is not ringing at this Terminal. For example, this method is used to answer a "queued" Call or a Call which is ringing at another Terminal across the room.

This method is similar to the `CallControlTerminal.pickup(Connection, Address)` method except an explicit TerminalConnection is given. Since an explicit TerminalConnection is given, this implies its Connection must be "alerting" since "queued" Connections may not have any associated TerminalConnections.

Parameters:

pickTermConn - The TerminalConnection to be picked up

terminalAddress - The Address associated with the Terminal

Returns:

The new TerminalConnection associated with the Terminal

Throws: InvalidArgumentException

Either the Connection or Address given as arguments is not valid.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "queued".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

pickup

```
 public abstract TerminalConnection pickup(Address pickupAddress,
                                           Address terminalAddress) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method "picks up" a Call at this Terminal. Picking up a Call is analogous to answering a Call at this Terminal (i.e. `TerminalConnection.answer()`), except the Call typically is not ringing at this Terminal. For example, this method is used to answer a "queued" Call or a Call which is ringing at another Terminal across the room.

This method is similar to the `CallControlTerminal.pickup(Connection, Address)` method except an Address is given instead of an explicit Connection. This method permits the implementation to choose a suitable Connection associated with 'pickupAddress'.

Parameters:

pickupAddress - The Address which is to be picked up

terminalAddress - The Address associated with the Terminal

Returns:

The new TerminalConnection associated with the Terminal

Throws: InvalidArgumentException

Either the Connection or Address given as arguments is not valid.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "queued" or "alerting".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

pickupFromGroup

```
 public abstract TerminalConnection pickupFromGroup(String pickupGroup,
                                                    Address terminalAddress) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method "picks up" a Call at this Terminal. Picking up a Call is analogous to answering a Call at this Terminal (i.e. `TerminalConnection.answer()`), except the Call typically is not ringing at this Terminal. For example, this method is used to answer a "queued" Call or a Call which is ringing at another Terminal across the room.

This method takes two arguments: a string "pickup group" code and an Address associated with this Terminal. The Address argument chooses the Address associated with this Terminal on which to pick up the Call. A new TerminalConnection is created and returned which is in the `CallControlTerminalConnection.TALKING` state and associated with this Terminal.

The Pickup Group Code

The application designates the Call to pick up via a string code rather than giving a specific Connection endpoint. An administrator of the underlying telephony platform can create groups of endpoints associated with a particular group code. From the code, the implementations decides which particular Connection is to be picked up. Once the Connection has been determined, this method behavior similarly to the `CallControlTerminal.pickup(Connection, Address)` method.

Parameters:

pickupGroup - The pickup group

terminalAddress - The Address associated with the Terminal

Returns:

The new TerminalConnection associated with the Terminal

Throws: InvalidArgumentException

Either the Connection or Address given as arguments is not valid.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "queued" or "alerting".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

pickupFromGroup

```
 public abstract TerminalConnection pickupFromGroup(Address terminalAddress) throws
InvalidArgumentException, InvalidStateException, MethodNotSupportedException,
PrivilegeViolationException, ResourceUnavailableException
```

This method "picks up" a Call at this Terminal. Picking up a Call is analogous to answering a Call at this Terminal (i.e. `TerminalConnection.answer()`), except the Call typically is not ringing at this Terminal. For example, this method is used to answer a "queued" Call or a Call which is ringing at another Terminal across the room.

This method takes a single argument: an Address associated with this Terminal. The Address argument chooses the Address associated with this Terminal on which to pick up the Call. A new TerminalConnection is created and returned which is in the `CallControlTerminalConnection.TALKING` state and associated with this Terminal.

The Pickup Group Code

This method does not take the pickup group code as an argument. Instead, the implementation chooses a suitable Call to be picked up. This Call should have a Connection in either the `CallControlConnection.ALERTING` state or the `CallControlConnection.QUEUED` state. The Address associated with this Connection should belong to the same pickup group as the Address given as the argument. Once the Connection has been determined, this method behavior similarly to the `CallControlTerminal.pickup(Connection, Address)` method.

Parameters:

terminalAddress - The Address associated with the Terminal

Returns:

The new TerminalConnection associated with the Terminal

Throws: InvalidArgumentException

Either the Connection or Address given as arguments is not valid.

Throws: InvalidStateException

Either the Provider is not "in service" or the Connection is not "queued" or "alerting".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.CallControlTerminalConnection

public interface **CallControlTerminalConnection**

extends TerminalConnection

Introduction

The `CallControlTerminalConnection` interface extends the core `TerminalConnection` interface with additional features and greater detail about the TerminalConnection state. Applications may query a TerminalConnection object using the `instanceof` operator to see whether it supports this interface.

CallControlTerminalConnection State

This interface defines states for the TerminalConnection which provide greater detail beyond the states defined in the `TerminalConnection` interface. These states are related to the states defined in the core package in certain, specific ways, as defined below. Applications may obtain the `CallControlTerminalConnection` state via the `getCallControlState()` method defined on this interface. This method returns one of the integer state constants defined in this interface.

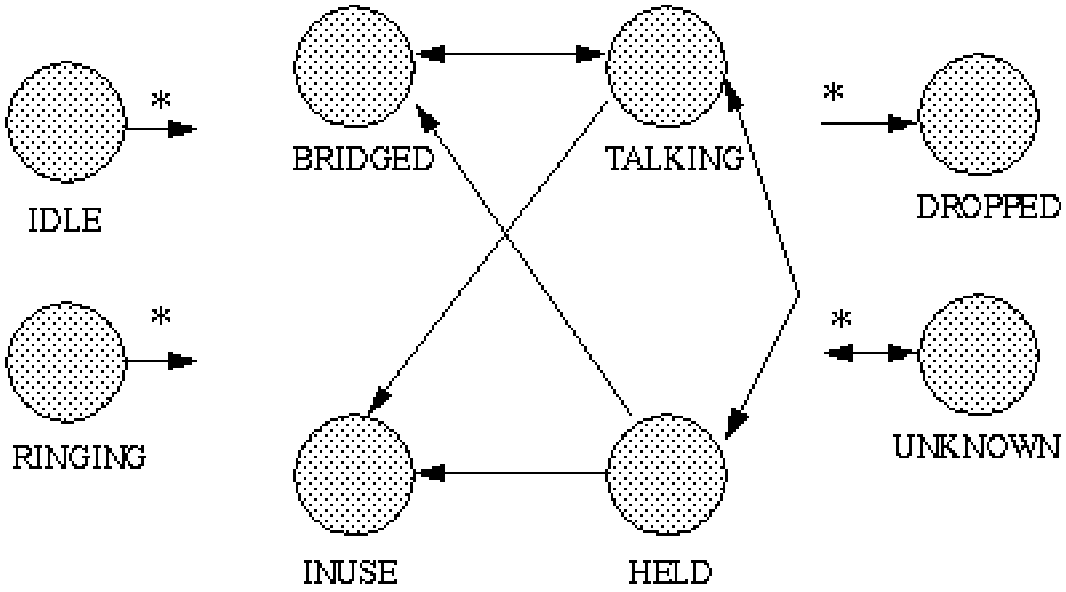
Below is a description of each `CallControlTerminalConnection` state in real-world terms. These real-world descriptions only serve to provide a more intuitive understanding of what is going on. Several methods in this specification state pre-conditions based upon the `CallControlTerminalConnection` state. Some of these states are identical to those defined in the core package.

`CallControlTerminalConnection.IDLE`	This state has the same definition as in the core package. It is the initial `CallControlTerminalConnection` state for all new TerminalConnection objects. TerminalConnections typically do not stay in this state for long, quickly transitioning to another state.
`CallControlTerminalConnection.RINGING`	This state has the same definition as in the core package. It indicates that the associated Terminal is ringing and has an incoming telephone call.
`CallControlTerminalConnection.TALKING`	This state indicates that the Terminal is actively part of a Call, is typically "off-hook", and the party is communicating on the telephone call.
`CallControlTerminalConnection.HELD`	This state indicates that a Terminal is part of a Call, but is on hold. Other Terminals which are on the same Call and associated with the same Connection may or may not also be in this state.
`CallControlTerminalConnection.BRIDGED`	This state indicates that a Terminal is currently bridged into a Call. A Terminal may typically join a Call when it is bridged. A bridged Terminal is part of the telephone call, but not active. Typically, some hardware resource is occupied while a Terminal is bridged into a Call.
`CallControlTerminalConnection.INUSE`	This state indicates that a Terminal hardware resource is currently in use. The terminal is not available for the Call associated with this Terminal Connection, that is the Terminal may not join the call. This state is similar to the `CallControlTerminalConnection.BRIDGED` state except that the Terminal may not join the Call.
`CallControlTerminalConnection.DROPPED`	This state has the same definition as in the core package. It indicates that a particular Terminal has permanently left the Call.
`CallControlTerminalConnection.UNKNOWN`	This state has the same definition as in the core package. It indicates that the implementation is unable to determine the state of the TerminalConnection. TerminalConnections may transition into and out of this state at any time.

State Transitions

Similar to the `TerminalConnection` state transition diagram, the `CallControlTerminalConnection` state must transition according to the rules illustrated in the finite state diagram below. The implementation must guarantee that the `CallControlTerminalConnection` state abides by this transition diagram.

The asterisk next to a state transition, as in the core package, implies a transition to/from another state as designated by the direction of the transition arrow.



Core vs. CallControl Package States

There is a strong relationship between the `TerminalConnection` states and the `CallControlTerminalConnection` states. If an implementation supports the call control package, it must ensure this relationship is properly maintained.

Since the states defined in the `CallControlTerminalConnection` interface provide more detail to the states defined in the `TerminalConnection` interface, each state in the `TerminalConnection` interface corresponds to a state defined in the `CallControlTerminalConnection` interface. Conversely, each `CallControlTerminalConnection` state corresponds to exactly one `TerminalConnection` state. This arrangement permits applications to view either the core state or the call control state and still see a consistent view.

The following table outlines the relationship between the core package TerminalConnection states and the call control package TerminalConnection states.

If the call control package state is...	***then the core package state must be...***
`CallControlTerminalConnection.IDLE`	`TerminalConnection.IDLE`
`CallControlTerminalConnection.RINGING`	`TerminalConnection.RINGING`
`CallControlTerminalConnection.TALKING`	`TerminalConnection.ACTIVE`
`CallControlTerminalConnection.HELD`	`TerminalConnection.ACTIVE`
`CallControlTerminalConnection.INUSE`	`TerminalConnection.PASSIVE`
`CallControlTerminalConnection.BRIDGED`	`TerminalConnection.PASSIVE`
`CallControlTerminalConnection.DROPPED`	`TerminalConnection.DROPPED`
`CallControlTerminalConnection.UNKNOWN`	`TerminalConnection.UNKNOWN`

Observers and Events

All events pertaining to the `CallControlTerminalConnection` interface are reported via the `CallObserver.callChangedEvent()` method. The application observer object must also implement the `CallControlCallObserver` interface to express interest in the call control package events. Applications receive TerminalConnection-related events in the call control package when the call control state changes.

Observers which are registered on a Call receive events when the `CallControlTerminalConnection` state changes. Note that when the `CallControlTerminalConnection` state changes, it sometimes results in the `TerminalConnection` state changing (according to the table above). In these instances, both the proper call control and core package events are delivered to the observer.

The `CallControlTerminalConnection` state events defined in this package are: `CallCtlTermConnBridgedEv`, `CallCtlTermConnDroppedEv`, `CallCtlTermConnHeldEv`, `CallCtlTermConnInUseEv`, `CallCtlTermConnRingingEv`, `CallCtlTermConnTalkingEv`, and `CallCtlTermConnUnknownEv`.

See Also:

TerminalConnection, CallObserver, CallControlCallObserver, CallCtlCallEv, CallCtlTermConnEv, CallCtlTermConnRingingEv, CallCtlTermConnTalkingEv, CallCtlTermConnHeldEv, CallCtlTermConnBridgedEv, CallCtlTermConnInUseEv, CallCtlTermConnDroppedEv, CallCtlTermConnUnknownEv

---

Variable Index

BRIDGED

The `CallControlTerminalConnection.BRIDGED` state indicates that a Terminal is currently bridged into a Call.

DROPPED

The `CallControlTerminalConnection.DROPPED` state has the same definition as in the core package.

HELD

The `CallControlTerminalConnection.HELD` state indicates that a Terminal is part of a Call, but is on hold.

IDLE

The `CallControlTerminalConnection.IDLE` state has the same definition as in the core package.

INUSE

The `CallControlTerminalConnection.INUSE` state indicates that a Terminal is part of a Call, but is not active.

RINGING

The `CallControlTerminalConnection.RINGING` state has the same definition as in the core package.

TALKING

The `CallControlTerminalConnection.TALKING` state indicates that a Terminal is actively part of a Call, is typically "off-hook", and the party is communicating on the telephone call.

UNKNOWN

The `CallControlTerminalConnection.UNKNOWN` state has the same definition as in the core package.

Method Index

getCallControlState()

Returns the call control state of the TerminalConnection.

hold()

Places a TerminalConnection on hold with respect to the Call of which it is a part.

join()

Makes a currently bridged TerminalConnection active on a Call.

leave()

Places a currently active TerminalConnection in a *bridged* state on a Call.

unhold()

Takes a TerminalConnection off hold with respect to the Call of which it is a part.

Variables

IDLE

```
public static final int IDLE
```

The `CallControlTerminalConnection.IDLE` state has the same definition as in the core package. It is the initial `CallControlTerminalConnection` state for all new TerminalConnections. TerminalConnections typically do not stay in this state for long, but quickly transition to another state.

RINGING

```
public static final int RINGING
```

The `CallControlTerminalConnection.RINGING` state has the same definition as in the core package. It indicates that the associated Terminal is ringing and has an incoming telephone call.

TALKING

```
public static final int TALKING
```

The `CallControlTerminalConnection.TALKING` state indicates that a Terminal is actively part of a Call, is typically "off-hook", and the party is communicating on the telephone call.

HELD

```
public static final int HELD
```

The `CallControlTerminalConnection.HELD` state indicates that a Terminal is part of a Call, but is on hold. Other Terminals which are on the same Call and associated with the same Connection may or may not also be in this state.

BRIDGED

```
public static final int BRIDGED
```

The `CallControlTerminalConnection.BRIDGED` state indicates that a Terminal is currently bridged into a Call. A Terminal may typically join a telephone call when it is bridged. A bridged Terminal is part of the telephone call, but not active. Typically, some hardware resource is occupied while a Terminal is bridged into a Call.

INUSE

```
public static final int INUSE
```

The `CallControlTerminalConnection.INUSE` state indicates that a Terminal is part of a Call, but is not active. It may not Call, however the resource on the Terminal is currently in use. This state is similar to the `CallControlTerminalConnection.BRIDGED` state however, the Terminal may not join the Call.

DROPPED

```
public static final int DROPPED
```

The `CallControlTerminalConnection.DROPPED` state has the same definition as in the core package. It indicates that a particular Terminal has permanently left the Call.

UNKNOWN

```
public static final int UNKNOWN
```

The `CallControlTerminalConnection.UNKNOWN` state has the same definition as in the core package. It indicates that the implementation is unable to determine the state of the TerminalConnection. TerminalConnections may transition into and out of this state at any time.

Methods

getCallControlState

```
 public abstract int getCallControlState()
```

Returns the call control state of the TerminalConnection. The return values will be one of the integer state constants defined above.

Returns:

The current call control state of the TerminalConnection.

hold

```
 public abstract void hold() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Places a TerminalConnection on hold with respect to the Call of which it is a part. Many Terminals may be on the same Call and associated with the same Connection. Any one of them may go "on hold" at any time, provided they are active in the Call. The TerminalConnection must be in the `CallControlTerminalConnection.TALKING` state. This method returns when the TerminalConnection has moved to the `CallControlTerminalConnection.HELD` state, or until an error occurs and an exception is thrown.

Pre-conditions:

1. (this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.TALKING

Post-conditions:

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.HELD
3. CallCtlTermConnHeldEv is delivered for this TerminalConnection

Throws: InvalidStateException

Either the Provider is not "in service" or the TerminalConnection is not "talking".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

CallCtlTermConnHeldEv

unhold

```
 public abstract void unhold() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Takes a TerminalConnection off hold with respect to the Call of which it is a part. Many Terminals may be on the same Call and associated with the same Connection. Any one of them may go "on hold" at any time, provided they are active in the Call. The TerminalConnection must be in the `CallControlTerminalConnection.HELD` state. This method returns successfully when the TerminalConnection moves into the `CallControlTerminalConnection.TALKING` state or until an error occurs and an exception is thrown.

Pre-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.HELD

Post-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.TALKING
3. CallCtlTermConnTalkingEv is delivered for this TerminalConnection

Throws: InvalidStateException

Either the Provider is not "in service" or the TerminalConnection is not "held".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

CallCtlTermConnTalkingEv

join

```
 public abstract void join() throws InvalidStateException,
MethodNotSupportedException, PrivilegeViolationException,
ResourceUnavailableException
```

Makes a currently bridged TerminalConnection active on a Call. Other Terminals, which share the same Address as this Terminal, may be active on the same Call. This situation is known as *bridging*. The TerminalConnection must be in the `CallControlTerminalConnection.BRIDGED` state. This method returns which the TerminalConnection has moved to the `CallControlTerminalConnection.TALKING` state or until an error occurs and an exception is thrown.

Pre-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.BRIDGED

Post-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.TALKING
3. CallCtlTermConnTalkingEv/TermConnActiveEv is delivered for this TerminalConnection

Throws: InvalidStateException

Either the Provider is not "in service" or the TerminalConnection is not "bridged".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

TermConnActiveEv, CallCtlTermConnTalkingEv

leave

```
 public abstract void leave() throws InvalidStateException,
```

`MethodNotSupportedException`, `PrivilegeViolationException`,
`ResourceUnavailableException`

Places a currently active TerminalConnection in a *bridged* state on a Call. Other Terminals, which share the same Address as this Terminal, may remain active on the same Call. This situation where Terminals share an Address on a call is known as *bridging*. The TerminalConnection on which this method is invoked must be in the `CallControlTerminalConnection.TALKING` state.

There are two possible outcomes of this method depending upon the number of remaining, active TerminalConnections on the Call. If there are other active TerminalConnections, then this TerminalConnection moves into the `CallControlTerminalConnection.BRIDGED` state and this method returns. If there are no other active TerminalConnections, then this TerminalConnection moves into the `CallControlTerminalConnection.DROPPED` state. Its associated Connection moves into the `CallControlConnection.DISCONNECTED` state, i.e. the entire endpoint leaves the telephone call. This method waits until one of these two outcomes occurs or until an error occurs and an exception is thrown.

Pre-conditions:

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.TALKING

Post-conditions Outcome 1: There are no other active TerminalConnections on this Call (no others which are either "held" or "talking"

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. Let connection = this.getConnection()
3. Let tc[] = connection.getTerminalConnections() before this method is invoked
4. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED, for all i
5. connection.getCallControlState() == CallControlConnection.DISCONNECTED
6. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]
7. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for connection

Post-conditions Outcome 2: There are other active TerminalConnections on this Call (others which are either "held" or "talking"

1. ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE
2. this.getCallControlState() == CallControlTerminalConnection.BRIDGED
3. CallCtlTermConnBridgedEv/TermConnPassiveEv is delivered for this TerminalConnection

Throws: InvalidStateException

Either the Provider is not "in service" or the TerminalConnection is not "talking".

Throws: MethodNotSupportedException

This method is not supported by the implementation.

Throws: PrivilegeViolationException

The application does not have the proper authority to invoke this method.

Throws: ResourceUnavailableException

An internal resource necessary for the successful invocation of this method is not available.

See Also:

TermConnPassiveEv, TermConnDroppedEv, ConnDisconnectedEv, CallCtlTermConnBridgedEv, CallCtlTermConnDroppedEv, CallCtlConnDisconnectedEv

---

`All Packages` `This Package` `Previous` `Next`

Interface javax.telephony.callcontrol.CallControlTerminalObserver

public interface **CallControlTerminalObserver**

extends TerminalObserver

The `CallControlTerminalObserver` interface reports all events for the `CallControlTerminal` interface. Applications implement this interface to receive `CallControlTerminal`-related events. All events are reported via the `TerminalObserver.terminalChangedEvent()` method. This interface, therefore, allows applications to signal to the implementation that they are interested in call control package events. This interface defines no additional methods.

All events must extend the `CallCtlTermEv` event interface, which in turn, extends the core `TermEv` interface.

The following are those events which are associated with this interface:

`CallCtlTermDoNotDisturbEv`	Indicates the Do Not Disturb characteristics of this Terminal has changed.

See Also:

TerminalObserver, TermEv, CallControlTerminal, CallCtlTermEv, CallCtlTermDoNotDisturbEv

Class javax.telephony.callcontrol.CallControlForwarding

```
java.lang.Object
   |
   +----javax.telephony.callcontrol.CallControlForwarding
```

public class **CallControlForwarding**

extends Object

The CallControlForwarding class represents a *forwarding instruction*. This instruction tells how the platform should forward incoming telephone calls to a specific address. There are several attributes to a forwarding instruction.

The first attribute is its *type*. The forwarding instruction's type tells the platform when to forward the call. There are currently three types of instructions: telling the platform to always forward incoming calls, telling the platform to forward incoming calls when the address is busy, and telling the platform to forward incoming calls when no one answers.

The second attribute of a forwarding instruction is its *filter*. The filter indicates which type of incoming calls this forwarding instruction should apply to. This forwarding instruction can apply to all calls, to external calls only, to internal calls only, or to a specific calling address.

Variable Index

- **ALL_CALLS**

  Forwarding filter: apply instruction to all incoming calls.
- **EXTERNAL_CALLS**

  Forwarding filter: apply instruction to calls originating from outside the provider domain.
- **FORWARD_ON_BUSY**

  Forwarding type: forward calls on busy.
- **FORWARD_ON_NOANSWER**

  Forwarding type: forward calls on no answer.
- **FORWARD_UNCONDITIONALLY**

  Forwarding type: forward calls unconditionally.
- **INTERNAL_CALLS**

  Forwarding filter: apply instruction to calls originating from the provider domain.
- **SPECIFIC_ADDRESS**

  Forwarding filter: apply instruction to calls originating from a specific address.

Constructor Index

- **CallControlForwarding**(String)

  This constructor is the default constructor, which only takes the address to apply this forwarding instruction.

- **CallControlForwarding**(String, int)

  This constructor takes the address to apply this forwarding instruction and the type of forwarding for all incoming calls.

- **CallControlForwarding**(String, int, boolean)

  This constructor takes the address to apply this forwarding instruction, the type of forwarding desired for this address, and a boolean flag indicating whether this instruction should apply to internal (true) or external (false) calls.

- **CallControlForwarding**(String, int, String)

  This constructor takes an address to apply the forwarding instruction for a specific incoming telephone call, identified by a string address.

Method Index

- **getDestinationAddress**()

  Returns the destination address of this forwarding instruction.

- **getFilter**()

  Returns the filter of this forwarding instruction.

- **getSpecificCaller**()

  If the filter for this forwarding instruction is SPECIFIC_ADDRESS, then this method returns that calling address to which this filter applies.

- **getType**()

  Returns the type of this forwarding instruction, either unconditionally, upon no answer, or upon busy.

Variables

ALL_CALLS

```
public static final int ALL_CALLS
```

Forwarding filter: apply instruction to all incoming calls.

INTERNAL_CALLS

```
public static final int INTERNAL_CALLS
```

Forwarding filter: apply instruction to calls originating from the provider domain.

EXTERNAL_CALLS

```
public static final int EXTERNAL_CALLS
```

Forwarding filter: apply instruction to calls originating from outside the provider domain.

SPECIFIC_ADDRESS

```
public static final int SPECIFIC_ADDRESS
```

Forwarding filter: apply instruction to calls originating from a specific address.

FORWARD_UNCONDITIONALLY

```
public static final int FORWARD_UNCONDITIONALLY
```

Forwarding type: forward calls unconditionally.

FORWARD_ON_BUSY

```
public static final int FORWARD_ON_BUSY
```

Forwarding type: forward calls on busy.

FORWARD_ON_NOANSWER

```
public static final int FORWARD_ON_NOANSWER
```

Forwarding type: forward calls on no answer.

Constructors

CallControlForwarding

```
public CallControlForwarding(String destAddress)
```

This constructor is the default constructor, which only takes the address to apply this forwarding instruction. The forwarding instruction forwards all calls unconditionally.

CallControlForwarding

```
public CallControlForwarding(String destAddress,
                             int type)
```

This constructor takes the address to apply this forwarding instruction and the type of forwarding for all incoming calls.

CallControlForwarding

```
public CallControlForwarding(String destAddress,
                             int type,
                             boolean internalCalls)
```

This constructor takes the address to apply this forwarding instruction, the type of forwarding desired for this address, and a boolean flag indicating whether this instruction should apply to internal (true) or external (false) calls.

CallControlForwarding

```
public CallControlForwarding(String destAddress,
                             int type,
                             String caller)
```

This constructor takes an address to apply the forwarding instruction for a specific incoming telephone call, identified by a string address. It also takes the type of forwarding desired for this specific address.

Methods

getDestinationAddress

```
public String getDestinationAddress()
```

Returns the destination address of this forwarding instruction.

Returns:

The destination address of this forwarding instruction.

getType

```
public int getType()
```

Returns the type of this forwarding instruction, either unconditionally, upon no answer, or upon busy.

Returns:

The type of this forwarding instruction.

getFilter

```
public int getFilter()
```

Returns the filter of this forwarding instruction. The filter indicates which calls should trigger this forwarding instruction. Filters include: applying this instruction to all calls, to only internal calls, to only external call, or for calls from a specific address.

Returns:

The filter for this forwarding instruction.

getSpecificCaller

```
public String getSpecificCaller()
```

If the filter for this forwarding instruction is SPECIFIC_ADDRESS, then this method returns that calling address to which this filter applies. If the filter is something other than SPECIFIC_ADDRESS, this method returns null.

Returns:

The specific address for this forwarding instruction.

All Packages

package javax.telephony.callcontrol.capabilities

Interface Index

- CallControlAddressCapabilities
- CallControlCallCapabilities
- CallControlConnectionCapabilities
- CallControlTerminalCapabilities
- CallControlTerminalConnectionCapabilities

Interface javax.telephony.callcontrol.capabilities.CallControlAddressCapabilities

public interface **CallControlAddressCapabilities**

extends AddressCapabilities

The `CallControlAddressCapabilities` interface extends the core `AddressCapabilities` interface. This interface provides methods to reflect the capabilities of the methods on the `CallControlAddress` interface.

The `Provider.getAddressCapabilities()` method returns the static Address capabilities, and the `Address.getCapabilities()` method returns the dynamic Address capabilities. The object returned from each of these methods can be queried with the `instanceof` operator to check if it supports this interface. This same interface is used to reflect both static and dynamic Address capabilities.

See Also:

Provider, Address, AddressCapabilities

Method Index

- **canCancelForwarding**()

  Returns true if the application can cancel the forwarding on this Address, false otherwise.
- **canGetDoNotDisturb**()

  Returns true if the application can obtain the do not disturb state, false otherwise.
- **canGetForwarding**()

  Returns true if the application can obtain the current forwarding status on this Address, false otherwise.
- **canGetMessageWaiting**()

  Returns true if the application can obtain the message waiting state, false otherwise.
- **canSetDoNotDisturb**()

  Returns true if the application can set the do not disturb state, false otherwise.
- **canSetForwarding**()

  Returns true if the application can set the forwarding on this Address, false otherwise.
- **canSetMessageWaiting**()

  Returns true if the application can set the message waiting state, false otherwise.

Methods

canSetForwarding

```
 public abstract boolean canSetForwarding()
```

Returns true if the application can set the forwarding on this Address, false otherwise.

Returns:

True if the application can set the forwarding on this Address, false otherwise.

canGetForwarding

```
 public abstract boolean canGetForwarding()
```

Returns true if the application can obtain the current forwarding status on this Address, false otherwise.

Returns:

True if the application can obtain the current forwarding status on this Address, false otherwise.

canCancelForwarding

```
 public abstract boolean canCancelForwarding()
```

Returns true if the application can cancel the forwarding on this Address, false otherwise.

Returns:

True if the application can cancel the forwarding on this Address, false otherwise.

canGetDoNotDisturb

```
public abstract boolean canGetDoNotDisturb()
```

Returns true if the application can obtain the do not disturb state, false otherwise.

Returns:

True if the application can obtain the do not disturb state, false otherwise.

canSetDoNotDisturb

```
public abstract boolean canSetDoNotDisturb()
```

Returns true if the application can set the do not disturb state, false otherwise.

Returns:

True if the application can set the do not disturb state, false otherwise.

canGetMessageWaiting

```
public abstract boolean canGetMessageWaiting()
```

Returns true if the application can obtain the message waiting state, false otherwise.

Returns:

True if the application can obtain the message waiting state, false otherwise.

canSetMessageWaiting

```
public abstract boolean canSetMessageWaiting()
```

Returns true if the application can set the message waiting state, false otherwise.

Returns:

True if the application can set the message waiting state, false otherwise.

All Packages   This Package   Previous   Next

Interface javax.telephony.callcontrol.capabilities.CallControlCallCapabilities

public interface **CallControlCallCapabilities**

extends CallCapabilities

The `CallControlCallCapabilities` interface extends the core `CallCapabilities` interface. This interface provides methods to reflect the capabilities of the methods on the `CallControlCall` interface.

The `Provider.getCallCapabilities()` method returns the static Call capabilities, and the `Call.getCapabilities()` method returns the dynamic Call capabilities. The object returned from each of these methods can be queried with the `instanceof` operator to check if it supports this interface. This same interface is used to reflect both static and dynamic Call capabilities.

See Also:

Provider, Call, CallCapabilities

Method Index

- **canAddParty**()

  Returns true if the application can invoke the add party feature, false otherwise.
- **canConference**()

  Returns true if the application can invoke the conference feature, false otherwise.
- **canConsult**()

  Returns true if the application can invoke the consult feature, false otherwise. **Deprecated.**
- **canConsult**(TerminalConnection)

  Returns true if the application can invoke the overloaded consult feature which takes a TerminalConnection as an argument, false otherwise.
- **canConsult**(TerminalConnection, String)

  Returns true if the application can invoke the overloaded consult feature which takes a TerminalConnection and string as arguments, false otherwise.
- **canDrop**()

  Returns true if the application can invoke the drop feature, false otherwise.
- **canOffHook**()
- **canSetConferenceController**()
- **canSetConferenceEnable**()

  Returns true if the application can invoke the set conferencing enabling feature, false otherwise.
- **canSetTransferController**()
- **canSetTransferEnable**()

  Returns true if the application can invoke the set transferring enabling feature, false otherwise.
- **canTransfer**()

  Returns true if the application can invoke the transfer feature, false otherwise. **Deprecated.**
- **canTransfer**(Call)

  Returns true if the application can invoke the overloaded transfer feature which takes a Call as an argument, false otherwise.
- **canTransfer**(String)

  Returns true if the application can invoke the overloaded transfer feature which takes a destination string as an argument, false otherwise.

Methods

canDrop

```
public abstract boolean canDrop()
```

Returns true if the application can invoke the drop feature, false otherwise.

Returns:

True if the application can invoke the drop feature, false otherwise.

canOffHook

```
public abstract boolean canOffHook()
```

canSetConferenceController

```
public abstract boolean canSetConferenceController()
```

canSetTransferController

```
public abstract boolean canSetTransferController()
```

canSetTransferEnable

```
public abstract boolean canSetTransferEnable()
```

Returns true if the application can invoke the set transferring enabling feature, false otherwise. The value returned by this method is independent of the ability of the application to invoke the transfer feature.

Applications are not required to inform the implementation of the purpose of the consultation call and may rely upon the default value returned by the `CallControlCall.getTransferEnable()` method.

Returns:

True if the application can invoke the set transferring enabling feature, false otherwise.

canSetConferenceEnable

```
public abstract boolean canSetConferenceEnable()
```

Returns true if the application can invoke the set conferencing enabling feature, false otherwise. The value returned by this method is independent of the ability of the application to invoke the conference feature.

Applications are not required to inform the implementation of the purpose of the consultation call and may rely upon the default value returned by the `CallControlCall.getConferenceEnable()` method.

Returns:

True if the application can invoke the set conferencing enabling feature, false otherwise.

canTransfer

```
public abstract boolean canTransfer()
```

Note: canTransfer() is deprecated. *Since JTAPI v1.2. The default behavior of this method in JTAPI v1.2 and later should invoke the canTransfer(Call) method.*

Returns true if the application can invoke the transfer feature, false otherwise.

Note: This method has been replaced in JTAPI 1.2 with overloaded versions. These versions permit applications to give typed argument to obtain the capabilities for a particular overloaded version of the `CallControlCall.transfer()` method.

Returns:

True if the application can invoke the transfer feature, false otherwise.

canTransfer

```
public abstract boolean canTransfer(Call call)
```

Returns true if the application can invoke the overloaded transfer feature which takes a Call as an argument, false otherwise.

The argument provided is for typing purposes only. The particular instance of the object given is ignored and not used to determine the capability outcome is any way.

Parameters:

call - This argument is used for typing information to determine the overloaded version of the transfer() method.

Returns:

True if the application can invoke the transfer feature which takes a Call as an argument, false otherwise.

canTransfer

```
public abstract boolean canTransfer(String destination)
```

Returns true if the application can invoke the overloaded transfer feature which takes a destination string as an argument, false otherwise.

The argument provided is for typing purposes only. The particular instance of the object given is ignored and not used to determine the capability outcome is any way.

Parameters:

destination - This argument is used for typing information to determine the overloaded version of the transfer() method.

Returns:

True if the application can invoke the transfer feature which takes a destination string as an argument, false otherwise.

canConference

```
public abstract boolean canConference()
```

Returns true if the application can invoke the conference feature, false otherwise.

Returns:

True if the application can invoke the conference feature, false otherwise.

canAddParty

```
public abstract boolean canAddParty()
```

Returns true if the application can invoke the add party feature, false otherwise.

Returns:

True if the application can invoke the add party feature, false otherwise.

canConsult

```
public abstract boolean canConsult()
```

Note: canConsult() is deprecated. *Since JTAPI v1.2. The default behavior of this method in JTAPI v1.2 and later should invoke the canConsult(TerminalConnection, String) method.*

Returns true if the application can invoke the consult feature, false otherwise.

Note: This method has been replaced in JTAPI 1.2 with overloaded versions. These versions permit applications to give typed argument to obtain the capabilities for a particular overloaded version of the `CallControlCall.consult()` method.

Returns:

True if the application can invoke the consult feature, false otherwise.

canConsult

```
public abstract boolean canConsult(TerminalConnection tc,
                                   String destination)
```

Returns true if the application can invoke the overloaded consult feature which takes a TerminalConnection and string as arguments, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

tc - This argument is used for typing information to determine the overloaded version of the consult() method.

destination - This argument is used for typing information to destination the overloaded version of the consult() method.

Returns:

True if the application can invoke the consult feature which takes a TerminalConnection and a string as arguments.

canConsult

```
public abstract boolean canConsult(TerminalConnection tc)
```

Returns true if the application can invoke the overloaded consult feature which takes a TerminalConnection as an argument, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

tc - This argument is used for typing information to determine the overloaded version of the consult() method.

Returns:

True if the application can invoke the consult feature which takes a TerminalConnection as an argument.

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.capabilities.CallControlConnectionCapabilities

public interface **CallControlConnectionCapabilities**
extends ConnectionCapabilities

The `CallControlConnectionCapabilities` interface extends the core `ConnectionCapabilities` interface. This interface provides methods to reflect the capabilities of the methods on the interface.

The `Provider.getConnectionCapabilities()` method returns the static Connection capabilities, and the `Connection.getCapabilities()` method returns the dynamic Connection capabilities. The object returned from each of these methods can be queried with the `instanceof` operator to check if it supports this interface. This same interface is used to reflect both static and dynamic Connection capabilities.

See Also:

Provider, Connection, ConnectionCapabilities

Method Index

canAccept()

Returns true if the application can invoke the accept feature, false otherwise.

canAddToAddress()

Returns true if the application can invoke the add to address feature, false otherwise.

canPark()

Returns true if the application can invoke the park feature, false otherwise.

canRedirect()

Returns true if the application can invoke the redirect feature, false otherwise.

canReject()

Returns true if the application can invoke the reject feature, false otherwise.

Methods

canRedirect

```
public abstract boolean canRedirect()
```

Returns true if the application can invoke the redirect feature, false otherwise.

Returns:

True if the application can invoke the redirect feature, false otherwise.

canAddToAddress

```
public abstract boolean canAddToAddress()
```

Returns true if the application can invoke the add to address feature, false otherwise.

Returns:

True if the application can invoke the add to address feature, false otherwise.

canAccept

```
public abstract boolean canAccept()
```

Returns true if the application can invoke the accept feature, false otherwise.

Returns:

True if the application can invoke the accept feature, false otherwise.

canReject

```
public abstract boolean canReject()
```

Returns true if the application can invoke the reject feature, false otherwise.

Returns:

True if the application can invoke the reject feature, false otherwise.

canPark

```
 public abstract boolean canPark()
```

Returns true if the application can invoke the park feature, false otherwise.

Returns:

True if the application can invoke the park feature, false otherwise.

All Packages   This Package Previous Next

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.capabilities.CallControlTerminalCapabilities

public interface **CallControlTerminalCapabilities**

extends TerminalCapabilities

The `CallControlTerminalCapabilities` interface extends the core `TerminalCapabilities` interface. This interface provides methods to reflect the capabilities of the methods on the `CallControlTerminal` interface.

The `Provider.getTerminalCapabilities()` method returns the static Terminal capabilities, and the `Terminal.getCapabilities()` method returns the dynamic Terminal capabilities. The object returned from each of these methods can be queried with the `instanceof` operator to check if it supports this interface. This same interface is used to reflect both static and dynamic Terminal capabilities.

See Also:

Provider, Terminal, TerminalCapabilities

Method Index

- **canGetDoNotDisturb**()

  Returns true if the application can obtain the do not disturb state, false otherwise.
- **canPickup**()

  Returns true if the application can invoke the pickup feature, false otherwise. **Deprecated.**
- **canPickup**(Address, Address)

  Returns true if the application can invoke the overloaded pickup feature which takes two Addresses as arguments, false otherwise.
- **canPickup**(Connection, Address)

  Returns true if the application can invoke the overloaded pickup feature which takes a Connection and an Address as arguments, false otherwise.
- **canPickup**(TerminalConnection, Address)

  Returns true if the application can invoke the overloaded pickup feature which takes a TerminalConnection and an Address as arguments, false otherwise.
- **canPickupFromGroup**()

  Returns true if the application can invoke the pickup from group feature, false otherwise. **Deprecated.**
- **canPickupFromGroup**(Address)

  Returns true if the application can invoke the pickup from group feature which takes an Address as an argument, false otherwise.
- **canPickupFromGroup**(String, Address)

  Returns true if the application can invoke the pickup from group feature which takes a string pickup group code and an Address as arguments, false otherwise.
- **canSetDoNotDisturb**()

  Returns true if the application can set the do not disturb state, false otherwise.

Methods

canGetDoNotDisturb

```
public abstract boolean canGetDoNotDisturb()
```

Returns true if the application can obtain the do not disturb state, false otherwise.

Returns:

True if the application can obtain the do not disturb state, false otherwise.

canSetDoNotDisturb

```
public abstract boolean canSetDoNotDisturb()
```

Returns true if the application can set the do not disturb state, false otherwise.

Returns:

True if the application can set the do not disturb state, false otherwise.

canPickup

```
public abstract boolean canPickup()
```

Note: canPickup() is deprecated. *Since JTAPI v1.2. The default behavior of this method in JTAPI v1.2 and later should invoke the canPickup(Connection, Address) method.*

Returns true if the application can invoke the pickup feature, false otherwise.

Note: This method has been replaced in JTAPI 1.2 with overloaded versions. These versions permit applications to give typed argument to obtain the capabilities for a particular overloaded version of the `CallControlTerminal.pickup()` method.

Returns:

True if the application can invoke the pickup feature, false otherwise.

canPickup

```
public abstract boolean canPickup(Connection connection,
                                  Address address)
```

Returns true if the application can invoke the overloaded pickup feature which takes a Connection and an Address as arguments, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

connection - This argument is used for typing information to determine the overloaded version of the pickup() method.

address - This argument is used for typing information to determine the overloaded version of the pickup() method.

Returns:

True if the application can invoke the pickup feature which takes a Connection and an Address as arguments, false otherwise.

canPickup

```
public abstract boolean canPickup(TerminalConnection tc,
                                  Address address)
```

Returns true if the application can invoke the overloaded pickup feature which takes a TerminalConnection and an Address as arguments, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

tc - This argument is used for typing information to determine the overloaded version of the pickup() method.

address - This argument is used for typing information to determine the overloaded version of the pickup() method.

Returns:

True if the application can invoke the pickup feature which takes a TerminalConnection and an Address as arguments, false otherwise.

canPickup

```
public abstract boolean canPickup(Address address1,
                                  Address address2)
```

Returns true if the application can invoke the overloaded pickup feature which takes two Addresses as arguments, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

address1 - This argument is used for typing information to determine the overloaded version of the pickup() method.

address2 - This argument is used for typing information to determine the overloaded version of the pickup() method.

Returns:

True if the application can invoke the pickup feature which takes two Addresses as arguments, false otherwise.

canPickupFromGroup

```
public abstract boolean canPickupFromGroup()
```

Note: canPickupFromGroup() is deprecated. *Since JTAPI v1.2. The default behavior of this method in JTAPI v1.2 and later should invoke the canPickupFromGroup(String, Address) method.*

Returns true if the application can invoke the pickup from group feature, false otherwise.

Note: This method has been replaced in JTAPI 1.2 with overloaded versions. These versions permit applications to give typed argument to obtain the capabilities for a particular overloaded version of the `CallControlTerminal.pickupFromGroup()` method.

Returns:

True if the application can invoke the pickup from group feature, false otherwise.

canPickupFromGroup

```
public abstract boolean canPickupFromGroup(String group,
                                           Address address)
```

Returns true if the application can invoke the pickup from group feature which takes a string pickup group code and an Address as arguments, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

group - This argument is used for typing information to determine the overloaded version of the pickupFromGroup() method.

address - This argument is used for typing information to determine the overloaded version of the pickupFromGroup() method.

Returns:

True if the application can invoke the pickup from group feature which takes a string pickup group code and an Address as arguments, false otherwise.

canPickupFromGroup

```
public abstract boolean canPickupFromGroup(Address address)
```

Returns true if the application can invoke the pickup from group feature which takes an Address as an argument, false otherwise.

The arguments provided are for typing purposes only. The particular instances of the objects given are ignored and not used to determine the capability outcome is any way.

Parameters:

address - This argument is used for typing information to determine the overloaded version of the pickupFromGroup() method.

Returns:

True if the application can invoke the pickup from group feature which takes an Address as an argument, false otherwise.

All Packages   This Package   Previous   Next

Interface javax.telephony.callcontrol.capabilities.CallControlTerminalConnectionCapabilities

public interface **CallControlTerminalConnectionCapabilities**

extends TerminalConnectionCapabilities

The `CallControlTerminalConnectionCapabilities` interface extends the core `TerminalConnectionCapabilities` interface. This interface provides methods to reflect the capabilities of the methods on the interface.

The `Provider.getTerminalConnectionCapabilities()` method returns the static TerminalConnection capabilities, and the `TerminalConnection.getCapabilities()` method returns the dynamic TerminalConnection capabilities. The object returned from each of these methods can be queried with the `instanceof` operator to check if it supports this interface. This same interface is used to reflect both static and dynamic TerminalConnection capabilities.

See Also:

Provider, TerminalConnection, TerminalConnectionCapabilities

Method Index

- **canHold**()

  Returns true if the application can invoke the hold feature, false otherwise.
- **canJoin**()

  Returns true if the application can invoke the join feature, false otherwise.
- **canLeave**()

  Returns true if the application can invoke the leave feature, false otherwise.
- **canUnhold**()

  Returns true if the application can invoke the unhold feature, false otherwise.

Methods

canHold

```
public abstract boolean canHold()
```

Returns true if the application can invoke the hold feature, false otherwise.

Returns:

True if the application can invoke the hold feature, false otherwise.

canUnhold

```
public abstract boolean canUnhold()
```

Returns true if the application can invoke the unhold feature, false otherwise.

Returns:

True if the application can invoke the unhold feature, false otherwise.

canJoin

```
public abstract boolean canJoin()
```

Returns true if the application can invoke the join feature, false otherwise.

Returns:

True if the application can invoke the join feature, false otherwise.

canLeave

```
public abstract boolean canLeave()
```

Returns true if the application can invoke the leave feature, false otherwise.

Returns:

True if the application can invoke the leave feature, false otherwise.

All Packages

package javax.telephony.callcontrol.events

Interface Index

- CallCtlAddrDoNotDisturbEv
- CallCtlAddrEv
- CallCtlAddrForwardEv
- CallCtlAddrMessageWaitingEv
- CallCtlCallEv
- CallCtlConnAlertingEv
- CallCtlConnDialingEv
- CallCtlConnDisconnectedEv
- CallCtlConnEstablishedEv
- CallCtlConnEv
- CallCtlConnFailedEv
- CallCtlConnInitiatedEv
- CallCtlConnNetworkAlertingEv
- CallCtlConnNetworkReachedEv
- CallCtlConnOfferedEv
- CallCtlConnQueuedEv
- CallCtlConnUnknownEv
- CallCtlEv
- CallCtlTermConnBridgedEv
- CallCtlTermConnDroppedEv
- CallCtlTermConnEv
- CallCtlTermConnHeldEv
- CallCtlTermConnInUseEv
- CallCtlTermConnRingingEv
- CallCtlTermConnTalkingEv
- CallCtlTermConnUnknownEv
- CallCtlTermDoNotDisturbEv
- CallCtlTermEv

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlAddrDoNotDisturbEv

public interface **CallCtlAddrDoNotDisturbEv**

extends CallCtlAddrEv

The `CallCtlAddrDoNotDisturbEv` interface indicates the state of the do not disturb feature has changed for the Address. This interface extends the `CallCtlAddrEv` interface and is reported via the `AddressObserver.addressChangedEvent()` method. The observer object must also implement the `CallControlAddressObserver` interface to signal it is interested in call control package events.

This interface supports a single method which returns the current state of the do not disturb feature.

See Also:

Address, AddressObserver, CallControlAddress, CallControlAddressObserver, CallCtlAddrEv

Variable Index

● **ID**

Event id

Method Index

● **getDoNotDisturbState**()

Returns true if the do not disturb feature is activated, false otherwise.

Variables

● **ID**

```
 public static final int ID
```

Event id

Methods

● **getDoNotDisturbState**

```
 public abstract boolean getDoNotDisturbState()
```

Returns true if the do not disturb feature is activated, false otherwise.

Returns:

True if the do not disturb feature is activated, false otherwise.

Interface javax.telephony.callcontrol.events.CallCtlAddrEv

public interface **CallCtlAddrEv**

extends CallCtlEv, AddrEv

The `CallCtlAddrEv` interface is the base interface for all call control package Address-related events. All events which pertain to the `CallControlAddress` interface must extend this interface. Events which extend this interface are reported via the `AddressObserver.addressChangedEvent()` method. The observer object must also implement the `CallControlAddressObserver` interface to signal it is interested in call control package events. This interface extends both the `CallCtlEv` and `AddrEv` interfaces.

The events defined in the call control package for the Address are the `CallCtlAddrDoNotDisturbEv`, `CallCtlAddrForwardEv`, and `CallCtlAddrMessageWaitingEv` events.

This interface supports no additional methods.

See Also:

Address, AddressObserver, AddrEv, CallControlAddress, CallControlAddressObserver, CallCtlAddrDoNotDisturbEv, CallCtlAddrForwardEv, CallCtlAddrMessageWaitingEv, CallCtlEv

Interface javax.telephony.callcontrol.events.CallCtlAddrForwardEv

public interface **CallCtlAddrForwardEv**

extends CallCtlAddrEv

The `CallCtlAddrForwardEv` interface indicates the state of the forward feature has changed for the Address. This interface extends the `CallCtlAddrEv` interface and is reported via the `AddressObserver.addressChangedEvent()` method. The observer object must also implement the `CallControlAddressObserver` interface to signal it is interested in call control package events.

This interface supports a single method which returns the current state of the forwarding feature.

See Also:

Address, AddressObserver, CallControlAddress, CallControlAddressObserver, CallCtlAddrEv

Variable Index

● **ID**

Event id

Method Index

● **getForwarding**()

Returns the current forwarding on the Address.

Variables

● **ID**

```
public static final int ID
```

Event id

Methods

● **getForwarding**

```
public abstract CallControlForwarding[] getForwarding()
```

Returns the current forwarding on the Address.

Returns:

An array of CallControlForwarding objects.

All Packages   This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlAddrMessageWaitingEv

public interface **CallCtlAddrMessageWaitingEv**

extends CallCtlAddrEv

The `CallCtlAddrMessageWaitingEv` interface indicates the state of the message waiting feature has changed for the Address. This interface extends the `CallCtlAddrEv` interface and is reported via the `AddressObserver.addressChangedEvent()` method. The observer object must also implement the `CallControlAddressObserver` interface to signal it is interested in call control package events.

This interface supports a single method which returns the current state of the message waiting feature.

See Also:

Address, AddressObserver, CallControlAddress, CallControlAddressObserver, CallCtlAddrEv

Variable Index

● **ID**

Event id

Method Index

● **getMessageWaitingState**()

Returns the current message waiting state of the Address.

Variables

● **ID**

```
public static final int ID
```

Event id

Methods

● **getMessageWaitingState**

```
public abstract boolean getMessageWaitingState()
```

Returns the current message waiting state of the Address.

Returns:

The message waiting state of the Address.

Interface javax.telephony.callcontrol.events.CallCtlCallEv

public interface **CallCtlCallEv**

extends CallCtlEv, CallEv

The `CallCtlCallEv` interface is the base interface for all call control package Call-related events. All events which pertain to the `CallControlCall` interface must extend this interface. Events which extend this interface are reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events. This interface extend both the `CallCtlEv` and core `CallEv` interfaces.

The `CallCtlConnEv` and `CallCtlTermConnEv` events extend this interface. This reflects the fact that all events pertaining to the `CallControlConnection` interface and the `CallControlTerminalConnection` interface are reported via the `CallControlCallObserver` interface.

Additional Call Information

This interface supports methods which return additional information regarding the telephone call. Specifically, it returns the *calling address*, *calling terminal*, *called address*, and *last redirected address* information. This information is returned by the `getCallingAddress()`, `getCallingTerminal()`, `getCalledAddress()`, and `getLastRedirectedAddress()` methods on this interface, respectively.

See Also:

Call, Address, Terminal, CallObserver, CallEv, CallControlCall, CallControlCallObserver, CallCtlEv, CallCtlConnEv, CallCtlTermConnEv

Method Index

- **getCalledAddress**()

  Returns the called Address associated with this Call.
- **getCallingAddress**()

  Returns the calling Address associated with this call.
- **getCallingTerminal**()

  Returns the calling Terminal associated with this Call.
- **getLastRedirectedAddress**()

  Returns the last redirected Address associated with this Call.

Methods

getCallingAddress

```
 public abstract Address getCallingAddress()
```

Returns the calling Address associated with this call. The calling Address is defined as the Address which placed the telephone call.

If the calling address is unknown or not yet known, this method returns null.

Returns:

The calling Address.

getCallingTerminal

```
public abstract Terminal getCallingTerminal()
```

Returns the calling Terminal associated with this Call. The calling Terminal is defined as the Terminal which placed the telephone call.

If the calling Terminal is unknown or not yet know, this method returns null.

Returns:

The calling Terminal.

getCalledAddress

```
public abstract Address getCalledAddress()
```

Returns the called Address associated with this Call. The called Address is defined as the Address to which the call has been originally placed.

If the called address is unknown or not yet known, this method returns null.

Returns:

s The called Address.

getLastRedirectedAddress

```
public abstract Address getLastRedirectedAddress()
```

Returns the last redirected Address associated with this Call. The last redirected Address is the Address at which the current telephone call was placed immediately before the current Address. This is common if a Call is forwarded to several Addresses before being answered.

If the the last redirected address is unknown or not yet known, this method returns null.

Returns:

s The last redirected Address for this telephone Call.

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlConnAlertingEv

public interface **CallCtlConnAlertingEv**

extends CallCtlConnEv

The `CallCtlConnAlertingEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.ALERTING`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnDialingEv

public interface **CallCtlConnDialingEv**

extends CallCtlConnEv

The `CallCtlConnDialingEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.DIALING`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports the `getDigits()` method which returns the digits which have already been dialed.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

ID

Event id

Method Index

getDigits()

Returns the digits that have already been dialed.

Variables

ID

```
public static final int ID
```

Event id

Methods

getDigits

```
public abstract String getDigits()
```

Returns the digits that have already been dialed.

Returns:

s The digits that have already been dialed.

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlConnDisconnectedEv

public interface **CallCtlConnDisconnectedEv**

extends CallCtlConnEv

The `CallCtlConnDisconnectedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.DISCONNECTED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnEstablishedEv

public interface **CallCtlConnEstablishedEv**

extends CallCtlConnEv

The `CallCtlConnEstablishedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.ESTABLISHED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnEv

public interface **CallCtlConnEv**

extends CallCtlCallEv, ConnEv

The `CallCtlConnEv` interface is the base interface for all call control package Connection-related events. All events which pertain to the `CallControlConnection` interface must extend this interface. Events which extend this interface are reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events. This interface extends both the `CallCtlCallEv` and core `ConnEv` interfaces.

A number of event interfaces defined in this package extend this interface. Each of these events conveys a change in the call control package state of the Connection. The event interfaces which extend this interface are: `CallCtlConnAlertingEv`, `CallCtlConnDialingEv`, `CallCtlConnDisconnectedEv`, `CallCtlConnEstablishedEv`, `CallCtlConnFailedEv`, `CallCtlConnInitiatedEv`, `CallCtlConnNetworkAlertingEv`, `CallCtlConnNetworkReachedEv`, CODE>CallCtlConnOfferedEv, `CallCtlConnQueuedEv`, and `CallCtlConnUnknownEv`

This interface supports no additional methods.

See Also:

Connection, CallObserver, ConnEv, CallControlConnection, CallControlCallObserver, CallCtlCallEv, CallCtlConnAlertingEv, CallCtlConnDialingEv, CallCtlConnDisconnectedEv, CallCtlConnEstablishedEv, CallCtlConnFailedEv, CallCtlConnInitiatedEv, CallCtlConnNetworkAlertingEv, CallCtlConnNetworkReachedEv, CallCtlConnOfferedEv, CallCtlConnQueuedEv, CallCtlConnUnknownEv

Interface javax.telephony.callcontrol.events.CallCtlConnFailedEv

public interface **CallCtlConnFailedEv**

extends CallCtlConnEv

The `CallCtlConnFailedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.FAILED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnInitiatedEv

public interface **CallCtlConnInitiatedEv**

extends CallCtlConnEv

The `CallCtlConnInitiatedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.INITIATED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnNetworkAlertingEv

public interface **CallCtlConnNetworkAlertingEv**

extends CallCtlConnEv

The `CallCtlConnNetworkAlertingEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.NETWORK_ALERTING`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnNetworkReachedEv

public interface **CallCtlConnNetworkReachedEv**

extends CallCtlConnEv

The `CallCtlConnNetworkReachedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.NETWORK_REACHED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlConnOfferedEv

public interface **CallCtlConnOfferedEv**

extends CallCtlConnEv

The `CallCtlConnOfferedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.OFFERING`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlConnQueuedEv

public interface **CallCtlConnQueuedEv**

extends CallCtlConnEv

The `CallCtlConnQueuedEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.QUEUED`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports the `getNumberInQueue()` method which returns the number of Connections being queue along with this Connection at the same Address.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

● **ID**

Event id

Method Index

● **getNumberInQueue**()

Returns the number of Connections which are queued at the Address of this Connection.

Variables

● **ID**

```
public static final int ID
```

Event id

Methods

● **getNumberInQueue**

```
public abstract int getNumberInQueue()
```

Returns the number of Connections which are queued at the Address of this Connection.

Returns:

The number of queued Connections.

Interface javax.telephony.callcontrol.events.CallCtlConnUnknownEv

public interface **CallCtlConnUnknownEv**

extends CallCtlConnEv

The `CallCtlConnUnknownEv` interface indicates that the call control package state of the Connection is now `CallControlConnection.UNKNOWN`. This method `CallControlConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, Connection, CallControlConnection, CallControlCallObserver, CallCtlConnEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlEv

public interface **CallCtlEv**

extends Ev

The `CallCtlEv` is the base event for all events in the call control package. Each event in this package must extend this interface. This interface extends the core `Ev` interface.

This interface contains the `getCallControlCause()` method which returns a call control package specific cause for the event. Cause codes pertaining to this package are defined in this interface as well.

In the call control package, this interface is extended by the following interfaces: `CallCtlCallEv`, `CallCtlAddrEv`, and `CallCtlTermEv`.

See Also:

Ev, CallCtlCallEv, CallCtlAddrEv, CallCtlTermEv

Variable Index

- **CAUSE_ALTERNATE**

  Cause code indicating that the call was put on hold and another retrieved in an atomic operation, typically on single line telephones.
- **CAUSE_BUSY**

  Cause code indicating that the call encountered a busy endpoint.
- **CAUSE_CALL_BACK**

  Cause code indicating the event is related to the callback feature.
- **CAUSE_CALL_NOT_ANSWERED**

  Cause code indicating that the call was not answered before a timer elapsed.
- **CAUSE_CALL_PICKUP**

  Cause code indicating that the call was redirected by the call pickup feature.
- **CAUSE_CONFERENCE**

  Cause code indicating the event is related to the conference feature.
- **CAUSE_DO_NOT_DISTURB**

  Cause code indicating the event is related to the do not disturb feature.
- **CAUSE_PARK**

  Cause code indicating the event is related to the park feature.
- **CAUSE_REDIRECTED**

  Cause code indicating the event is related to the redirect feature.
- **CAUSE_REORDER_TONE**

  Cause code indicating that the call encountered a reorder tone.
- **CAUSE_TRANSFER**

  Cause code indicating the event is related to the transfer feature.
- **CAUSE_TRUNKS_BUSY**

  Cause code indicating that the call encountered the "all trunks busy" condition.

CAUSE_UNHOLD

Cause code indicating the event is related to the unhold feature.

Method Index

getCallControlCause()

Returns the call control package cause associated with the event.

Variables

CAUSE_ALTERNATE

```
public static final int CAUSE_ALTERNATE
```

Cause code indicating that the call was put on hold and another retrieved in an atomic operation, typically on single line telephones.

CAUSE_BUSY

```
public static final int CAUSE_BUSY
```

Cause code indicating that the call encountered a busy endpoint.

CAUSE_CALL_BACK

```
public static final int CAUSE_CALL_BACK
```

Cause code indicating the event is related to the callback feature.

CAUSE_CALL_NOT_ANSWERED

```
public static final int CAUSE_CALL_NOT_ANSWERED
```

Cause code indicating that the call was not answered before a timer elapsed.

CAUSE_CALL_PICKUP

```
public static final int CAUSE_CALL_PICKUP
```

Cause code indicating that the call was redirected by the call pickup feature.

CAUSE_CONFERENCE

```
public static final int CAUSE_CONFERENCE
```

Cause code indicating the event is related to the conference feature.

CAUSE_DO_NOT_DISTURB

```
public static final int CAUSE_DO_NOT_DISTURB
```

Cause code indicating the event is related to the do not disturb feature.

CAUSE_PARK

```
public static final int CAUSE_PARK
```

Cause code indicating the event is related to the park feature.

CAUSE_REDIRECTED

```
public static final int CAUSE_REDIRECTED
```

Cause code indicating the event is related to the redirect feature.

CAUSE_REORDER_TONE

```
public static final int CAUSE_REORDER_TONE
```

Cause code indicating that the call encountered a reorder tone.

CAUSE_TRANSFER

```
public static final int CAUSE_TRANSFER
```

Cause code indicating the event is related to the transfer feature.

CAUSE_TRUNKS_BUSY

```
public static final int CAUSE_TRUNKS_BUSY
```

Cause code indicating that the call encountered the "all trunks busy" condition.

CAUSE_UNHOLD

```
public static final int CAUSE_UNHOLD
```

Cause code indicating the event is related to the unhold feature.

Methods

getCallControlCause

```
public abstract int getCallControlCause()
```

Returns the call control package cause associated with the event. The cause values are integer constants defined in this interface. This method may also returns the `Ev.CAUSE_NORMAL` and the `Ev.CAUSE_UNKNOWN` values as defined in the core package.

Returns:

s The call control package cause of the event.

All Packages   This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlTermConnBridgedEv

public interface **CallCtlTermConnBridgedEv**

extends CallCtlTermConnEv

The `CallCtlTermConnBridgedEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.BRIDGED`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlTermConnDroppedEv

public interface **CallCtlTermConnDroppedEv**

extends CallCtlTermConnEv

The `CallCtlTermConnDroppedEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.DROPPED`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlTermConnEv

public interface **CallCtlTermConnEv**

extends CallCtlCallEv, TermConnEv

The `CallCtlTermConnEv` interface is the base interface for all call control package TerminalConnection-related events. All events which pertain to the `CallControlTerminalConnection` interface must extend this interface. Events which extend this interface are reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events. This interface extends both the `CallCtlCallEv` and core `TermConnEv` interfaces.

A number of event interfaces defined in this package extend this interface. Each of these events conveys a change in the call control package state of the TerminalConnection. The event interfaces which extend this interface are: `CallCtlTermConnBridgedEv`, `CallCtlTermConnDroppedEv`, `CallCtlTermConnHeldEv`, `CallCtlTermConnInUseEv`, `CallCtlTermConnRingingEv`, `CallCtlTermConnTalkingEv`, and `CallCtlTermConnUnknownEv`

This interface supports no additional methods.

See Also:

TerminalConnection, CallObserver, TermConnEv, CallControlTerminalConnection, CallControlCallObserver, CallCtlCallEv, CallCtlTermConnBridgedEv, CallCtlTermConnDroppedEv, CallCtlTermConnHeldEv, CallCtlTermConnInUseEv, CallCtlTermConnRingingEv, CallCtlTermConnTalkingEv, CallCtlTermConnUnknownEv

Interface javax.telephony.callcontrol.events.CallCtlTermConnHeldEv

public interface **CallCtlTermConnHeldEv**

extends CallCtlTermConnEv

The `CallCtlTermConnHeldEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.HELD`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlTermConnInUseEv

public interface **CallCtlTermConnInUseEv**

extends CallCtlTermConnEv

The `CallCtlTermConnInUseEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.INUSE`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.callcontrol.events.CallCtlTermConnRingingEv

public interface **CallCtlTermConnRingingEv**

extends CallCtlTermConnEv

The `CallCtlTermConnRingingEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.RINGING`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

All Packages   This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlTermConnTalkingEv

public interface **CallCtlTermConnTalkingEv**

extends CallCtlTermConnEv

The `CallCtlTermConnTalkingEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.TALKING`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

All Packages   This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlTermConnUnknownEv

public interface **CallCtlTermConnUnknownEv**

extends CallCtlTermConnEv

The `CallCtlTermConnUnknownEv` interface indicates that the call control package state of the TerminalConnection is now `CallControlTerminalConnection.UNKNOWN`. This method `CallControlTerminalConnection.getCallControlState()` returns the current state. This interface extends the `CallCtlTermConnEv` interface and is reported via the `CallObserver.callChangedEvent()` method. The observer object must also implement the `CallControlCallObserver` interface to signal it is interested in call control package events.

This interface supports no additional methods.

See Also:

CallObserver, TerminalConnection, CallControlTerminalConnection, CallControlCallObserver, CallCtlTermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlTermDoNotDisturbEv

public interface **CallCtlTermDoNotDisturbEv**

extends CallCtlTermEv

The `CallCtlTermDoNotDisturbEv` interface indicates the state of the do not disturb feature has changed for the Terminal. This interface extends the `CallCtlTermEv` interface and is reported via the `TerminalObserver.terminalChangedEvent()` method. The observer object must also implement the `CallControlTerminalObserver` interface to signal it is interested in call control package events.

This interface supports a single method which returns the current state of the do not disturb feature.

See Also:

Terminal, TerminalObserver, CallControlTerminal, CallControlTerminalObserver, CallCtlTermEv

Variable Index

ID

Event id

Method Index

getDoNotDisturbState()

Returns true if the do not disturb feature is activated, false otherwise.

Variables

ID

```
public static final int ID
```

Event id

Methods

getDoNotDisturbState

```
public abstract boolean getDoNotDisturbState()
```

Returns true if the do not disturb feature is activated, false otherwise.

Returns:

True if the do not disturb feature is activated, false otherwise.

All Packages This Package Previous Next

Interface javax.telephony.callcontrol.events.CallCtlTermEv

public interface **CallCtlTermEv**

extends CallCtlEv, TermEv

The `CallCtlTermEv` interface is the base interface for all call control package Terminal-related events. All events which pertain to the `CallControlTerminal` interface must extend this interface. Events which extend this interface are reported via the `TerminalObserver.terminalChangedEvent()` method. The observer object must also implement the `CallControlTerminalObserver` interface to signal it is interested in call control package events. This interface extends both the `CallCtlEv` and `TermEv` interfaces.

The only event defined in the call control package for the Terminal is the `CallCtlTermDoNotDisturbEv`.

This interface supports no additional methods.

See Also:

Terminal, TerminalObserver, TermEv, CallControlTerminal, CallControlTerminalObserver, CallCtlTermDoNotDisturbEv, CallCtlEv

All Packages This Package Previous Next

package javax.telephony.capabilities

Interface Index

- AddressCapabilities
- CallCapabilities
- ConnectionCapabilities
- ProviderCapabilities
- TerminalCapabilities
- TerminalConnectionCapabilities

Interface javax.telephony.capabilities.AddressCapabilities

public interface **AddressCapabilities**

The `AddressCapabilities` interface represents the initial capabilities interface for the Address. This interface supports basic queries for the core package.

Applications obtain the static Address capabilities via the `Provider.getAddressCapabilities()` method, and the dynamic capabilities via the Address.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core Address interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider, Address

Method Index

- **isObservable**()
  Returns true if this Address can be observed, false otherwise.

Methods

isObservable

```
public abstract boolean isObservable()
```

Returns true if this Address can be observed, false otherwise.

Returns:

True if this Address can be observed, false otherwise.

Interface javax.telephony.capabilities.CallCapabilities

public interface **CallCapabilities**

The `CallCapabilities` interface represents the initial capabilities interface for the Call. This interface supports basic queries for the core package.

Applications obtain the static Call capabilities via the `Provider.getCallCapabilities()` method, and the dynamic capabilities via the Call.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core Call interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider, Call

Method Index

- **canConnect**()

  Returns true if the application can invoke Call.connect(), false otherwise.
- **isObservable**()

  Returns true if this Call can be observed, false otherwise.

Methods

canConnect

```
 public abstract boolean canConnect()
```

Returns true if the application can invoke Call.connect(), false otherwise.

Returns:

True if the application can perform a connect, false otherwise.

isObservable

```
 public abstract boolean isObservable()
```

Returns true if this Call can be observed, false otherwise.

Returns:

True if this Call can be observed, false otherwise.

Interface javax.telephony.capabilities.ConnectionCapabilities

public interface **ConnectionCapabilities**

The `ConnectionCapabilities` interface represents the initial capabilities interface for the Connection. This interface supports basic queries for the core package.

Applications obtain the static Connection capabilities via the `Provider.getConnectionCapabilities()` method, and the dynamic capabilities via the Connection.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core Connection interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider, Connection

Method Index

● **canDisconnect**()

Returns true if the application can invoke Connection.disconnect()perform a disconnect(), false otherwise.

Methods

● canDisconnect

```
public abstract boolean canDisconnect()
```

Returns true if the application can invoke Connection.disconnect()perform a disconnect(), false otherwise.

Returns:

True if the application can disconnect, false otherwise.

Interface javax.telephony.capabilities.ProviderCapabilities

public interface **ProviderCapabilities**

The `ProviderCapabilities` interface represents the initial capabilities interface for the Provider. This interface supports basic queries for the core package.

Applications obtain the static Provider capabilities via the `Provider.getProviderCapabilities()` method, and the dynamic capabilities via the Provider.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core Provider interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider

Method Index

- **isObservable**()

  Returns true if this Provider can be observed, false otherwise.

Methods

isObservable

```
public abstract boolean isObservable()
```

Returns true if this Provider can be observed, false otherwise.

Returns:

True if this Provider can be observed, false otherwise.

Interface javax.telephony.capabilities.TerminalCapabilities

public interface **TerminalCapabilities**

The `TerminalCapabilities` interface represents the initial capabilities interface for the Terminal. This interface supports basic queries for the core package.

Applications obtain the static Terminal capabilities via the `Provider.getTerminalCapabilities()` method, and the dynamic capabilities via the Terminal.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core Terminal interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider, Terminal

Method Index

- **isObservable**()

  Returns true if this Terminal is observable, false otherwise.

Methods

isObservable

```
public abstract boolean isObservable()
```

Returns true if this Terminal is observable, false otherwise.

Returns:

True if this Terminal is observable, false otherwise.

Interface javax.telephony.capabilities.TerminalConnectionCapabilities

public interface **TerminalConnectionCapabilities**

The `TerminalConnectionCapabilities` interface represents the initial capabilities interface for the TerminalConnection. This interface supports basic queries for the core package.

Applications obtain the static TerminalConnection capabilities via the `Provider.getTerminalConnectionCapabilities()` method, and the dynamic capabilities via the TerminalConnection.getCapabilities() method. This interface is used to represent both static and dynamic capabilities.

Any package which extends the core TerminalConnection interface should also extend this interface to provide additional capability queries for that particular package.

See Also:

Provider, TerminalConnection

Method Index

canAnswer()

Returns true if the application can invoke TerminalConnection.answer()
, false otherwise.

Methods

canAnswer

```
 public abstract boolean canAnswer()
```

Returns true if the application can invoke TerminalConnection.answer()
, false otherwise.

Returns:

True if the application can answer, false otherwise.

All Packages

package javax.telephony.events

Interface Index

- AddrEv
- AddrObservationEndedEv
- CallActiveEv
- CallEv
- CallInvalidEv
- CallObservationEndedEv
- ConnAlertingEv
- ConnConnectedEv
- ConnCreatedEv
- ConnDisconnectedEv
- ConnEv
- ConnFailedEv
- ConnInProgressEv
- ConnUnknownEv
- Ev
- ProvEv
- ProvInServiceEv
- ProvObservationEndedEv
- ProvOutOfServiceEv
- ProvShutdownEv
- TermConnActiveEv
- TermConnCreatedEv
- TermConnDroppedEv
- TermConnEv
- TermConnPassiveEv
- TermConnRingingEv
- TermConnUnknownEv
- TermEv
- TermObservationEndedEv

Interface javax.telephony.events.AddrEv

public interface **AddrEv**
extends Ev

The `AddrEv` interface is the base interface for all Address- related events. All events which pertain to the Address object must extend this interface. Events which extend this interface are reported via the `AddressObserver` interface.

The only event defined in the core package for the Address is the `AddrObservationEndedEv`.

The `AddrEv.getAddress()` method on this interface returns the Address associated with the Address event.

See Also:

AddrObservationEndedEv, Ev, AddressObserver, Address

Method Index

getAddress()

Returns the Address associated with this Address event.

Methods

getAddress

```
 public abstract Address getAddress()
```

Returns the Address associated with this Address event.

Returns:

The Address associated with this event.

Interface javax.telephony.events.AddrObservationEndedEv

public interface **AddrObservationEndedEv**

extends AddrEv

The AddrObservationEndedEv event indicates that the application will no longer receive Address events on the instance of the AddressObserver. This interface extends the AddrEv interface and is reported on the AddressObserver interface.

See Also:

AddrEv, AddressObserver

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.CallActiveEv

public interface **CallActiveEv**

extends CallEv

The `CallActiveEv` interface indicates that the state of the Call object has changed to `Call.ACTIVE`. This interface extends the `CallEv` interface and is reported via the `CallObserver` interface.

See Also:

Call, CallObserver, CallEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.events.CallEv

public interface **CallEv**

extends Ev

The `CallEv` interface is the base interface for all Call-related events. All events which pertain to the Call object must extend this interface. Events which extend this interface are reported via the `CallObserver` interface.

The core package defines events which are reported when the Call changes state. These events are: `CallActiveEv` and `CallInvalidEv`. Also, the core package defines the `CallObservationEndedEv` event which is sent when the Call becomes unobservable.

The `ConnEv` and `TermConnEv` events extend this interface. This reflects the fact that all Connection and TerminalConnection events are reported via the `CallObserver` interface.

The `CallEv.getCall()` method on this interface returns the Call associated with the Call event.

See Also:

CallActiveEv, CallInvalidEv, CallObservationEndedEv, Ev, ConnEv, TermConnEv, CallObserver, Call

Method Index

- **getCall**()

  Returns the Call object associated with this Call event.

Methods

getCall

```
public abstract Call getCall()
```

Returns the Call object associated with this Call event.

Returns:

The Call associated with this event.

Interface javax.telephony.events.CallInvalidEv

public interface **CallInvalidEv**

extends CallEv

The `CallInvalidEv` interface indicates that the state of the Call object has changed to `Call.INVALID`. This interface extends the `CallEv` interface and is reported via the `CallObserver` interface.

See Also:

Call, CallObserver, CallEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.CallObservationEndedEv

public interface **CallObservationEndedEv**

extends CallEv

The CallObservationEndedEv event indicates that the application will no longer receive Call events on the instance of the CallObserver This interface extends the CallEv interface and is reported on the CallObserver interface.

See Also:

CallEv, CallObserver

Variable Index

- **ID**

  Event id

Method Index

- **getEndedObject**()

  This method returns the object which is responsible for the observation of the call ending.

Variables

ID

```
public static final int ID
```

Event id

Methods

getEndedObject

```
public abstract Object getEndedObject()
```

This method returns the object which is responsible for the observation of the call ending. If this method returns a Call, the observation ended because either the Call could no longer be observed or the observer was removed via the Call.removeObserver() method. If this method returns either an Address or Terminal, then additional obsevers will not be added if the Call arrives at the returned Address or Terminal.

Returns:

The object responsible for the observation ending for the Call.

All Packages This Package Previous Next

Interface javax.telephony.events.ConnAlertingEv

public interface **ConnAlertingEv**

extends ConnEv

The `ConnAlertingEv` interface indicates that the state of the Connection object has changed to `Connection.ALERTING`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ConnConnectedEv

public interface **ConnConnectedEv**

extends ConnEv

The `ConnConnectedEv` interface indicates that the state of the Connection object has changed to `Connection.CONNECTED`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.events.ConnCreatedEv

public interface **ConnCreatedEv**

extends ConnEv

The `ConnUnknownEv` interface indicates that a new Connection object has been created. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ConnDisconnectedEv

public interface **ConnDisconnectedEv**

extends ConnEv

The `ConnDisconnectedEv` interface indicates that the state of the Connection object has changed to `Connection.DISCONNECTED`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ConnEv

public interface **ConnEv**

extends CallEv

The `ConnEv` interface is the base event interface for all Connection-related events. All events which pertain to the Connection object must extend this interface. This interface extends the `CallEv` interface and therefore is reported via the `CallObserver` interface.

The core package defines events which are reported when the Connection changes state. These events are: `ConnInProgressEv`, `ConnAlertingEv`, `ConnConnectedEv`, `ConnDisconnectedEv`, `ConnFailedEv`, and `ConnUnknownEv`. Also, the `ConnCreatedEv` is sent when a new Connection is created.

The `ConnEv.getConnection()` method on this interface returns the Connection associated with this Connection event.

See Also:

Connection, CallObserver, CallEv, ConnCreatedEv, ConnInProgressEv, ConnAlertingEv, ConnConnectedEv, ConnDisconnectedEv, ConnFailedEv, ConnUnknownEv

Method Index

getConnection()

Returns the Connection associated with this Connection event.

Methods

getConnection

```
 public abstract Connection getConnection()
```

Returns the Connection associated with this Connection event.

Returns:

The Connection associated with this event.

Interface javax.telephony.events.ConnFailedEv

public interface **ConnFailedEv**

extends ConnEv

The `ConnFailedEv` interface indicates that the state of the Connection object has changed to `Connection.FAILED`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ConnInProgressEv

public interface **ConnInProgressEv**

extends ConnEv

The `ConnInProgressEv` interface indicates that the state of the Connection object has changed to `Connection.IN_PROGRESS`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ConnUnknownEv

public interface **ConnUnknownEv**

extends ConnEv

The `ConnUnknownEv` interface indicates that the state of the Connection object has changed to `Connection.UNKNOWN`. This interface extends the `ConnEv` interface and is reported via the `CallObserver` interface.

See Also:

Connection, CallObserver, ConnEv

Variable Index

ID

Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.events.Ev

public interface **Ev**

Introduction

The `Ev` interface is the parent of all JTAPI event interfaces. All JTAPI event interfaces extend this interface, either directly or indirectly. Event interfaces within each JTAPI package are organized in a hierarchical fashion. The architecture of the core package event hierarchy is described later.

The JTAPI event system notifies applications when changes in various JTAPI object occur. Each individual change in an object is represented by an event sent to the appropriate observer. Because several changes may happen to an object at once, events are delivered as a *batch*. A batch of events represents a series of events and changes to the call model which happened exactly at the same time. For this reason, events are delivered to observers as arrays.

Event IDs

Each event carries a corresponding identification integer. The `Ev.getID()` method returns this identification number for each event. The actual event identification integer is defined in each of the specific event interfaces. Each event interface must carry a unique id.

Cause Codes

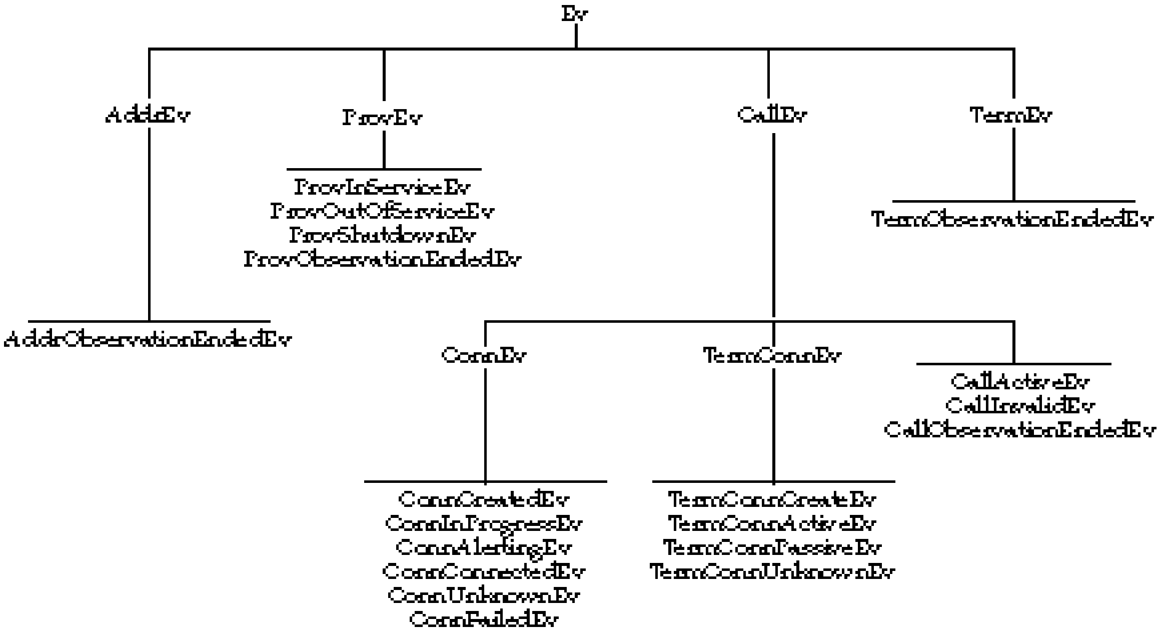
Each events carries a *cause* or a reason why the event happened. The `Ev.getCause()` method returns this cause value. The different types of cause values are also defined in this interface.

Core Package Event Hierarchy

The core package defines a hierarchy of event interfaces. The base of this hierarchy is the `Ev` interface. Directly extending this interface are those events interfaces for each object which supports an observer: `ProvEv`, `CallEv`, `AddrEv`, and `TermEv`.

Since Connection and TerminalConnection events are reported via the `CallObserver` interface, the `ConnEv` and `TermConnEv` interfaces extends the `CallEv` interface.

The following diagram illustrates the complete core package event structure.



Meta Codes

The `Ev.getMetaCode()` method returns the *meta code* for the event. Events are grouped together using meta codes to provide a higher-level description of an update to the call model. Since events represent singular changes in one particular object in the call model, it may be difficult for the application to infer a higher-level interpretation of several of these singular events. Meta codes exist on events to assist the application in this regard.

Events which belong to the same higher-level action and contain the same meta code are reported consecutively in an event batch sent to an observer. In fact, multiple meta code grouping of events may exist in a single event batch. In that case, the `Ev.isNewMetaEvent()` method is used to indicate the beginning of a new meta code event grouping. This method also indicates whether a meta code grouping exists across event batch boundaries. That is, events belonging to the same meta code grouping may be delivered in two contiguous event batches.

There are five types of meta codes which pertain to individual calls, and two which pertain to a mutli-call action, and two miscellaneous meta codes. The five meta codes which pertain to individual calls are:

Ev.META_CALL_STARTING	Indicates that a new active call has been presented to the application, either by an application creating a call and performing an action on it, or by an incoming call to an object being observed by the application.
Ev.META_CALL_PROGRESS	Indicates that the objects belonging to a call have changed state, with the exception of Connections moving to `Connection.DISCONNECTED`. For example, when a remote party answers a telephone call and the corresponding Connection moves into the `Connection.CONNECTED` state, this is the meta code associated with the resulting batch of events.
Ev.META_CALL_ADDING_PARTY	Indicates that a party has been added to the call. A "party" corresponds to a Connection being added. Note that if a TerminalConnection is added, it carries a meta code of `Ev.META_CALL_PROGRESS`.
Ev.META_CALL_REMOVING_PARTY	Indicates that a party (i.e. Connection) has been removed from the call by moving into the `Connection.DISCONNECTED` state.
Ev.META_CALL_ENDING	Indicates that an entire telephone call has ended, which implies the call has moved into the `Call.INVALID` state and all of its Connections have moved into the `Connection.DISCONNECTED` state.

The two meta codes pertaining to a mutli-call actions are as follows:

Ev.META_CALL_MERGING	Indicates that a party has moved from one call to another as part of the two calls merging. A common example is when two telephone calls are conferenced.
Ev.META_CALL_TRANSFERRING	Indicates that a party has moved from one call to another as part of one call being transferred to another. The differs from `Ev.META_CALL_MERGING` because a common party leaves both calls.

The two miscellaneous meta codes are as follows:

Ev.META_SNAPSHOT	Indicates that the sequence of events are part of a "snapshot" given to the application to bring it up-to-date with the current state of the call model.
Ev.META_UNKNOWN	Indicates that the meta code is unknown for the event.

Variable Index

- **CAUSE_CALL_CANCELLED**

  Cause code indicating the user has terminated call without going on-hook.

- **CAUSE_DEST_NOT_OBTAINABLE**

  Cause code indicating the destination is not available.

- **CAUSE_INCOMPATIBLE_DESTINATION**

  Cause code indicating that a call has encountered an incompatible destination.

- **CAUSE_LOCKOUT**

  Cause code indicating that a call encountered inter-digit timeout while dialing.

- **CAUSE_NETWORK_CONGESTION**

  Cause code indicating call encountered network congestion.

- **CAUSE_NETWORK_NOT_OBTAINABLE**

  Cause code indicating call could not reach a destination network.

- **CAUSE_NEW_CALL**

  Cause code indicating that a new call.

- **CAUSE_NORMAL**

  Cause code indicating normal operation

- **CAUSE_RESOURCES_NOT_AVAILABLE**

  Cause code indicating resources were not available.

- **CAUSE_SNAPSHOT**

  Cause code indicating that the event is part of a snapshot of the current state of the call.

- **CAUSE_UNKNOWN**

  Cause code indicating the cause was unknown

- **META_CALL_ADDITIONAL_PARTY**

  Meta code description for the addition of a party to call.

- **META_CALL_ENDING**

  Meta code description for the entire call ending.

- **META_CALL_MERGING**

  Meta code description for an action of merging two calls.

- **META_CALL_PROGRESS**

  Meta code description for the progress of a call.

- **META_CALL_REMOVING_PARTY**

  Meta code description for a party leaving the call.

- **META_CALL_STARTING**

  Meta code description for the initiation or starting of a call.

- **META_CALL_TRANSFERRING**

Meta code description for an action of transferring one call to another.

- **META_SNAPSHOT**

  Meta code description for a snapshot of events.

- **META_UNKNOWN**

  Meta code is unknown.

Method Index

- **getCause**()

  Returns the cause associated with this event.

- **getID**()

  Returns the id of event.

- **getMetaCode**()

  Returns the meta code associated with this event.

- **getObserved**()

  Returns the object that is being observed. **Deprecated.**

- **isNewMetaEvent**()

  Returns true when this event is the start of a meta code group.

Variables

CAUSE_NORMAL

```
public static final int CAUSE_NORMAL
```

Cause code indicating normal operation

CAUSE_UNKNOWN

```
public static final int CAUSE_UNKNOWN
```

Cause code indicating the cause was unknown

CAUSE_CALL_CANCELLED

```
public static final int CAUSE_CALL_CANCELLED
```

Cause code indicating the user has terminated call without going on-hook.

CAUSE_DEST_NOT_OBTAINABLE

```
public static final int CAUSE_DEST_NOT_OBTAINABLE
```

Cause code indicating the destination is not available.

CAUSE_INCOMPATIBLE_DESTINATION

```
public static final int CAUSE_INCOMPATIBLE_DESTINATION
```

Cause code indicating that a call has encountered an incompatible destination.

CAUSE_LOCKOUT

```
public static final int CAUSE_LOCKOUT
```

Cause code indicating that a call encountered inter-digit timeout while dialing.

CAUSE_NEW_CALL

```
public static final int CAUSE_NEW_CALL
```

Cause code indicating that a new call.

CAUSE_RESOURCES_NOT_AVAILABLE

```
public static final int CAUSE_RESOURCES_NOT_AVAILABLE
```

Cause code indicating resources were not available.

CAUSE_NETWORK_CONGESTION

```
public static final int CAUSE_NETWORK_CONGESTION
```

Cause code indicating call encountered network congestion.

CAUSE_NETWORK_NOT_OBTAINABLE

```
public static final int CAUSE_NETWORK_NOT_OBTAINABLE
```

Cause code indicating call could not reach a destination network.

CAUSE_SNAPSHOT

```
public static final int CAUSE_SNAPSHOT
```

Cause code indicating that the event is part of a snapshot of the current state of the call.

META_CALL_STARTING

```
public static final int META_CALL_STARTING
```

Meta code description for the initiation or starting of a call. This implies that the call is a new call and in the active state with at least one Connection added to it.

META_CALL_PROGRESS

```
public static final int META_CALL_PROGRESS
```

Meta code description for the progress of a call. This indicates an update in state of certain objects in the call, or the addition of TerminalConnections (but not Connections).

META_CALL_ADDITIONAL_PARTY

```
public static final int META_CALL_ADDITIONAL_PARTY
```

Meta code description for the addition of a party to call. This includes adding a connection to the call.

META_CALL_REMOVING_PARTY

```
public static final int META_CALL_REMOVING_PARTY
```

Meta code description for a party leaving the call. This includes exactly one Connection moving to the `Connection.DISCONNECTED` state.

META_CALL_ENDING

```
public static final int META_CALL_ENDING
```

Meta code description for the entire call ending. This includes the call going to `Call.INVALID`, all of the Connections moving to the `Connection.DISCONNECTED` state.

META_CALL_MERGING

```
public static final int META_CALL_MERGING
```

Meta code description for an action of merging two calls. This involves the removal of one party from one call and the addition of the same party to another call.

META_CALL_TRANSFERRING

```
public static final int META_CALL_TRANSFERRING
```

Meta code description for an action of transferring one call to another. This involves the removal of parties from one call and the addition to another call, and the common party dropping off completely.

META_SNAPSHOT

```
public static final int META_SNAPSHOT
```

Meta code description for a snapshot of events.

META_UNKNOWN

```
public static final int META_UNKNOWN
```

Meta code is unknown.

Methods

getCause

```
public abstract int getCause()
```

Returns the cause associated with this event. Every event has a cause. The various cause values are defined as public static final variables in this interface.

Returns:

The cause of the event.

getMetaCode

```
public abstract int getMetaCode()
```

Returns the meta code associated with this event. The meta code provides a higher-level description of the event.

Returns:

The meta code for this event.

isNewMetaEvent

```
public abstract boolean isNewMetaEvent()
```

Returns true when this event is the start of a meta code group. This method is used to distinguish two contiguous groups of events bearing the same meta code.

Returns:

True if this event represents a new meta code grouping, false otherwise.

getID

```
public abstract int getID()
```

Returns the id of event. Every event has an id. The defined id of each event matches the object type of each event. The defined id allows applications to switch on event id rather than having to use multiple "if instanceof" statements.

Returns:

The id of the event.

getObserved

```
public abstract Object getObserved()
```

Note: getObserved() is deprecated. *Since JTAPI v1.2 This interface no longer needs to supply this information and may return null.*

Returns the object that is being observed.

Note: Implementation need no longer supply this information. The `CallObsevationEndedEv.getObservedObject()` method has been added which returns related information. This method may return null in JTAPI v1.2 and later.

Returns:

The object that is being observed.

All Packages This Package Previous Next

All Packages This Package Previous Next

Interface javax.telephony.events.ProvEv

public interface **ProvEv**

extends Ev

The `ProvEv` interface is the base interface for all Provider- related events. All events which pertain to the Provider object must extend this interface. Events which extend this interface are reported via the `ProviderObserver` interface.

The core package defines events which are reported when the Provider changes state. These events are: `ProvInServiceEv`, `ProvOutOfServiceEv`, and `ProvShutdownEv`. Also, the core package defines the `ProvObservationEndedEv` event which is sent when the Provider becomes unobservable.

The `ProvEv.getProvider()` method on this interface returns the Provider associated with the Provider event.

See Also:

ProvInServiceEv, ProvOutOfServiceEv, ProvShutdownEv, ProvObservationEndedEv, Ev, ProviderObserver, Provider

Method Index

getProvider()

Returns the Provider associated with this Provider event.

Methods

getProvider

```
public abstract Provider getProvider()
```

Returns the Provider associated with this Provider event.

Returns:

The Provider associated with this event.

All Packages This Package Previous Next

Interface javax.telephony.events.ProvInServiceEv

public interface **ProvInServiceEv**

extends ProvEv

The `ProvInServiceEv` interface indicates that the state of the Provider object has changed to `Provider.IN_SERVICE`. This interface extends the `ProvEv` interface and is reported via the `ProviderObserver` interface.

See Also:

Provider, ProviderObserver, ProvEv

Variable Index

- **ID**

  Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.events.ProvObservationEndedEv

public interface **ProvObservationEndedEv**

extends ProvEv

The `ProvObservationEndedEv` event indicates that the application will no longer receive Provider events on the instance of the ProviderObserver. This interface extends the ProvEv interface and is reported on the ProviderObserver interface.

See Also:

ProvEv, ProviderObserver

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.ProvOutOfServiceEv

public interface **ProvOutOfServiceEv**

extends ProvEv

The `ProvOutOfServiceEv` interface indicates that the state of the Provider object has changed to `Provider.OUT_OF_SERVICE`. This interface extends the `ProvEv` interface and is reported via the `ProviderObserver` interface.

See Also:

Provider, ProviderObserver, ProvEv

Variable Index

ID

Event id

Variables

ID

```
public static final int ID
```

Event id

Interface javax.telephony.events.ProvShutdownEv

public interface **ProvShutdownEv**

extends ProvEv

The `ProvShutdownEv` interface indicates that the state of the Provider object has changed to `Provider.SHUTDOWN`. This interface extends the `ProvEv` interface and is reported via the `ProviderObserver` interface.

See Also:

Provider, ProviderObserver, ProvEv

Variable Index

- **ID**

  Event id

Variables

● ID

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermConnActiveEv

public interface **TermConnActiveEv**

extends TermConnEv

The `TermConnActiveEv` interface indicates that the state of the TerminalConnection object has changed to `TerminalConnection.ACTIVE`. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermConnCreatedEv

public interface **TermConnCreatedEv**

extends TermConnEv

The `TermConnDroppedEv` interface indicates that a new TerminalConnection object has been created. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermConnDroppedEv

public interface **TermConnDroppedEv**

extends TermConnEv

The `TermConnDroppedEv` interface indicates that the state of the TerminalConnection object has changed to `TerminalConnection.DROPPED`. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

Interface javax.telephony.events.TermConnEv

public interface **TermConnEv**

extends CallEv

The `TermConnEv` interface is the base event interface for all TerminalConnection-related events. All events which pertain to the TerminalConnection object must extend this interface. This interface extends the `CallEv` interface and therefore is reported via the `CallObserver` interface.

The core package defines events which are reported when the TerminalConnection changes state. These events are: `TermConnRingingEv`, `TermConnActiveEv`, `TermConnPassiveEv`, `TermConnDroppedEv`, and `TermConnUnknownEv`. Also, a `TermConnCreatedEv` is sent when a new TerminalConnection is created.

The `TermConnEv.getTerminalConnection()` method on this interface returns the TerminalConnection associated with this TerminalConnection event.

See Also:

TerminalConnection, CallObserver, CallEv, TermConnEv, TermConnRingingEv, TermConnActiveEv, TermConnPassiveEv, TermConnDroppedEv, TermConnUnknownEv

Method Index

- **getTerminalConnection**()

  Returns the TerminalConnection associated with this event.

Methods

getTerminalConnection

```
public abstract TerminalConnection getTerminalConnection()
```

Returns the TerminalConnection associated with this event.

Returns:

The TerminalConnection associated with this event.

Interface javax.telephony.events.TermConnPassiveEv

public interface **TermConnPassiveEv**

extends TermConnEv

The `TermConnPassiveEv` interface indicates that the state of the TerminalConnection object has changed to `TerminalConnection.PASSIVE`. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermConnRingingEv

public interface **TermConnRingingEv**

extends TermConnEv

The `TermConnRingingEv` interface indicates that the state of the TerminalConnection object has changed to `TerminalConnection.RINGING`. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermConnUnknownEv

public interface **TermConnUnknownEv**

extends TermConnEv

The `TermConnUnknownEv` interface indicates that the state of the TerminalConnection object has changed to `TerminalConnection.UNKNOWN`. This interface extends the `TermConnEv` interface and is reported via the `CallObserver` interface.

See Also:

TerminalConnection, CallObserver, TermConnEv

Variable Index

● **ID**

Event id

Variables

● **ID**

```
public static final int ID
```

Event id

Interface javax.telephony.events.TermEv

public interface **TermEv**

extends Ev

The `TermEv` interface is the base interface for all Terminal- related events. All events which pertain to the Terminal object must extend this interface. Events which extend this interface are reported via the `TerminalObserver` interface.

The only event defined in the core package for the Terminal is the `TermObservationEndedEv`.

The `TermEv.getTerminal()` method on this interface returns the Terminal associated with the Terminal event.

See Also:

TermObservationEndedEv, Ev, TerminalObserver, Terminal

Method Index

- **getTerminal**()

  Returns the Terminal associated with this Terminal event.

Methods

getTerminal

```
public abstract Terminal getTerminal()
```

Returns the Terminal associated with this Terminal event.

Returns:

The Terminal associated with this event.

Interface javax.telephony.events.TermObservationEndedEv

public interface **TermObservationEndedEv**

extends TermEv

The `TermObservationEndedEv` event indicates that the application will no longer receive Terminal events on the instance of the TerminalObserver This interface extends the TermEv interface and is reported on the TerminalObserver interface.

See Also:

TermEv, TerminalObserver

Variable Index

- **ID**

  Event id

Variables

● ID

```
public static final int ID
```

Event id

All Packages

package javax.telephony.media

Interface Index

- MediaCallObserver
- MediaTerminalConnection

Interface javax.telephony.media.MediaCallObserver

public interface **MediaCallObserver**

extends CallObserver

The MediaCallObserver extends the CallObserver interface and reports all events pertaining to the MediaTerminalConnection object. Events for this object are reported on this observer because, in the core, TerminalConnection events are reported on the CallObserver object.

This interface does not have any methods. All events for the MediaTerminalConnection object are reported via the callChangedEvent() method on the CallObserver interface. All MediaTerminalConnection events, therefore, extend the core TermConnEv interface (which extends the core CallEv interface.

Applications which desire MediaTerminalConnection events implement this interface as a "signal" to the implementation that is wants to be sent events for the MediaTerminalConnection object.

Interface javax.telephony.media.MediaTerminalConnection

public interface **MediaTerminalConnection**

extends TerminalConnection

Introduction

The MediaTerminalConnection interface extends the TerminalConnection interface to add media capabilities. Media streams are associated with the TerminalConnection object in the call model. Therefore, different Terminals which are part of the same call at the same Address may have their own media streams. Additionality, Terminals which are part of more than one call have separate media streams for each of its calls.

The media interface consists of a base media API which supports all of the various types of media-based telephony applications. A simplier, voice-based API exist for applications which desire only the most simply voice-based media features. The base media API is still under development. This specification only represent the voice API.

The voice API supports the following applications: routing voice data to/from the telephone line to/from a workstation's speaker of microphone; routing voice data to/from the telephone line to/from audio files; starting and stoping of playing and recording; and DTMF tone detection.

In this specification, "playing" is defined as sending information to the telephone line. For example, an application would "play" an audio file to the telephone line for the opposite parties to hear. The term "recording" is defines as receiving information from the telephone line. For example, an application may "record" data from the telephone line into a file on disk.

Playing

For playing, applications may either route data from a URL with the usePlayURL() method or from the workstatation's default microphone using the useDefaultMicrophone() method. Note that if there is more than one microphone on the workstation, then the default microphone may be set using the javax.telephony.phone package. Applications begin playing using the startPlaying() method and stop playing using the stopPlaying() method. If an application issues a startPlaying() after a stopPlaying(), the implementation attempts to read from the media where it last left off, if possible. If the application wishes to "rewind" the media to the beginning, it should re-issue the usePlayURL() method.

Recording

For recording, applications may either route data to a URL with the useRecordURL() method or to the workstation's default speaker using the useDefaultSpeaker() method. Note that if there is more than one speaker on the workstation, then the default speaker may be set using the javax.telephony.phone package. Applications begin recording using the startRecording() method and stop recording using the stopRecording() method. If an application issues a startRecording() after a stopRecording(), the implementation attempts to write to the media where it last left off, if possible. If the application wishes to "overwrite" the media from the beginning, it should re-issue the useRecordURL() method.

Variable Index

- **AVAILABLE**

  Media is currently available on this terminal connection
- **NOACTIVITY**

  There is currently no activity on this TerminalConnection.
- **PLAYING**

  There is currently playing on this terminal connection
- **RECORDING**

There is currently recording on this terminal connection

- **UNAVAILABLE**

  Media is currently not available on this terminal connection

Method Index

- **generateDtmf**(String)
- **getMediaAvailability**()

  Returns the current media availability state, either AVAILABLE or UNAVAILABLE.
- **getMediaState**()

  Returns the current state of the terminal connection as a bit mask of PLAYING and RECORDING.
- **setDtmfDetection**(boolean)
- **startPlaying**()

  Start the playing.
- **startRecording**()

  Start the recording.
- **stopPlaying**()

  Stop the playing.
- **stopRecording**()

  Stop the recording.
- **useDefaultMicrophone**()

  Instructs the terminal connection to use the default microphone for playing to the telephone line.
- **useDefaultSpeaker**()

  Instructs the terminal connection to use the default speaker for recording from the telephone line.
- **usePlayURL**(URL)

  Instructs the terminal connection to use a file for playing to the telephone line.
- **useRecordURL**(URL)

  Instructs the terminal connection to use a file for recording from the telephone line.

Variables

AVAILABLE

```
 public static final int AVAILABLE
```

Media is currently available on this terminal connection

UNAVAILABLE

```
 public static final int UNAVAILABLE
```

Media is currently not available on this terminal connection

PLAYING

```
 public static final int PLAYING
```

There is currently playing on this terminal connection

RECORDING

```
public static final int RECORDING
```

There is currently recording on this terminal connection

NOACTIVITY

```
public static final int NOACTIVITY
```

There is currently no activity on this TerminalConnection.

Methods

getMediaAvailability

```
public abstract int getMediaAvailability()
```

Returns the current media availability state, either AVAILABLE or UNAVAILABLE.

Returns:

The current availability of the media channel.

getMediaState

```
public abstract int getMediaState()
```

Returns the current state of the terminal connection as a bit mask of PLAYING and RECORDING. If there is not activity, then this method returns `MediaTerminalConnection.NOACTIVITY`.

Returns:

The current state of playing or recording.

useDefaultSpeaker

```
public abstract void useDefaultSpeaker() throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException
```

Instructs the terminal connection to use the default speaker for recording from the telephone line.

Throws: PrivilegeViolationException

Indicates the application is not permitted to direct voice media to the default speaker.

Throws: ResourceUnavailableException

Indicates that the speaker is not currently available for use.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

useRecordURL

```
public abstract void useRecordURL(URL url) throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException
```

Instructs the terminal connection to use a file for recording from the telephone line.

Parameters:

url - The URL-destination for the voice data for recording.

Throws: PrivilegeViolationException

Indicates the application is not permitted to use the give URL for recording.

Throws: ResourceUnavailableException

Indicates the URL given is not available, either because the URL was invalid or a network problem occurred.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

useDefaultMicrophone

```
 public abstract void useDefaultMicrophone() throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException
```

Instructs the terminal connection to use the default microphone for playing to the telephone line.

Throws: PrivilegeViolationException

Indicates the application is not permitted to direct voice media from the default microphone.

Throws: ResourceUnavailableException

Indicates that the microphone is not currently available for use.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

usePlayURL

```
 public abstract void usePlayURL(URL url) throws PrivilegeViolationException,
ResourceUnavailableException, MethodNotSupportedException
```

Instructs the terminal connection to use a file for playing to the telephone line.

Parameters:

url - The URL-source of the voice data to play. valid or available source of voice data.

Throws: PrivilegeViolationException

Indicates the application is not permitted to use the give URL for playing.

Throws: ResourceUnavailableException

Indicates the URL given is not available, either because the URL was invalid or a network problem occurred.

Throws: MethodNotSupportedException

This method is not supported by the implementation.

startPlaying

```
 public abstract void startPlaying() throws MethodNotSupportedException,
ResourceUnavailableException, InvalidStateException
```

Start the playing. This method returns once playing has begun, that is, when getMediaState() & PLAYING == PLAYING.

Throws: MethodNotSupportedException

The implementation does not support playing to the telephone line.

Throws: ResourceUnavailableException

Indicates playing is not able to be started because some resource is unavailable.

Throws: InvalidStateException

Indicates the TerminalConnection is not in the media channel available state.

stopPlaying

```
 public abstract void stopPlaying()
```

Stop the playing. This method returns once the playing has stopped, that is, when getMediaState() & PLAYING == 0. If playing is not currently taking place, this method has no effect.

startRecording

```
 public abstract void startRecording() throws MethodNotSupportedException,
ResourceUnavailableException, InvalidStateException
```

Start the recording. This method returns once the recording has started, that is, when getMediaState() & RECORDING == RECORDING.

Throws: MethodNotSupportedException

The implementation does not support recording from the telephone line.

Throws: ResourceUnavailableException

Indicates recording is not able to be started because some resource is unavailable.

Throws: InvalidStateException

Indicates the TerminalConnection is not in the media channel available state.

stopRecording

```
 public abstract void stopRecording()
```

Stop the recording. This method returns once the recording has stopped, that is, when getMediaState() & RECORDING == 0. If recording is not currently taking place, this method has no effect.

setDtmfDetection

```
 public abstract void setDtmfDetection(boolean enable) throws
MethodNotSupportedException, ResourceUnavailableException, InvalidStateException
```

generateDtmf

```
 public abstract void generateDtmf(String digits) throws
MethodNotSupportedException, ResourceUnavailableException, InvalidStateException
```

All Packages   This Package   Previous   Next

All Packages

package javax.telephony.media.capabilities

Interface Index

- MediaTerminalConnectionCapabilities

Interface javax.telephony.media.capabilities.MediaTerminalConnectionCapabilities

public interface **MediaTerminalConnectionCapabilities**

extends TerminalConnectionCapabilities

The MediaTerminalConnectionCapabilities interface extends the TerminalConnectionCapabilities interface. This interface provides capabilities methods for the MediaTerminalConnection object. The methods in this interface provides applications the ability to query for those actions where are possible on the MediaTerminalConnection interface as part of the capabilities package.

Method Index

- **canDetectDtmf**()

  This method returns true if the application is able to detect DTMF-tones on the telephone line.
- **canGenerateDtmf**()

  This method returns true if the application is able to generate DTMF- tones the telephone line.
- **canStartPlaying**()

  This method returns true if the application is able to start playing to the telephone line.
- **canStartRecording**()

  This method returns true if the application is able to start recording from the telephone line.
- **canStopPlaying**()

  This method returns true if the application is able to stop playing to the telephone line.
- **canStopRecording**()

  This method returns true if the application is able to stop recording from the telephone line.
- **canUseDefaultMicrophone**()

  This method returns true if the application can invoke the useDefaultMicrophone() method and route the media from the default microphone.
- **canUseDefaultSpeaker**()

  This method returns true if the application can invoke the useDefaultSpeaker() method and route the media from the telephone line to the default speaker.
- **canUsePlayURL**()

  This method returns true if the application can invoke the usePlayURL() method and route voice media from URL's.
- **canUseRecordURL**()

  This method returns true if the application can invoke the useRecordURL() method and route voice media to URL's.

Methods

canUseDefaultSpeaker

```
public abstract boolean canUseDefaultSpeaker()
```

This method returns true if the application can invoke the useDefaultSpeaker() method and route the media from the telephone line to the default speaker. Returns false otherwise.

Returns:

True if the application can route voice media to the default speaker, false otherwise.

canUseDefaultMicrophone

```
public abstract boolean canUseDefaultMicrophone()
```

This method returns true if the application can invoke the useDefaultMicrophone() method and route the media from the default microphone. Returns false otherwise.

Returns:

True if the application can route voice media from the default microphone, false otherwise.

canUseRecordURL

```
public abstract boolean canUseRecordURL()
```

This method returns true if the application can invoke the useRecordURL() method and route voice media to URL's. Returns false otherwise.

Returns:

True if the application can route voice media to URL's, false otherwise.

canUsePlayURL

```
public abstract boolean canUsePlayURL()
```

This method returns true if the application can invoke the usePlayURL() method and route voice media from URL's. Returns false otherwise.

Returns:

True if the application can route voice media from URL's, false otherwise.

canStartPlaying

```
public abstract boolean canStartPlaying()
```

This method returns true if the application is able to start playing to the telephone line. Returns false otherwise.

Returns:

True if the application can begin playing to the telephone line, false otherwise.

canStopPlaying

```
public abstract boolean canStopPlaying()
```

This method returns true if the application is able to stop playing to the telephone line. Returns false otherwise.

Returns:

True if the application can stop playing to the telephone line, false otherwise.

canStartRecording

```
public abstract boolean canStartRecording()
```

This method returns true if the application is able to start recording from the telephone line. Returns false otherwise.

Returns:

True if the application can start recording from the telephone line, false otherwise.

canStopRecording

```
public abstract boolean canStopRecording()
```

This method returns true if the application is able to stop recording from the telephone line. Returns false otherwise.

Returns:

True if the application can stop recording from the telephone line, false otherwise.

canDetectDtmf

```
public abstract boolean canDetectDtmf()
```

This method returns true if the application is able to detect DTMF-tones on the telephone line. Returns false otherwise. This method indicates whether the application is able to invoke the setDtmfDetection(true) method.

Returns:

True if the application can detect DTMF-tones from the telephone line, false otherwise.

canGenerateDtmf

```
public abstract boolean canGenerateDtmf()
```

This method returns true if the application is able to generate DTMF- tones the telephone line. Returns false otherwise.

Returns:

True if the application can generate DTMF-tones to the telephone line, false otherwise.

All Packages

package javax.telephony.media.events

Interface Index

- MediaEv
- MediaTermConnAvailableEv
- MediaTermConnDtmfEv
- MediaTermConnEv
- MediaTermConnStateEv
- MediaTermConnUnavailableEv

Interface javax.telephony.media.events.MediaEv

public interface **MediaEv**

extends Ev

The MediaEv is the base event for all events in the Media package. Each event in this package must extend this interface. This interface is not meant to be a public interface, it is just a building block for other event interfaces.

The MediaEv interface contains getMediaCause(), which returns the reason for the event.

Variable Index

- **CAUSE_NORMAL**

  Cause code indicating normal operation
- **CAUSE_UNKNOWN**

  Cause code indicating the cause was unknown

Method Index

- **getMediaCause**()

  Returns the media and core causes associated with this event.

Variables

CAUSE_NORMAL

```
public static final int CAUSE_NORMAL
```

Cause code indicating normal operation

CAUSE_UNKNOWN

```
public static final int CAUSE_UNKNOWN
```

Cause code indicating the cause was unknown

Methods

getMediaCause

```
public abstract int getMediaCause()
```

Returns the media and core causes associated with this event. Every event has a cause. The various cause values are defined as public static final variablies in this interface, with the exception of CAUSE_NORMAL and CAUSE_UNKNOWN, which are defined in the core.

Returns:

s The cause of the event.

All Packages   This Package Previous Next

Interface javax.telephony.media.events.MediaTermConnAvailableEv

public interface **MediaTermConnAvailableEv**

extends MediaTermConnEv

The MediaTermConnAvailableEv interface indicates that media is currently available on the TerminalConnection. Media becomes available on the TerminalConnection when the state of the TerminalConnection changes with respect to the telephone call. For example, when a TerminalConnection becomes active on the telephone call, media is made available to the application. This event interface extends the javax.telephony.events.TermConnEv interface, through which the application may obtain the TerminalConnection object associated with this event.

Variable Index

- **ID**

  Event id

Variables

● ID

```
public static final int ID
```

Event id

Interface javax.telephony.media.events.MediaTermConnDtmfEv

public interface **MediaTermConnDtmfEv**

extends MediaTermConnEv

The MediaTermConnDtmfEv interface indicates that a DTMF-tone has been detection on the telephone line. This event interface extends the javax.telephony.events.TermConnEv interface, through which the application may obtain the TerminalConnection object associated with this event.

Applications may obtain the detected DTMF-digit via the getDtmfDigit() method on this interface.

Variable Index

ID

Event id

Method Index

getDtmfDigit()

Returns the DTMF-digit which has been recognized.

Variables

ID

```
public static final int ID
```

Event id

Methods

getDtmfDigit

```
public abstract char getDtmfDigit()
```

Returns the DTMF-digit which has been recognized. This digit may either be the numbers zero through nine (0-9), the asterisk (*), or the pound (#).

Returns:

The DTMF-digit which has been detected.

All Packages This Package Previous Next

Interface javax.telephony.media.events.MediaTermConnEv

public interface **MediaTermConnEv**
extends MediaEv, TermConnEv

Interface javax.telephony.media.events.MediaTermConnStateEv

public interface **MediaTermConnStateEv**

extends MediaTermConnEv

The MediaTermConnStateEv interface indicates that the playing/recording state has changed on the TerminalConnection object. This event interface extends the javax.telephony.events.TermConnEv interface, through which the application may obtain the TerminalConnection object associated with this event.

Applications may obtain the new state via the getMediaState() method on this interface, or via the MediaTerminalConnection.getMediaState() method.

Variable Index

ID

Event id

Method Index

getMediaState()

Returns the current state of playing/recording on the TerminalConnection in the form of a bit mask.

Variables

ID

```
public static final int ID
```

Event id

Methods

getMediaState

```
public abstract int getMediaState()
```

Returns the current state of playing/recording on the TerminalConnection in the form of a bit mask.

Returns:

The current playing/recording state.

Interface javax.telephony.media.events.MediaTermConnUnavailableEv

public interface **MediaTermConnUnavailableEv**

extends MediaTermConnEv

The MediaTermConnUnavailableEv interface indicates that there is currently no media available on the TerminalConnection. This event is most likely cause by a change in state of the TerminalConnection which respect to the call. For example, when someone goes on hold, media is no longer avaiable on that TerminalConnection. This event interface extends the javax.telephony.events.TermConnEv interface, through which the application may obtain the TerminalConnection object associated with this event.

Variable Index

● **ID**

Event id

Variables

● **ID**

```
 public static final int ID
```

Event id

package javax.telephony.phone

Interface Index

- Component
- ComponentGroup
- PhoneButton
- PhoneDisplay
- PhoneGraphicDisplay
- PhoneHookswitch
- PhoneLamp
- PhoneMicrophone
- PhoneRinger
- PhoneSpeaker
- PhoneTerminal
- PhoneTerminalObserver

Interface javax.telephony.phone.Component

public interface **Component**

The Component interface is the base interface for all individual components used to model telephone hardware. Each individual component extends this interface.

Each component is identified not only by its type, but also by an identifying name, which may be obtained via the *getName()* method on this interface.

Method Index

- **getCapabilities**()

  Returns the dynamic capabilities for this `Component` instance.
- **getName**()

  Returns the name of the Component.

Methods

getName

```
public abstract String getName()
```

Returns the name of the Component.

Returns:

The name of this component.

getCapabilities

```
public abstract ComponentCapabilities getCapabilities()
```

Returns the dynamic capabilities for this `Component` instance. Static capabilities are not available for components.

Returns:

The dynamic component capabilities.

All Packages This Package Previous Next

Interface javax.telephony.phone.ComponentGroup

public interface **ComponentGroup**

A ComponentGroup is a grouping of Component objects. Terminals may be composed of zero or more ComponentGroups. Applications query the PhoneTerminal interface for the available ComponentGroups. Then they query this interface for the components which make up this component group.

Variable Index

- **HAND_SET**

  The component group is of type HAND_SET.
- **HEAD_SET**

  The component group is of type HEAD_SET.
- **OTHER**

  The component group is of type OTHER.
- **PHONE_SET**

  The componet group is of type PHONE_SET.
- **SPEAKER_PHONE**

  The component group is of type SPEAKER_PHONE.

Method Index

- **activate**()

  Enables all routing of events or media stream between all Components of this group and calls on any of the Addresses asociated with the parent Terminal.
- **activate**(Address)

  Enables all routing of events or media stream between all Components of this group and calls to the specified Address.
- **deactivate**()

  Disables all routing of events or media stream between all Components of this group and calls on any of the Addresses associated with the parent Terminal.
- **deactivate**(Address)

  Disables all routing of events or media stream between all Components of this group and the specified Address.
- **getCapabilities**()

  Returns the dynamic capabilities for this `ComponentGroup` instance.
- **getComponents**()

  Returns the groups components, null if the group contains zero components.
- **getDescription**()

  Returns a string describing the component group.

- getType()

  Returns the type of group, either HEAD_SET, HAND_SET, SPEAKER_PHONE, PHONE_SET or OTHER.

Variables

HEAD_SET

```
public static final int HEAD_SET
```

The component group is of type HEAD_SET.

HAND_SET

```
public static final int HAND_SET
```

The component group is of type HAND_SET.

SPEAKER_PHONE

```
public static final int SPEAKER_PHONE
```

The component group is of type SPEAKER_PHONE.

PHONE_SET

```
public static final int PHONE_SET
```

The componet group is of type PHONE_SET.

OTHER

```
public static final int OTHER
```

The component group is of type OTHER.

Methods

getType

```
public abstract int getType()
```

Returns the type of group, either HEAD_SET, HAND_SET, SPEAKER_PHONE, PHONE_SET or OTHER.

Returns:

The type of group.

getDescription

```
public abstract String getDescription()
```

Returns a string describing the component group.

Returns:

A string description of the component group.

getComponents

```
public abstract Component[] getComponents()
```

Returns the groups components, null if the group contains zero components.

Returns:

An array of Component objects.

activate

```
public abstract boolean activate()
```

Enables all routing of events or media stream between all Components of this group and calls on any of the Addresses asociated with the parent Terminal.

Returns:

true if successful and false if unsuccessful.

deactivate

```
public abstract boolean deactivate()
```

Disables all routing of events or media stream between all Components of this group and calls on any of the Addresses associated with the parent Terminal.

Returns:

true if successful and false if unsuccessful.

activate

```
public abstract boolean activate(Address address) throws InvalidArgumentException
```

Enables all routing of events or media stream between all Components of this group and calls to the specified Address.

Parameters:

address - The Address that the group is to be activated on.

Returns:

true if successful and false if unsuccessful.

Throws: InvalidArgumentException

The provided Address is not valid for the Terminal.

deactivate

```
public abstract boolean deactivate(Address address) throws InvalidArgumentException
```

Disables all routing of events or media stream between all Components of this group and the specified Address.

Parameters:

address - The Address that the group is to be deactivated on.

Returns:

true if successful and false if unsuccessful.

Throws: InvalidArgumentException

The provided Address is not valid for the Terminal.

getCapabilities

```
public abstract ComponentGroupCapabilities getCapabilities()
```

Returns the dynamic capabilities for this `ComponentGroup` instance. Static capabilities are not available for component groups.

Returns:

The dynamic component group capabilities.

Interface javax.telephony.phone.PhoneButton

public interface **PhoneButton**
extends Component

Method Index

- **buttonPress**()

  Press the button.
- **getAssociatedPhoneLamp**()

  Returns the associated lamp information.
- **getInfo**()

  Returns the button information.
- **setInfo**(String)

  Sets button information.

Methods

getInfo

```
public abstract String getInfo()
```

Returns the button information.

Returns:

The string button information.

setInfo

```
public abstract void setInfo(String buttonInfo)
```

Sets button information.

Parameters:

buttonInfo - The button information.

getAssociatedPhoneLamp

```
public abstract PhoneLamp getAssociatedPhoneLamp()
```

Returns the associated lamp information.

Returns:

The associated lamp object.

buttonPress

```
public abstract void buttonPress()
```

Press the button.

All Packages This Package Previous Next

Interface javax.telephony.phone.PhoneDisplay

public interface **PhoneDisplay**
extends Component

Method Index

● **getDisplay**(int, int)

Returns the displayed string starting at coordinates (x, y).

● **getDisplayColumns**()

Returns the number of display columns.

● **getDisplayRows**()

Returns the number of display rows.

● **setDisplay**(String, int, int)

Displays the given string starting at coordinates (x, y).

Methods

● getDisplayRows

```
 public abstract int getDisplayRows()
```

Returns the number of display rows.

Returns:

The number of display rows.

● getDisplayColumns

```
 public abstract int getDisplayColumns()
```

Returns the number of display columns.

Returns:

The number of display columns.

● getDisplay

```
 public abstract String getDisplay(int x,
                                   int y) throws InvalidArgumentException
```

Returns the displayed string starting at coordinates (x, y).

Parameters:

x - The x-coordinate.

y - The y-coordinate.

Returns:

The string displayed starting at coordinates (x, y).

Throws: InvalidArgumentException

Either the coordinates provided were invalid.

setDisplay

```
 public abstract void setDisplay(String string,
                                 int x,
                                 int y) throws InvalidArgumentException
```

Displays the given string starting at coordinates (x, y).

Parameters:

string - The string to display.

x - The x-coordinate.

y - The y-coordinate.

Throws: InvalidArgumentException

Either the coordinates provided were invalid.

All Packages   This Package   Previous   Next

Interface javax.telephony.phone.PhoneGraphicDisplay

public interface **PhoneGraphicDisplay**
extends Component

A PhoneGraphicsDisplay represents a display device that is pixel-addressable, and which can be drawn into using AWT primitives.

Method Index

- **getGraphics**()

  Returns a Graphics object for drawing into the display.
- **size**()

  Returns the size of the display.

Methods

● getGraphics

```
public abstract Graphics getGraphics()
```

Returns a Graphics object for drawing into the display.

Returns:

A Graphic object, as defined in the AWT.

● size

```
public abstract Dimension size()
```

Returns the size of the display.

Returns:

The size of the display, packaged in an AWT Dimension object.

Interface javax.telephony.phone.PhoneHookswitch

public interface **PhoneHookswitch**
extends Component

Variable Index

- **OFF_HOOK**

  The Hookswitch is OFF_HOOK.

- **ON_HOOK**

  The Hookswitch is ON_HOOK.

Method Index

- **getHookSwitchState**()

  Returns the current state of the hookswitch.

- **setHookSwitch**(int)

  Sets the state of the hookswitch to either ON_HOOK or OFF_HOOK.

Variables

ON_HOOK

```
 public static final int ON_HOOK
```

The Hookswitch is ON_HOOK.

OFF_HOOK

```
 public static final int OFF_HOOK
```

The Hookswitch is OFF_HOOK.

Methods

setHookSwitch

```
 public abstract void setHookSwitch(int hookSwitchState) throws
```

```
InvalidArgumentException
```

Sets the state of the hookswitch to either ON_HOOK or OFF_HOOK.

Parameters:

hookSwtichState - The desired state of the hook switch.

Throws: InvalidArgumentException

The provided hookswitch state is not valid.

● getHookSwitchState

```
 public abstract int getHookSwitchState()
```

Returns the current state of the hookswitch.

Returns:

The current state of the hookswitch.

All Packages   This Package Previous Next

Interface javax.telephony.phone.PhoneLamp

public interface **PhoneLamp**
extends Component

Variable Index

LAMPMODE_BROKENFLUTTER

The lamp mode is BROKENFLUTTER, which is the superposition of flash and flutter.

LAMPMODE_FLASH

The lamp mode is FLASH, which means slow on and off.

LAMPMODE_FLUTTER

The lamp mode is FLUUTER, which means fast on and off.

LAMPMODE_OFF

The lamp mode is OFF.

LAMPMODE_STEADY

The lamp is STEADY, which means continuously lit.

LAMPMODE_WINK

The lamp mode is WINK.

Method Index

getAssociatedPhoneButton()

Returns the button associated with the lamp.

getMode()

Returns the current lamp mode.

getSupportedModes()

Returns an array of supported lamp modes.

setMode(int)

Sets the current lamp mode to a mode supported by the lamp and returns by getSupportedModes().

Variables

LAMPMODE_OFF

```
public static final int LAMPMODE_OFF
```

The lamp mode is OFF.

LAMPMODE_FLASH

```
public static final int LAMPMODE_FLASH
```

The lamp mode is FLASH, which means slow on and off.

LAMPMODE_STEADY

```
public static final int LAMPMODE_STEADY
```

The lamp is STEADY, which means continuously lit.

LAMPMODE_FLUTTER

```
public static final int LAMPMODE_FLUTTER
```

The lamp mode is FLUUTER, which means fast on and off.

LAMPMODE_BROKENFLUTTER

```
public static final int LAMPMODE_BROKENFLUTTER
```

The lamp mode is BROKENFLUTTER, which is the superposition of flash and flutter.

LAMPMODE_WINK

```
public static final int LAMPMODE_WINK
```

The lamp mode is WINK.

Methods

getSupportedModes

```
public abstract int[] getSupportedModes()
```

Returns an array of supported lamp modes.

Returns:

An array of supported lamp modes.

setMode

```
public abstract void setMode(int mode) throws InvalidArgumentException
```

Sets the current lamp mode to a mode supported by the lamp and returns by getSupportedModes().

Parameters:

mode - The desired lamp mode.

Throws: InvalidArgumentException

The provided lamp mode is not valid.

getMode

```
public abstract int getMode()
```

Returns the current lamp mode.

Returns:

The current lamp mode.

getAssociatedPhoneButton

```
public abstract PhoneButton getAssociatedPhoneButton()
```

Returns the button associated with the lamp.

Returns:

The button associated with the lamp.

All Packages This Package Previous Next

Interface javax.telephony.phone.PhoneMicrophone

public interface **PhoneMicrophone**
extends Component

Variable Index

- **FULL**

  The full microhphone gain.
- **MID**

  The microphone gain is MID.
- **MUTE**

  The microphone gain is MUTE.

Method Index

- **getGain**()

  Returns the current microphone gain.
- **setGain**(int)

  Sets the microphone gain to a value between MUTE and FULL, inclusive.

Variables

● MUTE

```
public static final int MUTE
```

The microphone gain is MUTE.

● MID

```
public static final int MID
```

The microphone gain is MID.

● FULL

```
public static final int FULL
```

The full microhphone gain.

Methods

getGain

```
public abstract int getGain()
```

Returns the current microphone gain.

Returns:

The current microphone gain.

setGain

```
public abstract void setGain(int gain) throws InvalidArgumentException
```

Sets the microphone gain to a value between MUTE and FULL, inclusive.

Parameters:

gain - A microphone gain between MUTE and FULL, inclusive.

Throws: InvalidArgumentException

The microphone gain is not valid.

All Packages This Package Previous Next

Interface javax.telephony.phone.PhoneRinger

public interface **PhoneRinger**
extends Component

Variable Index

FULL

Ringer volume definition for the ringer at maximum volume.

MIDDLE

Ringer volume definition for the middle volume.

OFF

Ringer volume definition for the ringer off.

Method Index

getNumberOfRingPatterns()

Returns the number of available ringing patterns.

getNumberOfRings()

Returns the number of complete ring cycles that the ringer has been ringing.

getRingerPattern()

Returns the current ringer pattern.

getRingerVolume()

Returns the current ringer volume.

isRingerOn()

Returns true if the ringer is on, false otherwise.

setRingerPattern(int)

Set the ringer pattern given an valid index number returned by getNumberOfRingPatterns().

setRingerVolume(int)

Sets the ringer volume between ZERO or FULL, inclusive.

Variables

OFF

```
 public static final int OFF
```

Ringer volume definition for the ringer off.

MIDDLE

```
public static final int MIDDLE
```

Ringer volume definition for the middle volume.

FULL

```
public static final int FULL
```

Ringer volume definition for the ringer at maximum volume.

Methods

isRingerOn

```
public abstract int isRingerOn()
```

Returns true if the ringer is on, false otherwise.

Returns:

True if the ringer is on, false otherwise

getRingerVolume

```
public abstract int getRingerVolume()
```

Returns the current ringer volume.

Returns:

The current ringer volume.

setRingerVolume

```
public abstract void setRingerVolume(int volume) throws InvalidArgumentException
```

Sets the ringer volume between ZERO or FULL, inclusive.

Parameters:

volume - The ringer volume, between ZERO and FULL, inclusive.

Throws: InvalidArgumentException

The volume provided was not valid.

getRingerPattern

```
public abstract int getRingerPattern()
```

Returns the current ringer pattern.

Returns:

The current ringer pattern.

getNumberOfRingPatterns

```
public abstract int getNumberOfRingPatterns()
```

Returns the number of available ringing patterns. An index between zero and the returns value minus one may be used for the setRingerPattern() method.

Returns:

The number of available ringer patterns.

● setRingerPattern

```
 public abstract void setRingerPattern(int ringerPattern) throws
InvalidArgumentException
```

Set the ringer pattern given an valid index number returned by getNumberOfRingPatterns().

Parameters:

ringerPattern - The desired ringer pattern.

Throws: InvalidArgumentException

The ring pattern provided was not valid.

● getNumberOfRings

```
 public abstract int getNumberOfRings()
```

Returns the number of complete ring cycles that the ringer has been ringing. A value of 0 indicates that the ringer is not being rung.

Returns:

The current ringer count.

All Packages   This Package   Previous   Next

Interface javax.telephony.phone.PhoneSpeaker

public interface **PhoneSpeaker**
extends Component

Variable Index

- **FULL**
  Speaker volume definition for highest volume.
- **MID**
  Speaker volume definition for the middle volume.
- **MUTE**
  Speaker volume definition for muting.

Method Index

- **getVolume**()
  Returns the volume of the speaker.
- **setVolume**(int)
  Sets the speaker or handset volume.

Variables

MUTE

```
public static final int MUTE
```

Speaker volume definition for muting.

MID

```
public static final int MID
```

Speaker volume definition for the middle volume.

FULL

```
public static final int FULL
```

Speaker volume definition for highest volume.

Methods

getVolume

```
public abstract int getVolume()
```

Returns the volume of the speaker.

Returns:

The volume of the speaker.

setVolume

```
public abstract void setVolume(int volume)
```

Sets the speaker or handset volume. The volume value may be anything between MUTE or FULL, inclusive.

Parameters:

volume - The volume, between MUTE and FULL.

All Packages This Package Previous Next

Interface javax.telephony.phone.PhoneTerminal

public interface **PhoneTerminal**

extends Terminal

The PhoneTerminal interface extends the Terminal interface to provide functionality for the Phone package. It allows applications to obtain arrays of telephony Components (each group is called a ComponentGroup) which represents the physical components of telephones.

Method Index

getComponentGroups()

Returns an array of ComponentGroup objects available on the Terminal.

Methods

getComponentGroups

```
 public abstract ComponentGroup[] getComponentGroups()
```

Returns an array of ComponentGroup objects available on the Terminal. A ComponentGroup object is composed of a number of Components. Examples of Component objects include headsets, handsets, speakerphones, and buttons. ComponentGroup objects group Components together.

Returns:

An array of ComponetGroup objects on this Terminal.

Interface javax.telephony.phone.PhoneTerminalObserver

public interface **PhoneTerminalObserver**

extends TerminalObserver

The PhoneTerminalObserver interface is used to report all Phone-related events. Note that this observer does not have any method associated with it. Applications which implement a TerminalObserver class should also implement this interface to indicate to the implementation that it wants Phone-related events sent to it. If an application's observer does not implement this interface, phone-related events will not be sent to the application.

All Packages

---

package javax.telephony.phone.capabilities

Interface Index

- ComponentCapabilities
- ComponentGroupCapabilities

Interface javax.telephony.phone.capabilities.ComponentCapabilities

public interface **ComponentCapabilities**

Method Index

- **canControl**()

  Returns true if the component can be controlled.
- **canObserve**()

  Returns true if the component can be observed.

Methods

canObserve

```
public abstract boolean canObserve()
```

Returns true if the component can be observed. For example, this method on a PhoneMicrophone component would return true, if events for changes in gain setting can be received through the TerminalObserver interface and also if the "get" methods on each of the component interfaces is expected to be successful.

Returns:

True if the component can be observed, false otherwise.

canControl

```
public abstract boolean canControl()
```

Returns true if the component can be controlled. For example, this method on a PhoneMicrophone component would return true, if the gain setting can be adjusted programmatically.

Returns:

True if the componet can be controlled, false otherwise.

Interface javax.telephony.phone.capabilities.ComponentGroupCapabilities

public interface **ComponentGroupCapabilities**

Method Index

- **canActivate**()

  Returns true if the ComponentGroup can be "activated" on the Terminal that the ComponentGroup is associated with.

- **canActivate**(Address)

  Returns true if the ComponentGroup can be "activated" on the specified Address at the Terminal that the ComponentGroup is associated with.

Methods

canActivate

```
public abstract boolean canActivate()
```

Returns true if the ComponentGroup can be "activated" on the Terminal that the ComponentGroup is associated with. For example, activation of a headset on a certain Terminal allows media to flow between the headset and the telephone line associated with the terminal for all calls on the line . This method allows the application to determine if activation of the ComponentGroup on its Terminal is supported.

Returns:

True if the component group can be activated on its Terminal, false otherwise.

canActivate

```
public abstract boolean canActivate(Address address)
```

Returns true if the ComponentGroup can be "activated" on the specified Address at the Terminal that the ComponentGroup is associated with. For example, activation of a headset on a certain Address at a Terminal allows media to flow between the headset and the telephone line associated with the Terminal for all calls on the specified Address. This method allows the application to determine if activation of the ComponentGroup on a specific Address at a Terminal is supported.

Returns:

True if the component group can be activated on its Terminal at the specified Address, false otherwise.

All Packages

package javax.telephony.phone.events

Interface Index

- ButtonInfoEv
- ButtonPressEv
- DisplayUpdateEv
- HookswitchStateEv
- LampModeEv
- MicrophoneGainEv
- PhoneEv
- PhoneTermEv
- RingerPatternEv
- RingerVolumeEv
- SpeakerVolumeEv

Interface javax.telephony.phone.events.ButtonInfoEv

public interface **ButtonInfoEv**

extends PhoneTermEv

The ButtonInfoEv interface extends the PhoneTermEv interface and is reported via the PhoneTermObserver interface. This event interface indicates the information associated with a button component has changed.

Applications may obtain the new information associated with this button via the *getInfo()* method on this interface. The old information (before the change) may be obtained via the *getOldInfo()* method on this interface.

Variable Index

● **ID**

Event id

Method Index

● **getInfo**()

Returns the button information.

● **getOldInfo**()

Returns the information previously associated with this button.

Variables

● **ID**

```
public static final int ID
```

Event id

Methods

● **getInfo**

```
public abstract String getInfo()
```

Returns the button information.

Returns:

The string button information.

● **getOldInfo**

```
 public abstract String getOldInfo()
```

Returns the information previously associated with this button.

Returns:

The old button information.

All Packages This Package Previous Next

All Packages This Package Previous Next

Interface javax.telephony.phone.events.ButtonPressEv

public interface **ButtonPressEv**

extends PhoneTermEv

The ButtonPressEv interface extends the PhoneTermEv interface and is reported via the PhoneTermObserver interface. This event interface indicates that a button component has been pressed.

Applications may obtain the identifying information associated with this button via the *getInfo()* method.

Variable Index

- **ID**

  Event id

Method Index

- **getInfo**()

  Returns the button information.

Variables

ID

```
public static final int ID
```

Event id

Methods

getInfo

```
public abstract String getInfo()
```

Returns the button information.

Returns:

The string button information.

Interface javax.telephony.phone.events.DisplayUpdateEv

public interface **DisplayUpdateEv**

extends PhoneTermEv

The DisplayUpdateEv interface extends the PhoneTermEv interface and is reported via the PhoneTermObserver interface. This event interface indicates that the contents of the display component has changed.

Applications may obtain the new contents of the display component via the *getDisplay(int x, int y)* method on this interface.

Variable Index

ID

Event id

Method Index

getDisplay(int, int)

Returns the displayed string starting at coordinates (x, y).

Variables

ID

```
 public static final int ID
```

Event id

Methods

getDisplay

```
 public abstract String getDisplay(int x,
                                   int y)
```

Returns the displayed string starting at coordinates (x, y).

Parameters:

x - The x-coordinate.

y - The y-coordinate.

Returns:

The string displayed starting at coordinates (x, y).

All Packages This Package Previous Next

All Packages   This Package Previous Next

Interface javax.telephony.phone.events.HookswitchStateEv

public interface **HookswitchStateEv**

extends PhoneTermEv

The HookswitchStateEv interface extends the PhoneTermEv interface and is reported via the PhoneTermObserver interface. This event interface indicates that the state of the hookswitch component has changed.

Applications may obtain the new state of the hookswitch (either on-hook or off-hook) via the *getHookSwitchState()* method on this interface.

Variable Index

● **ID**

Event id

Method Index

● **getHookSwitchState**()

Returns the current state of the hookswitch.

Variables

● **ID**

```
public static final int ID
```

Event id

Methods

● **getHookSwitchState**

```
public abstract int getHookSwitchState()
```

Returns the current state of the hookswitch.

Returns:

The current state of the hookswitch.

Interface javax.telephony.phone.events.LampModeEv

public interface **LampModeEv**

extends PhoneTermEv

The LampModeEv interface extends the PhoneTermEv and is reported via the PhoneTerminalObserver interface. This event indicates that the mode of the lamp has changed.

Applications may use the *getMode()* method on this interface to obtain the new mode of the lamp.

Variable Index

- **ID**

  Event id

Method Index

- **getMode**()

  Returns the current lamp mode.

Variables

ID

```
public static final int ID
```

Event id

Methods

getMode

```
public abstract int getMode()
```

Returns the current lamp mode.

Returns:

The current lamp mode.

Interface javax.telephony.phone.events.MicrophoneGainEv

public interface **MicrophoneGainEv**

extends PhoneTermEv

The MicrophoneGainEv interface extends the PhoneTermEv interface and is reported via the PhoneTerminalObserver interface. This event interface indicates that the gain of a microphone component has changed.

Applications may use the *getGain()* method on this interface to obtain the new gain of the microphone component.

Variable Index

- **ID**

  Event id

Method Index

- **getGain**()

  Returns the gain of the microphone.

Variables

ID

```
public static final int ID
```

Event id

Methods

getGain

```
public abstract int getGain()
```

Returns the gain of the microphone.

Returns:

The gain of the microphone.

Interface javax.telephony.phone.events.PhoneEv

public interface **PhoneEv**

extends Ev

The PhoneEv is the base event for all events in the Phone package. Each event in this package must extend this interface. This interface is not meant to be a public interface, it is just a building block for other event interfaces.

The PhoneEv interface contains getPhoneCause(), which returns the reason for the event.

Variable Index

- **CAUSE_NORMAL**

  Cause code indicating normal operation
- **CAUSE_UNKNOWN**

  Cause code indicating the cause was unknown

Method Index

- **getPhoneCause**()

  Returns the phone and core causes associated with this event.

Variables

CAUSE_NORMAL

```
public static final int CAUSE_NORMAL
```

Cause code indicating normal operation

CAUSE_UNKNOWN

```
public static final int CAUSE_UNKNOWN
```

Cause code indicating the cause was unknown

Methods

getPhoneCause

```
public abstract int getPhoneCause()
```

Returns the phone and core causes associated with this event. Every event has a cause. The various cause values are defined as public static final variablies in this interface, with the exception of CAUSE_NORMAL and CAUSE_UNKNOWN, which are defined in the core.

Returns:

s The cause of the event.

All Packages This Package Previous Next

Interface javax.telephony.phone.events.PhoneTermEv

public interface **PhoneTermEv**

extends PhoneEv, TermEv

The PhoneTermEv interface extends the TermEv interface and is the base event interface for all phone-components related events. All component events must extends this interface. These events are reported through the TerminalObserver interface.

Method Index

- **getComponent**()

  Returns the Component object responsible for this event.
- **getComponentGroup**()

  Returns the ComponentGroup object associated with this event.

Methods

getComponentGroup

```
 public abstract ComponentGroup getComponentGroup()
```

Returns the ComponentGroup object associated with this event.

Returns:

s The ComponentGroup object associated with this event.

getComponent

```
 public abstract Component getComponent()
```

Returns the Component object responsible for this event.

Returns:

s The Component object responsible for this event.

Interface javax.telephony.phone.events.RingerPatternEv

public interface **RingerPatternEv**

extends PhoneTermEv

The RingerPatternEv interface extends the PhoneTermEv interface and is reported via the PhoneTerminalObserver interface. This event interface indicates that the pattern of a ringer component has changed.

Applications may use the *getPattern()* method on this interface to obtain the new pattern of the ringer component.

Variable Index

● **ID**

Event id

Method Index

● **getRingerPattern**()

Returns the pattern of the ringer.

Variables

● **ID**

```
 public static final int ID
```

Event id

Methods

● **getRingerPattern**

```
 public abstract int getRingerPattern()
```

Returns the pattern of the ringer.

Returns:

The pattern of the ringer.

Interface javax.telephony.phone.events.RingerVolumeEv

public interface **RingerVolumeEv**
extends PhoneTermEv

The RingerVolumeEv interface extends the PhoneTermEv interface and is reported via the PhoneTerminalObserver interface. This event interface indicates that the volume of a ringer component has changed.

Applications may use the *getVolume()* method on this interface to obtain the new volume of the ringer component.

Variable Index

● **ID**

Event id

Method Index

● **getVolume**()

Returns the volume of the ringer.

Variables

● ID

```
 public static final int ID
```

Event id

Methods

● getVolume

```
 public abstract int getVolume()
```

Returns the volume of the ringer.

Returns:

The volume of the ringer.

All Packages This Package Previous Next

Interface javax.telephony.phone.events.SpeakerVolumeEv

public interface **SpeakerVolumeEv**
extends PhoneTermEv

The SpeakerVolumeEv interface extends the PhoneTermEv interface and is reported via the PhoneTerminalObserver interface. This event interface indicates that the volume of a speaker component has changed.

Applications may use the *getVolume()* method on this interface to obtain the new volume of the speaker component.

Variable Index

ID

Event id

Method Index

getVolume()

Returns the volume of the speaker.

Variables

ID

```
 public static final int ID
```

Event id

Methods

getVolume

```
 public abstract int getVolume()
```

Returns the volume of the speaker.

Returns:

The volume of the speaker.

All Packages

package javax.telephony.privatedata

Interface Index

- PrivateData

Interface javax.telephony.privatedata.PrivateData

public interface **PrivateData**

Introduction

The *private data* mechanism in JTAPI is a means by which applications can send platform-specific messages to the underlying telephone platform. The `PrivateData` interface may be implemented on any JTAPI object. Applications may query an object to see if it supports this interface via the `instanceof` operator. This interface makes no attempt to interpret the data sent to the underlying platform.

Note: Use of this interface interferes with application portability across different JTAPI implementations. Applications which make use of this interface may not work properly with other JTAPI-compliant implementations.

Setting vs. Sending Private Data

There are two ways in which information is sent to the platform. Applications can *set* a piece of data to be associated with the next method invocation on the object. The data is only valid for the next method invocation on the same object. This data is not transmitted to the underlying platform until the next method is invoked. Also, applications may immediately *send* a piece of data to the underlying platform. This data is not associated with any future method invocation.

Private Data Events

Implementations may also send platform-specific events to the application. Each individual object carries its own private data event. The data carried in these objects are specific to the implementation. The private data event interfaces defined are: `PrivateAddrEv`, `PrivateCallEv`, `PrivateProvEv`, and `PrivateTermEv`

See Also:

PrivateDataCapabilities, PrivateAddrEv, PrivateCallEv, PrivateProvEv, PrivateTermEv

Method Index

- **getPrivateData**()

  Returns some platform-specific data associated with the last method that was invoked on the object for which this PrivateData is implemented.
- **sendPrivateData**(Object)

  Immediately performs some platform-specific action.
- **setPrivateData**(Object)

  Associates some platform-specific data with the next method that is invoked on the object for which this interface is implemented.

Methods

setPrivateData

```
 public abstract void setPrivateData(Object data)
```

Associates some platform-specific data with the next method that is invoked on the object for which this interface is implemented. The format of this data and the manner in which it modifies the method invocation is platform-dependent. This data applies to the next method invocation ONLY and does not affect any future method invocations.

Parameters:

data - The platform-dependent data.

getPrivateData

```
public abstract Object getPrivateData()
```

Returns some platform-specific data associated with the last method that was invoked on the object for which this PrivateData is implemented. The format of this data is platform-dependent. This data pertains to the last method invocation ONLY.

Returns:

Object The platform-dependent data.

sendPrivateData

```
public abstract Object sendPrivateData(Object data)
```

Immediately performs some platform-specific action. The effect of this methods invocation is immediate and does not directly relate to any future object method invocations. The action taken upon receipt of this data is platform-dependent as is the format of the data itself. This method returns the platform-dependent data actually sent.

Parameters:

data - The platform-dependent data.

Returns:

The platform-dependent data sent.

All Packages   This Package Previous Next

All Packages

package javax.telephony.privatedata.capabilities

Interface Index

- PrivateDataCapabilities

All Packages   This Package Previous Next

Interface javax.telephony.privatedata.capabilities.PrivateDataCapabilities

public interface **PrivateDataCapabilities**

The `PrivateDataCapabilities` interface is the capabilities interface for the `PrivateData` interface. Additional packages which want to extend the private data package should extend this interface for its capabilities.

Since the `PrivateData` interface is always implemented on some existing JTAPI object (e.g. Provider, Call, etc), this interface should be implemented along with the corresponding object's capabilities interface. For example, if the implementation's Call object supports private data, the `Provider.getCallCapabilities()` and `Call.getCapabilities()` methods should return objects which implement `PrivateDataCapabilities` in addition to the `CallCapabilities` interface.

See Also:

PrivateData

Method Index

- **canGetPrivateData**()

  This method returns true if the `PrivateData.getPrivateData()` method is supported, false otherwise.
- **canSendPrivateData**()

  This method returns true if the `PrivateData.sendPrivateData()` method is supported, false otherwise.
- **canSetPrivateData**()

  This method returns true if the `PrivateData.setPrivateData()` method is supported, false otherwise.

Methods

canSetPrivateData

```
public abstract boolean canSetPrivateData()
```

This method returns true if the `PrivateData.setPrivateData()` method is supported, false otherwise.

Returns:

True if the setting of private data is supported, false otherwise.

canGetPrivateData

```
public abstract boolean canGetPrivateData()
```

This method returns true if the `PrivateData.getPrivateData()` method is supported, false otherwise.

Returns:

True if obtaining the private data is supported, false otherwise.

canSendPrivateData

```
public abstract boolean canSendPrivateData()
```

This method returns true if the `PrivateData.sendPrivateData()` method is supported, false otherwise.

Returns:

True if the sending of private data is supported, false otherwise.

All Packages This Package Previous Next

All Packages

package javax.telephony.privatedata.events

Interface Index

- PrivateAddrEv
- PrivateCallEv
- PrivateProvEv
- PrivateTermEv

Interface javax.telephony.privatedata.events.PrivateAddrEv

public interface **PrivateAddrEv**

extends AddrEv

The `PrivateAddrEv` interface sends platform-specific event information to an `AddressObserver`. This interface extends the core `AddrEv` interface. This interface could be a stand-alone event for private data that is not associated with any other event. This interface could also be used to extend any other event for private data.

When used as a stand-alone event, the ID returned by `Ev.getID()` should be the ID defined in this interface. When used to extend another event to add private data to that event, the ID returned by `Ev.getID()` should be the ID defined in the other event interface.

See Also:

AddrEv, AddressObserver, PrivateData

Variable Index

ID

The Event ID.

Method Index

getPrivateData()

Returns platform-specific information to the application.

Variables

ID

```
public static final int ID
```

The Event ID.

Methods

getPrivateData

```
public abstract Object getPrivateData()
```

Returns platform-specific information to the application. The format of the data and the action that should be taken upon receipt of the data is platform-dependent.

Returns:

The platform-specific data.

All Packages This Package Previous Next

Interface javax.telephony.privatedata.events.PrivateCallEv

public interface **PrivateCallEv**

extends CallEv

The `PrivateCallEv` interface sends platform-specific event information to a `CallObserver`. This interface extends the core `CallEv` interface. This event could be a stand-alone event for private data that is not associated with any other event. This interface could also be used to extend any other event for private data.

When used as a stand-alone event, the ID returned by `Ev.getID()` should be the ID defined in this interface. When used to extend another event to add private data to that event, the ID returned by `Ev.getID()` should be the ID defined in the other event interface.

See Also:

CallEv, CallObserver, PrivateData

Variable Index

- **ID**

  The Event ID.

Method Index

- **getPrivateData**()

  Returns platform-specific information to the application.

Variables

ID

```
 public static final int ID
```

The Event ID.

Methods

getPrivateData

```
 public abstract Object getPrivateData()
```

Returns platform-specific information to the application. The format of the data and the action that should be taken upon receipt of the data is platform-dependent.

Returns:

The platform-specific data.

Interface javax.telephony.privatedata.events.PrivateProvEv

public interface **PrivateProvEv**

extends ProvEv

The `PrivateProvEv` interface sends platform-specific event information to a `ProviderObserver`. This interface extends the core `ProvEv` interface. This event could be a stand-alone event for private data that is not associated with any other event. This interface could also be used to extend any other event for private data.

When used as a stand-alone event, the ID returned by `Ev.getID()` should be the ID defined in this interface. When used to extend another event to add private data to that event, the ID returned by `Ev.getID()` should be the ID defined in the other event interface.

See Also:

ProvEv, ProviderObserver, PrivateData

Variable Index

ID

The Event ID.

Method Index

getPrivateData()

Returns platform-specific information to the application.

Variables

ID

```
public static final int ID
```

The Event ID.

Methods

getPrivateData

```
public abstract Object getPrivateData()
```

Returns platform-specific information to the application. The format of the data and the action that should be taken upon receipt of the data is platform-dependent.

Returns:

The platform-specific data.

All Packages This Package Previous Next

Interface javax.telephony.privatedata.events.PrivateTermEv

public interface **PrivateTermEv**

extends TermEv

The `PrivateTermEv` interface sends platform-specific event information to a `TerminalObserver`. This interface extends the core `TermEv` interface. This event could be a stand-alone event for private data that is not associated with any other event. This interface could also be used to extend any other event for private data.

When used as a stand-alone event, the ID returned by `Ev.getID()` should be the ID defined in this interface. When used to extend another event to add private data to that event, the ID returned by `Ev.getID()` should be the ID defined in the other event interface.

See Also:

TermEv, TerminalObserver, PrivateData

Variable Index

● **ID**

The Event ID.

Method Index

● **getPrivateData**()

Returns platform-specific information to the application.

Variables

● **ID**

```
 public static final int ID
```

The Event ID.

Methods

● **getPrivateData**

```
 public abstract Object getPrivateData()
```

Returns platform-specific information to the application. The format of the data and the action that should be taken upon receipt of the data is platform-dependent.

Returns:

The platform-specific data.

All Packages   This Package Previous Next