

Java™ Call Control v1.0 to H.323 API Mapping

JAIN™ Community

A Fujitsu, NEC, NTT, Sun Microsystems, Telcordia Technologies Document

Date: October 25, 2001

Version: 1.0

Author: Chammika Subasinghe – Fujitsu

Co-Author: Phelim O’Doherty – Sun Microsystems, Inc.

(C) 2001 Sun Microsystems, Inc.

All rights reserved. Sun, Sun Microsystems, the Sun Logo, JAIN, Java and Javadoc are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Abstract

This document has been authored by various companies within the JAIN™ Community as an example mapping from the Java Call Control protocol to the H323 Protocol. Its purpose is to aid potential implementers of the Java Call Control API Specification to wire down the Java Call Control protocol to the H323 protocol. This document should be regarded as an example mapping and not as the standardized way to map from the Java Call Control API Specification to the H323 protocol. The mapping will not detail error behavior, successful behavior is assumed.

The audience is assumed to be knowledgeable of the Java Call Control API and the H323 protocol. This document is not a tutorial on the Java Call Control API or H323.

This document is accompanied by a mapping from the Java Call Control protocol to the Intelligent Network Application Part (INAP) and the Session Initiation Protocol (SIP) these additional documents are available from <http://java.sun.com/products/jain>.

Table Of Contents

ABSTRACT.....	3
1.0 INTRODUCTION.....	5
1.1 BRIEF INTRODUCTION TO JAVA CALL CONTROL.....	5
1.2 BRIEF INTRODUCTION TO H323.....	6
1.3 APPROACH.....	6
2.0 MAPPED METHODS.....	7
2.1 JCCPROVIDER.....	7
2.2 JCPROVIDERLISTENER.....	16
2.3 CALLLOADCONTROLLISTENER.....	19
2.4 JCCCALL.....	21
2.5 JCCCALLLISTENER.....	26
2.6 JCCCONNECTION.....	32
2.7 JCCCONNECTIONLISTENER.....	40
3.0 METHODS NOT MAPPED.....	52
3.1 JCCPROVIDER.....	52
3.2 JCCCALL.....	53
3.3 JCCCONNECTION.....	54
4.0 REFERENCES.....	55
5.0 APPENDIX.....	56
5.1 THE JAVA CALL CONTROL CONNECTION OBJECT FSM.....	56

1.0 Introduction

The Java Call Control API provides a common open interface into any kind of telecommunications network. This document exemplifies a possible mapping from Java Call Control 1.0 to H323. This mapping is descriptive; it does not introduce any implementations constraints or prescribe how the mapping from Java Call Control to the underlying network is to be performed, whatsoever. For reasons of clarity, a mapping from the Java Call Control API to the H323 individual messages is given.

Only the call control aspects of the Java Call Control API are considered in this document. The authentication and discovery aspects of the API are assumed to have completed successfully. Although the API consists of two packages, where the objects in one package inherit from the other, only the methods on objects in the Java Call Control package are mapped.

1.1 *Brief introduction to Java Call Control*

Java Call Control is an abstraction for performing call control and hence is independent of any network. It is up to the implementation of the Java Call Control API to support any network such as SIP, H.323, ISUP, INAP, etc. by performing the appropriate mapping. This document gives a descriptive mapping from the Java Call Control API to H.323.

The Java Call Control API is a Java interface for creating, monitoring, controlling, manipulating and tearing down communications sessions in a converged PSTN, packet-switched, and wireless environment. It provides facilities for first-party as well as third-party applications, and is applicable to network elements (such as switches or Call Agents) both at the network periphery (e.g. Class 5 or end-office switches) and at the core (e.g. Class 4 or tandem switches).

Java Call Control allows applications to be invoked or triggered during session set-up in a manner similar in spirit to the way in which IN or AIN services can be invoked. Java Call Control thus allows programmers to develop applications that can execute on any platform that supports the API, increasing the market for their applications. It also allows service providers to rapidly and efficiently offer services to end users by developing the services themselves, by outsourcing development, purchasing services developed by third parties, or a combination thereof.

The API is not intended to open up telecommunications networks' signaling infrastructure for public usage. Rather, network capabilities are intended to be encapsulated and made visible using object technology in a secure, manageable, and billable manner. This approach allows third party service providers to develop applications supported by the network without compromising network security, integrity, and reliability.

The API is specified in terms of a coherent collection of related and interacting objects that model different physical and logical elements involved in a session, and related functions. Applications interact with these objects via an object-oriented Listener paradigm. Note that the API is applicable to control of voice, data or multimedia sessions, and not just voice calls, but for convenience we often use the word “call” in the specification.

1.2 *Brief introduction to H323*

H.323, is a standard for audio, video and data communication, which describes how multimedia communications occur between terminals, network equipment and services on LANs which do not provide a guaranteed Quality of Service (e.g., IP networks). H.323 itself references numerous other recommendations, together these define new network components which interoperate with other standards-compliant end-points and networks by virtue of an H.323 Gateway. H.323 provides standardization for services provided by the computer industry to create the level of reliability expected in the telecommunications market. H.323 performs the functions necessary to establish and maintain real-time audio/video/data conferencing sessions over IP data networks and provides high-level API for the audio and video streams and the user application.

1.3 *Approach*

Each Java Call Control API method will be mapped to a H323 message a top down approach; hence certain H323 messages may not be mapped. Prior to mapping the method some explanation is given. Further information in each method can be found in the Javadoc™ file packed with the specification.

Figure 1 is a template for the call flow, it will be shown per Java Call Control method (if appropriate); it details the H.323 message(s) that cause a Java Call Control Application method invocation or the Java Call Control Implementation method that causes H.323 message(s) to be sent.

Following the call flow, the normal operation and context is outlined. Next, the Java Call Control method is mapped to H.323 messages and fields (if appropriate).

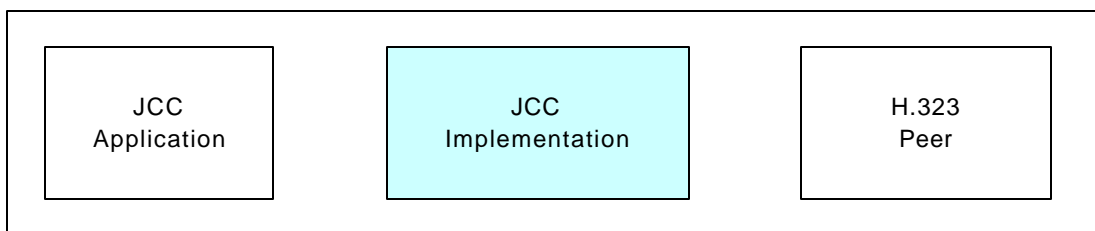


Figure 1 Call flow diagram

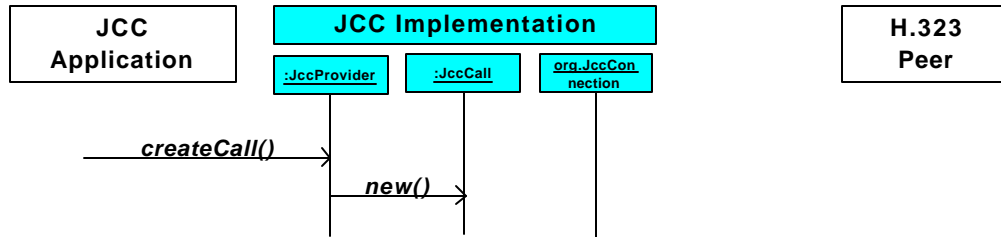
2.0 Mapped methods

2.1 JccProvider

`JcpCall createCall()`

Creates a new instance of the call with no connections.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-condition	The application is instructing the Gateway to initiate a call setup.
1	The application invokes the <i>createCall</i> method.
2	The Gateway creates a new call object.

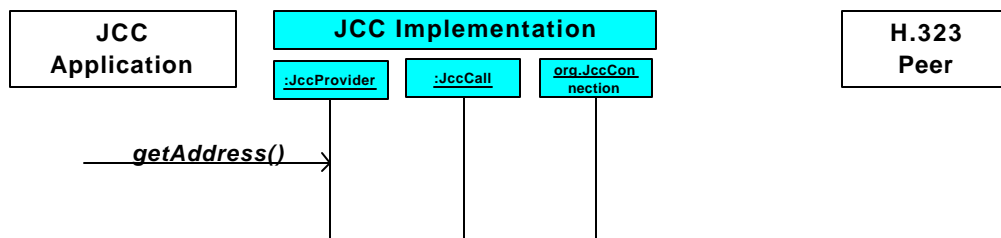
Parameter Mapping

None

`JcpAddress getAddress(String address)`

Returns an `JcpAddress` object which corresponds to the (telephone) number string provided.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The JccAddress object corresponding to the (telephone) number string provided must be instantiated.
1	The Application invokes the <i>getAddress</i> method.
2.	The Java Call Control Implementation returns an address object corresponding to the given number.

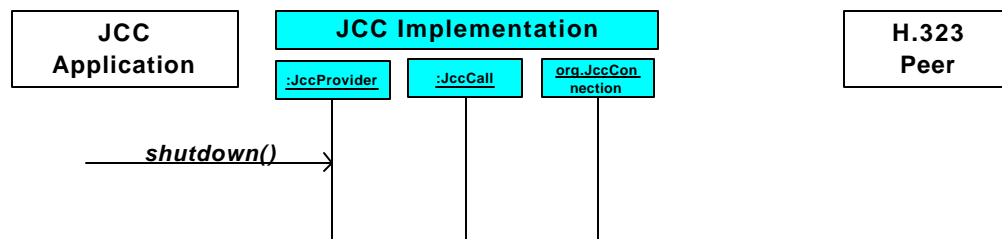
Parameter Mapping

None

```
void shutdown()
```

Instructs the JccProvider to shut itself down and provide all necessary cleanup.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None
1	The Application invokes the <i>shutdown</i> method.
2.	The Java Call Control Implementation instructs JccProvider to shutdown itself.

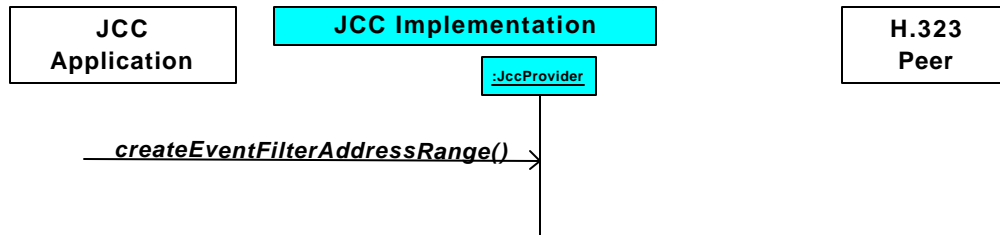
Parameter Mapping

None

```
EventFilter createEventFilterAddressRange(String lowAddress, String highAddress, int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1.	The Application invokes the <i>createEventFilterAddressRange</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

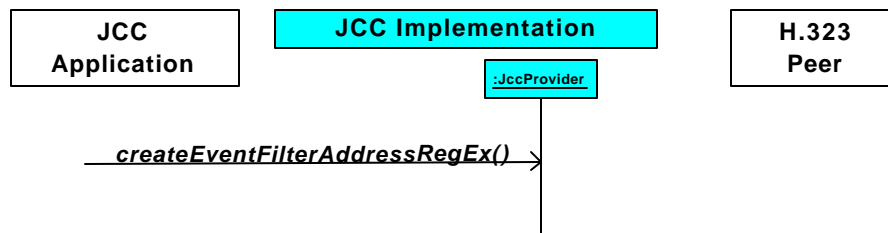
Parameter Mapping

None

```
EventFilter createEventFilterAddressRegEx(String addressRegex, int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterAddressRegEx</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

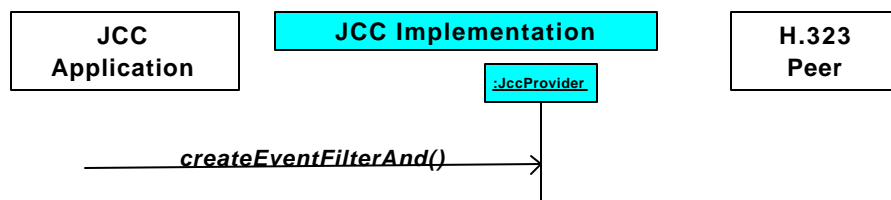
Parameter Mapping

None

```
EventFilter createEventFilterAnd(EventFilter[] filters, int  
nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None
1	The Application invokes the <i>createEventFilterAnd</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

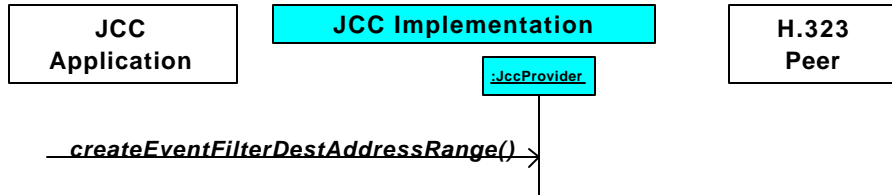
Parameter Mapping

None

```
EventFilter createEventFilterDestAddressRange(String lowDestAddress,
String highDestAddress, int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1.	The Application invokes the <i>createEventFilterDestAddressRange</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

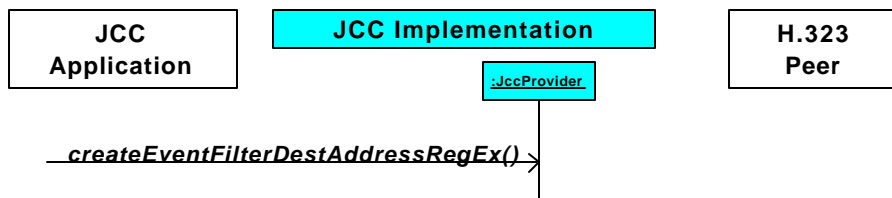
Parameter Mapping

None

```
EventFilter createEventFilterDestAddressRegEx(String destAddressRegex,
int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterDestAddressRegEx</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

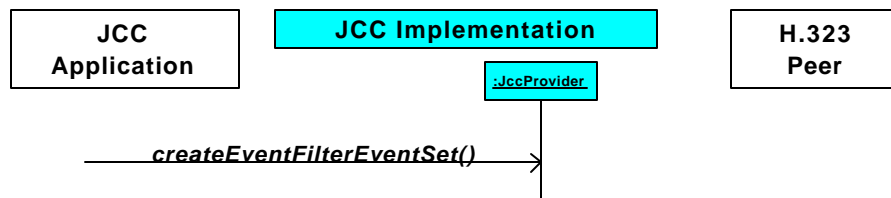
Parameter Mapping

None

```
EventFilter createEventFilterEventSet(int[] blockEvents, int[]  
notifyEvents)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterEventSet</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

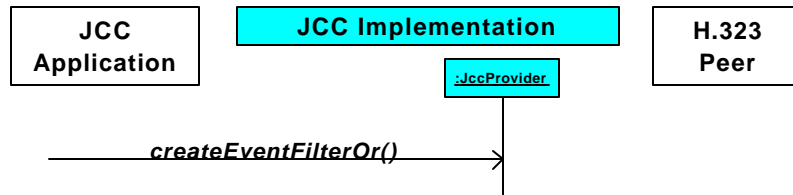
Parameter Mapping

None

```
EventFilter createEventFilterOr(EventFilter[] filters, int  
nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterOr</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

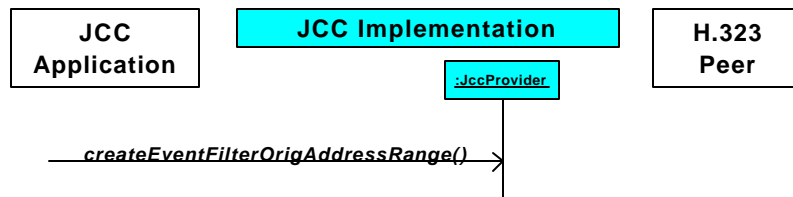
Parameter Mapping

None

```
EventFilter createEventFilterOrigAddressRange(String lowOrigAddress,  
String highOrigAddress, int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterOrigAddressRange</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

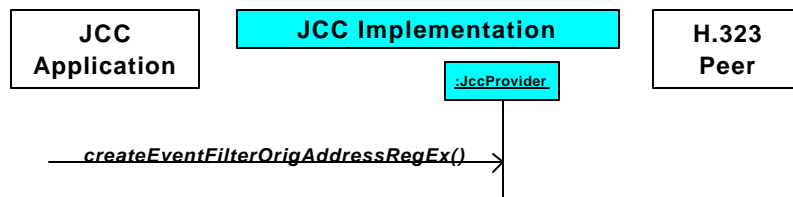
Parameter Mapping

None

```
EventFilter createEventFilterOrigAddressRegEx(String origAddressRegex,  
int matchDisposition, int nomatchDisposition)
```

This method returns a standard EventFilter, which is implemented by the Java Call Control platform.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None.
1	The Application invokes the <i>createEventFilterOrigAddressRegEx</i> method.
2.	The Java Call Control Implementation creates EventFilter object according to the arguments, and returns it.

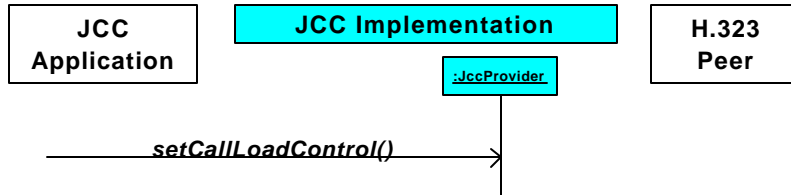
Parameter Mapping

None

```
void setCallLoadControl(JccAddress[] address, double duration, double[] mechanism, int[] treatment)
```

This method imposes or removes load control on calls made to the specified addresses.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	None
1	The Application invokes the <i>setCallLoadControl</i> method.
2.	The Java Call Control Implementation imposes or removes load control on calls corresponding to the specified address.

Parameter Mapping

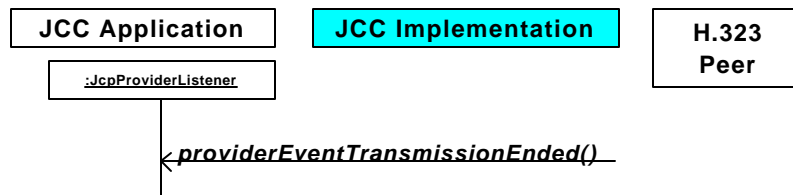
None

2.2 JcpProviderListener

```
void providerEventTransmissionEnded(JcpProviderEvent providerevent)
```

Indicates that the application will no longer receive JcpProvider events on the instance of the JcpProviderListener.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for the Provider has been registered, and the Application invokes the JcpProvider.removeProviderListener method to stop receiving the events related to the Provider.
1	The Java Call Control Implementation invokes the <i>providerEventTransmissionEnded</i> method.
2.	The Java Call Control Implementation removes the registration of JcpProviderListener.

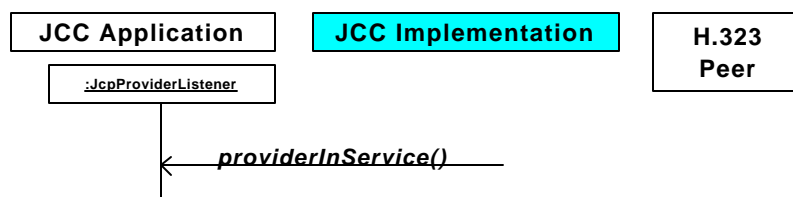
Parameter Mapping

None

```
void providerInService(JcpProviderEvent providerevent)
```

Indicates that the state of the JcpProvider has changed to JcpProvider.IN_SERVICE.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for the Provider has been registered. The Java Call Control API is ready to accept call control requests from the applications.
1.	The Java Call Control API Implementation detects that the state of Provider object has been transited to SERVICE.
2.	The Java Call Control API Implementation invokes the <i>providerInService</i> method.

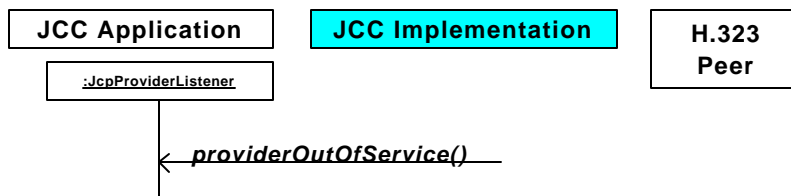
Parameter Mapping

None

```
void providerOutOfService(JcpProviderEvent providerevent)
```

Indicates that the state of the JcpProvider has changed to JcpProvider.OUT_OF_SERVICE.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for the Provider has been registered.
1.	The Java Call Control Implementation detects that the state of Provider object has been transited to OUT_OF_SERVICE.
2.	The Java Call Control Implementation invokes the <i>providerOutOfService</i> method.

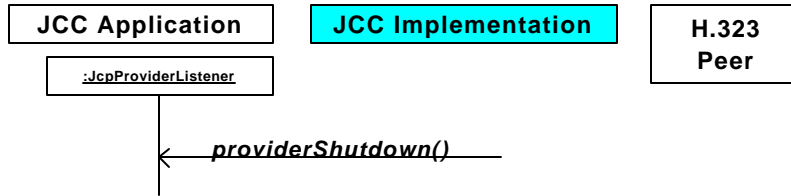
Parameter Mapping

None

```
void providerShutdown(JcpProviderEvent providerevent)
```

Indicates that the state of the JcpProvider has changed to JcpProvider.SHUTDOWN.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for the Provider has been registered, and the API implementation moves to the SHUTDOWN state.
1	The Java Call Control Implementation detects that the state of the Provider has transited to SHUTDOWN.
2.	The Java Call Control Implementation invokes the <i>providerShutdown</i> method

Parameter Mapping

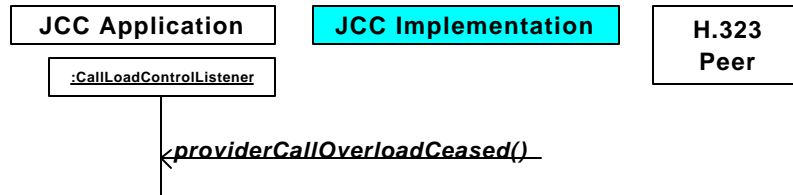
None

2.3 CallLoadControlListener

```
void providerCallOverloadCeased(CallLoadControlEvent loadcontrolevent)
```

This method indicates that the network has detected that the overload has ceased and has automatically removed load control on calls requested to a particular address range or calls made to a particular destination.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for load control events has been registered, and load control functionality has been available.
1	The Java Call Control Implementation detects that the overload has ceased and has automatically removed load control on calls requested to a particular address range.
2.	The Java Call Control Implementation invokes the <i>providerCallOverloadCeased</i> method

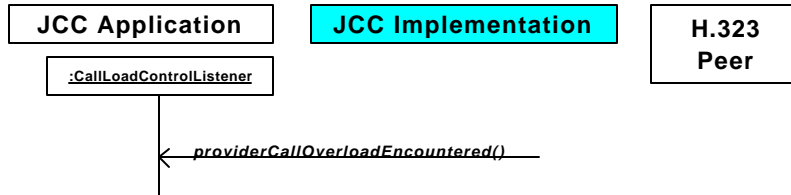
Parameter Mapping

None

```
void providerCallOverloadEncountered(CallLoadControlEvent
loadcontrolevent)
```

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination.

Call Flow



Note: There are no associated H.323 messages.

Normal Operation

Pre-conditions	The listener object for load control events has been registered, and load control functionality has been available.
1	The Java Call Control Implementation detects that overload and may have automatically imposed load control on calls requested to a particular address range.
2.	The Java Call Control Implementation invokes the <i>providerCallOverloadEncountered</i> method

Parameter Mapping

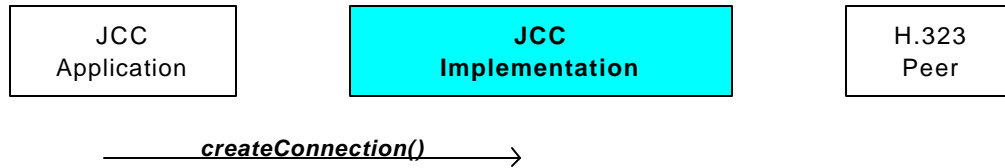
None

2.4 JccCall

```
JccConnection createConnection(String targetAddress, String  
originatingAddress, String originalCalledAddress, String  
redirectingAddress)
```

Creates (a) new JccConnection(s) and attaches it/them to this JccCall.

Call Flow



Note: There are no associated H.323 call flows

Normal Operation

Pre-conditions	Call object has been created.
1	The Application invokes the <i>createConnection</i> method.
2.	The Java Call Control Implementation creates (a) new connection(s) corresponding to the string(s) given as (an) address parameter(s), and attaches it/them to the call.

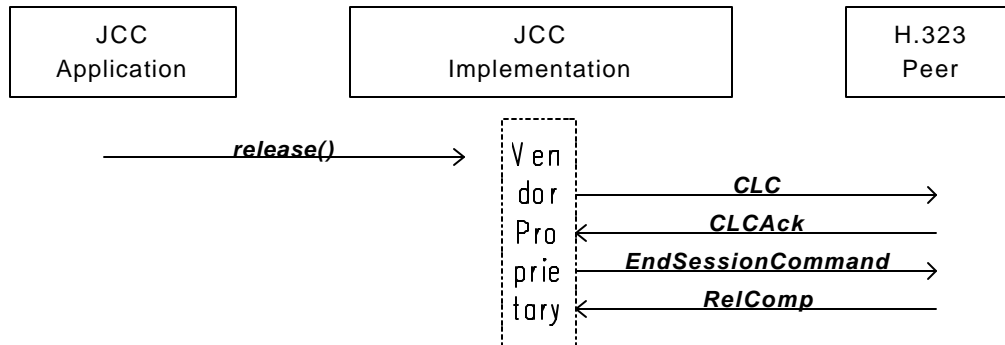
Parameter Mapping

None

void release()

This method requests the release of the call object and associated connection objects.

Call Flow



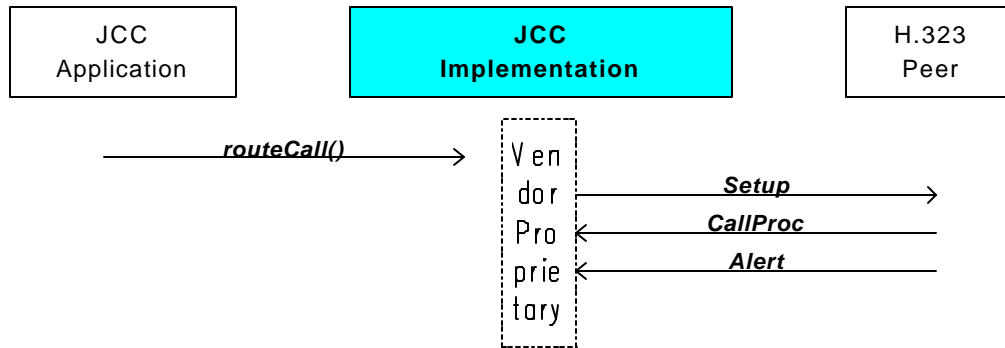
Normal Operation

Pre-conditions	Call in progress.
1	The Application invokes the <i>release</i> method.
2	If terminating party has not answered, the Java Call Control Implementation sends CLC request and EndSessionCommand to terminating party.
3	If all connections are released, the Java Call Control Implementation releases this call.

```
JccConnection routeCall(String targetAddress, String
originatingAddress, String originalDestinationAddress, String
redirectingAddress)
```

This method requests routing of a call to the targetAddress given as an input parameter.

Call Flow



Normal Operation

1. Third party call setup (The targetAddress and originatingAddress are specified, and the targetAddress is equal to the originatingAddress.)

Pre-conditions	Call object has been created, and does not have any connections of which state is not DISCONNECTED.
1	The Application invokes the <i>routeCall</i> method.
2.	The Java Call Control Implementation creates a new JccConnection object corresponding to the targetAddress parameter, and attaches it to call object.
3	The Java Call Control Implementation sends Setup message to the destination related with the targetAddress parameter.

2. Third party call setup (The targetAddress is different from the originatingAddress, and call does not have any connection.)

Pre-conditions	Call object has been created, and does not have any connection of which state is not DISCONNECTED.
1	The Application invokes the <i>routeCall</i> method.
2.	The Java Call Control Implementation creates a new JccConnection object corresponding to the originatingAddress parameter, and attaches it to call object.
3	The Java Call Control Implementation sends Setup message to the

	destination related with the originatingAddress parameter.
4	The Java Call Control Implementation creates a new JccConnection object corresponding to the targetAddress parameter, and attaches it to call object.
5	The Java Call Control Implementation sends Setup message to the destination related with the targetAddress parameter.

3. Third party call setup (The targetAddress is different from the originatingAddress, and call has any connections.)

Pre-conditions	Call object has been created. Connection object(s) have been created.
1	The Application invokes the <i>routeCall</i> method.
2.	The Java Call Control Implementation creates a new JccConnection object corresponding to the targetAddress parameter, and attaches it to call object. Then, the Java Call Control Implementation sends Setup message to the destination related with the targetAddress parameter.

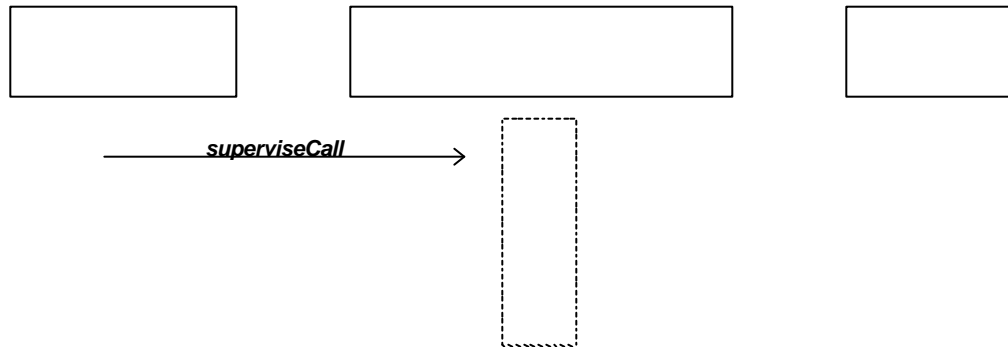
4. Number translation of incoming call.

Pre-conditions	Call object has been created. Connection objects have been created.
1	The Application invokes the <i>routeCall</i> method.
2.	The Java Call Control Implementation creates a new JccConnection object corresponding to the targetAddress parameter, and attaches it to call object. Then, the Java Call Control Implementation sends Setup message to the destination related with the targetAddress parameter.

```
void superviseCall(JccCallListener calllistener, double time, int
treatment, double bytes)
```

The application calls this method to supervise a call.

Call Flow



Note: There are no associated H.323 call flows

Normal Operation

Pre-conditions	The application has been notified of a new call.
1	The Application invokes the <i>supreviseCall</i> method.
2.	The Java Call Control Implementation registers a reference of the listener object to notify the supervision events. Then, it starts a time based or a volume based supervision. If an application calls this function before calling a routeCall method, the time based supervision will start as soon as the call is answered by the called party.

Parameter Mapping

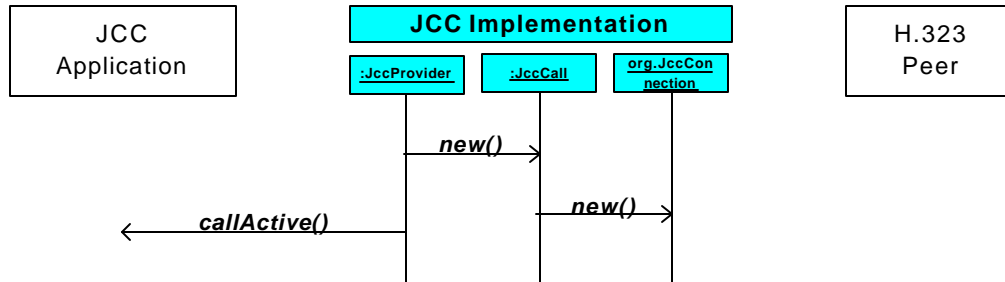
None

2.5 JccCallListener

```
void callActive(JcpCallEvent callevent)
```

Indicates that the state of the JcpCall object has changed to JcpCall.ACTIVE.

Call Flow



Note: The state of the call object changes to ACTIVE when there is minimum of one connection in the call that is not in the DISCONNECTED state.

Normal Operation

Pre-conditions	<p>The listener object for the call has been registered.</p> <p>The call object has been created, and does not have any connections of which state is not DISCONNECTED.</p> <p>The Java Call Control Implementation will create a new connection. (The application will invoke JccCall.createConnection / JccCall.routeCall method, or The Java Call Control Implementation has been received a new INVITE message and has not created a connection yet.)</p>
1	The Java Call Control Implementation creates a new connection object.
2	The state of the call object has changed to ACTIVE.
3	The Java Call Control Implementation invokes the <i>callActive</i> method

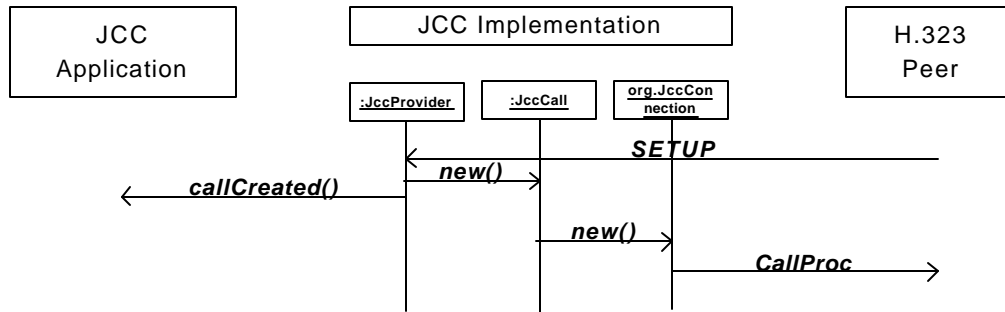
Parameter Mapping

None

```
void callCreated(JcpCallEvent callevent)
```

Indicates that the state of the JcpCall object has changed to JcpCall.IDLE.

Call Flow



Normal Operation:

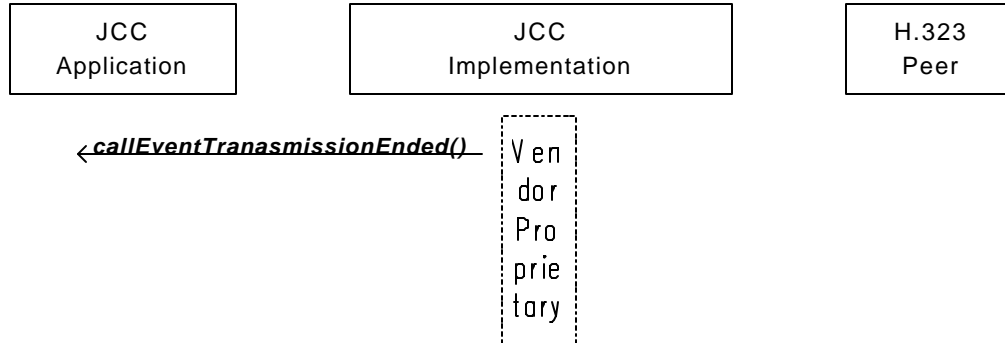
In case of incoming call.

Pre-conditions	The listener object for the call has been registered.
1	The Java Call Control Implementation receives an SETUP message which is sent from within a new session.
2	The Java Call Control Implementation creates a call object.
3	The Java Call Control Implementation invokes the callCreated method

```
void callEventTransmissionEnded(JcpCallEvent callevent)
```

This method is called to indicate that the application will no longer receive JcpCallEvent events on the instance of the JcpCallListener.

Call Flow



Note: This is not associated with any particular H.323 message. The timing is decided by the platform based on the information provided to it.

Normal Operation

Pre-conditions	The listener object for the call has been registered, and the Application invokes the JccCall.removeCallListener or JccProvider.removeCallListener method to stop receiving the events related to a specified call or all calls.
1	The Java Call Control Implementation invokes the <i>callEventTransmissionEnded</i> method.
2	The Java Call Control Implementation removes the registration of JccCallListener object.

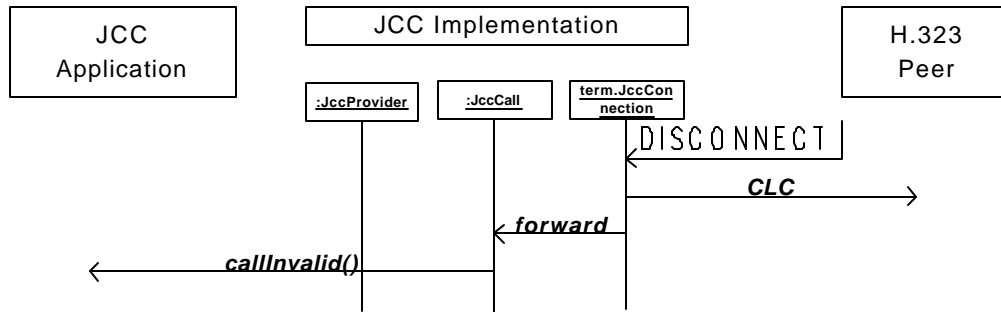
Parameter Mapping

None

```
void callInvalid(JcpCallEvent callevent)
```

Indicates that the state of the JcpCall object has changed to JcpCall.INVALID.

Call Flow



Note: `callInvalid()` is transmitted all the connections related to the call has moved to DISCONNECTED state.

Normal Operation

Pre-conditions	The listener object for the call has been registered. All connection objects on this call will be deleted.
1	The Java Call Control Implementation deletes all connections on this call.
2	The Java Call Control Implementation invokes the <i>callInvalid</i> method

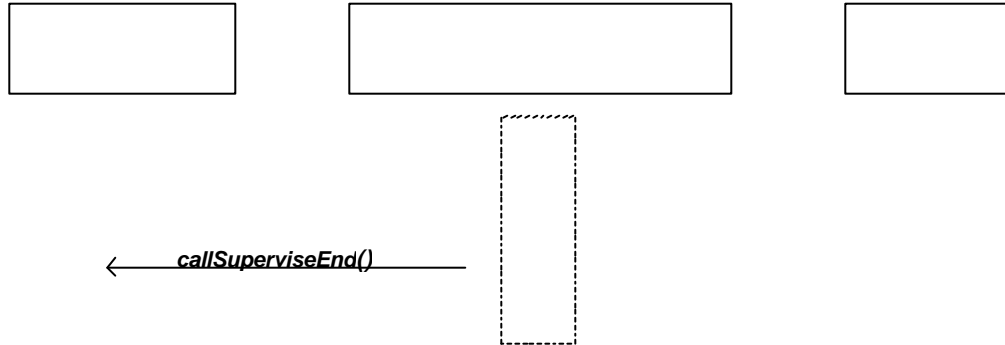
Parameter Mapping

None

```
void callSuperviseEnd(JccCallEvent callevent)
```

Indicates that the supervision of the call has ended.

Call Flow



Note: There are no associated H.323 call flows

Normal Operation

Pre-conditions	The Application has started the supervision of a call, and the supervision will end soon.
1	The Java Call Control Implementation detects the end of the supervision of a call.
2	The Java Call Control Implementation invokes the <i>callSuperviseEnd</i> method

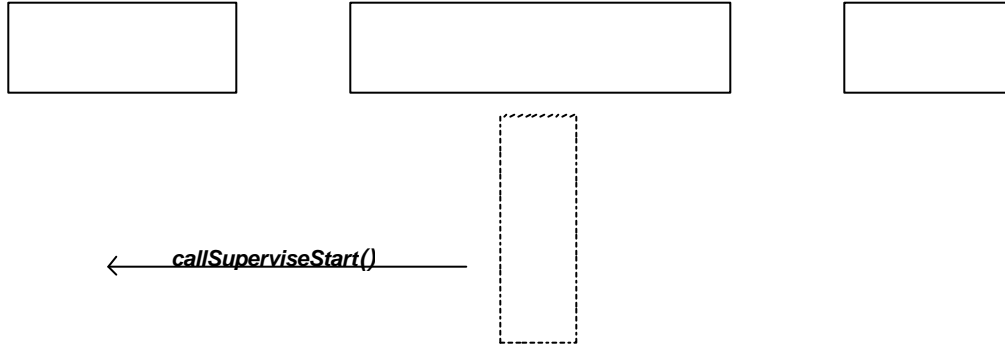
Parameter Mapping

None

```
void callSuperviseStart(JccCallEvent callevent)
```

Indicates that the supervision of the call has started.

Call Flow



Normal Operation

Pre-conditions	The Application will start the supervision of a call using <code>JccCall.superviseCall</code> method.
1	The Java Call Control Implementation starts the supervision of a call.
2	The Java Call Control Implementation invokes the <i><code>callSuperviseStart</code></i> method.

Parameter Mapping

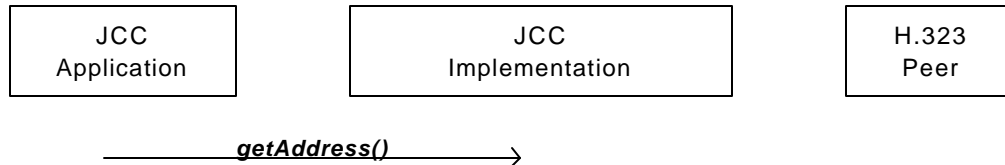
None

2.6 JccConnection

```
JcpAddress getAddress()
```

Returns the JcpAddress associated with this JcpConnection.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	Call and connection objects have been created.
1	The Application invokes the <i>getAddress</i> method.
2.	The Java Call Control Implementation returns the address associated with the connection.

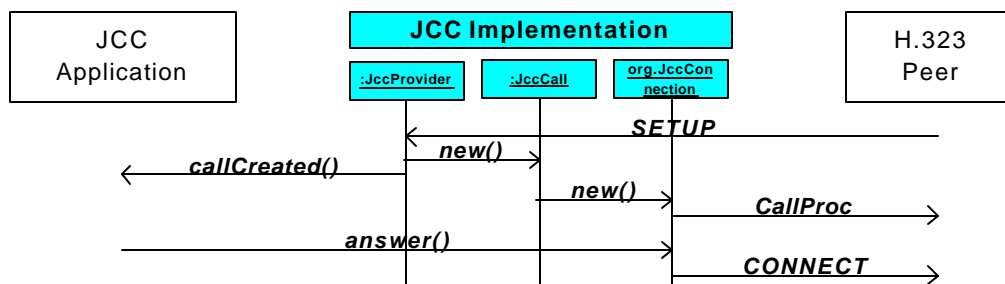
Parameter Mapping

None

```
void answer()
```

This method causes the call to be answered.

Call Flow



Normal Operation

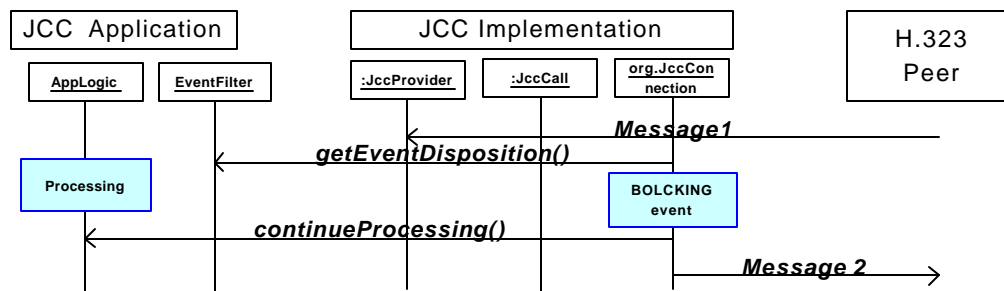
The application invokes the method to be enabled.

Pre-conditions	The Java Call Control Implementation has received SETUP message from H.323 Peer, and call does not answer yet.
1.	The Application invokes the <i>answer</i> method.
2.	The Java Call Control Implementation instructs the call to answer with a CONNECT.

```
void continueProcessing()
```

This method requests the platform to continue processing.

Call Flow



Note: This method could be invoked after any message depending on the application requirements. Once the application has completed what ever the processing that needs to be done, it will inform the platform to continue with the call processing.

Normal Operation

1. The application invokes the method to be enabled.

Pre-conditions	The Java Call Control Implementation has suspended the call processing due to the firing of a blocking event (trigger)
1.	The Application invokes the <i>continueProcessing</i> method.
2.	The Java Call Control Implementation continues the call processing.

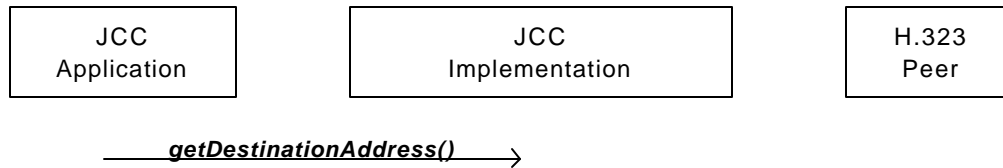
Parameter Mapping

None

String `getDestinationAddress()`

Returns the address of the destination party.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	Call and connection objects have been created.
1	The Application invokes the <i>getDestinationAddress</i> method.
2.	When invoked on an originating connection, The Java Call Control Implementation returns the address of the destination party. Otherwise, it returns null.

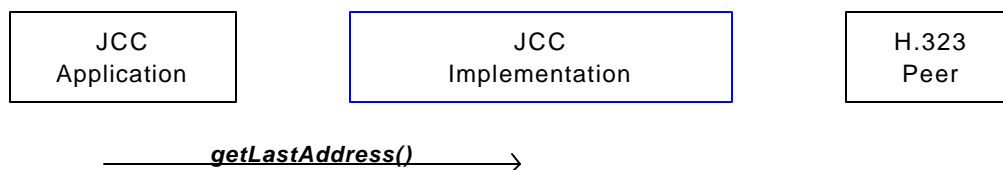
Parameter Mapping

None

String `getLastAddress()`

Returns the last redirected address associated with this `JccConnection`.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	Call and connection objects have been created.
1	The Application invokes the <i>getLastAddress</i> method on originating connection.
2.	The Java Call Control Implementation returns the last redirected address associated with the connection.

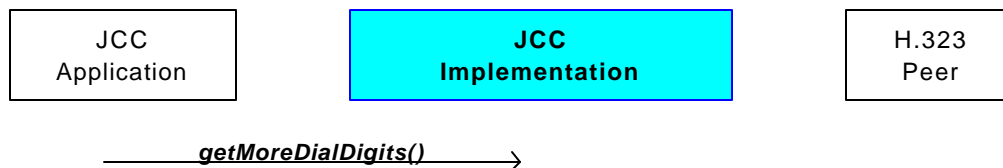
Parameter Mapping

None

```
String getMoreDialedDigits()
```

This method is used by the application to instruct the platform to collect further address information (which may be in the form of digits or letters) and return this to the application.

Call Flow



Note: This is not associated with any particular H.323 message. Also not sure at this time whether this method could be used with H.323.

Normal Operation

Pre-conditions	The state of connection object is ADDRESS_COLLECT or ADDRESS_ANALYZE.
1	The Application invokes the <i>getMoreDialedDigits</i> method.
2.	The Java Call Control Implementation starts to collect further address information and returns this to the application

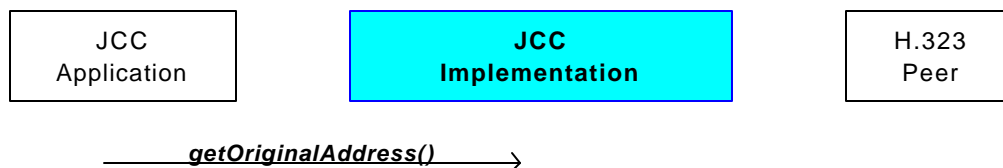
Parameter Mapping

None

```
String getOriginalAddress()
```

Returns the original address associated with this JccConnection.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	Call and connection objects have been created.
1	The Application invokes the <i>getOriginalAddress</i> method.
2.	The Java Call Control Implementation returns the address which was called initially.

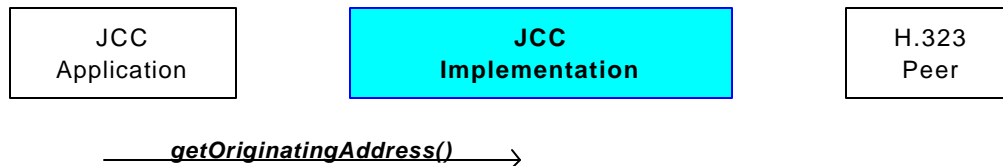
Parameter Mapping

None

```
JccAddress getOriginatingAddress()
```

Returns the address of the originating party.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	Call and connection objects have been created.
1	The Application invokes the <i>getOriginatingAddress</i> method.
2.	When invoked on a terminating connection, The Java Call Control Implementation returns the address of the originating party. When invoked on an originating connection, The Java Call Control Implementation returns the address of which value is equal the return value of <code>JcpConnection.getAddress()</code> . If there is no originating party, The Java Call Control Implementation returns null.

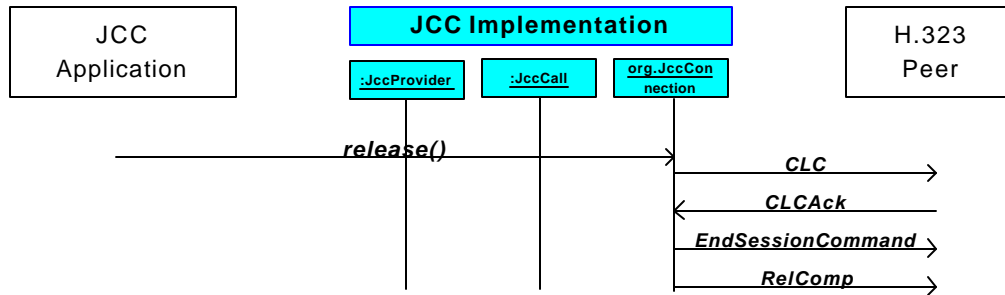
Parameter Mapping

None

void release()

Drops a JccConnection from an active telephone call.

Call Flow



Normal Operation

1. In case of incoming call.

Pre-conditions	Call in progress.
1	The Application invokes the <i>release</i> method.
2	If terminating party has not answered, The Java Call Control Implementation sends CLC and EndSessionCommand to terminating party.

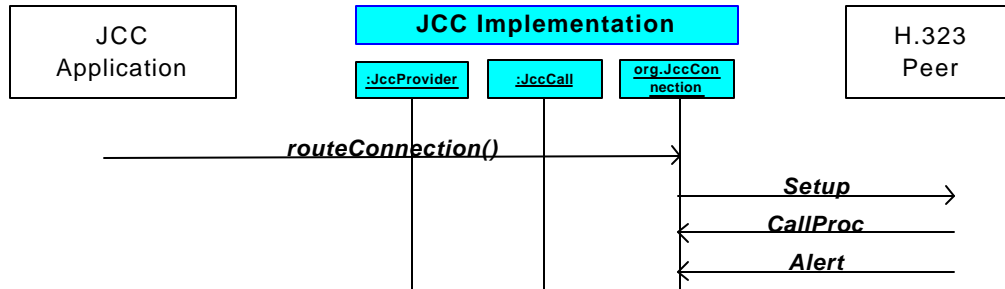
2. In case of third party call.

Pre-conditions	Call in progress.
1	The Application invokes the <i>release</i> method.
2	According to the connection invoked this method, The Java Call Control Implementation sends CLC and EndSessionCommand to terminating party.

```
void routeConnection(boolean attachmedia)
```

Routes this JccConnection to the target address associated with this JccConnection object.

Call Flow



Normal Operation

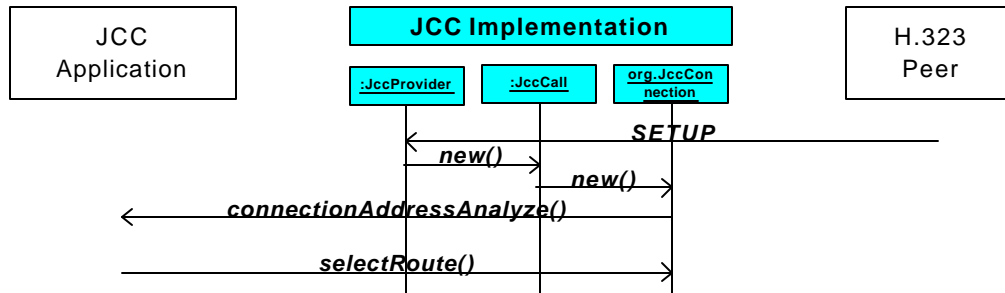
1. The application invokes the method to be enabled.

Pre-conditions	The state of the connection objects is IDLE or AUTHORIZE_CALL_ATTEMPT.
1	The Application invokes the <i>routeConnection</i> method.
2	The Java Call Control Implementation sends SETUP message to the destination corresponding to the address associated with this connection object. If <i>attachmedia</i> argument is set to 'TRUE', The Java Call Control Implementation attaches the media after the connection is routed. If <i>attachmedia</i> argument is set to 'FALSE', The Java Call Control Implementation does nothing.

```
void selectRoute(String address)
```

Replaces address information onto an existing JccConnection.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

1. The application invokes the method to be enabled.

Pre-conditions	The state of connection object is ADDRESS_COLLECT or ADDRESS_ANALYZE.
1	The Application invokes the <i>selectRoute</i> method.
2.	The Java Call Control Implementation replaces the address information with new address specified by this method.
3	The Java Call Control Implementation sends SETUP message to the destination corresponding the target address.

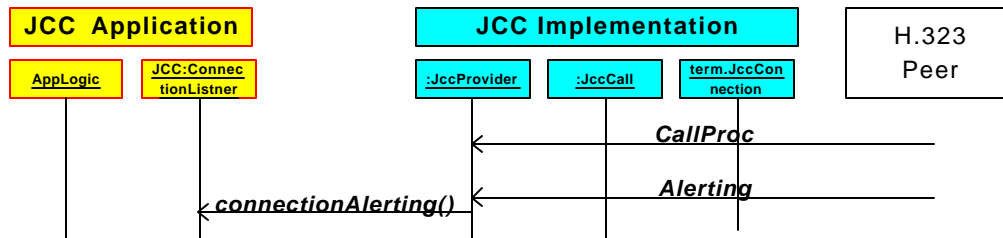
2.7 JccConnectionListener

Note: JccConnectionListener extends the JccCall interface. The methods found in this interface are already described in a previous section. They will not be repeated here.

```
void connectionAlerting(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.ALERTING state

Call Flow



Normal Operation

1. In case of third party call (originating party setup)

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the originating party.
1	The Java Call Control Implementation receives ALERTING message.
2	The state of connection object transits to ALERTING.
3	The Java Call Control Implementation invokes the <i>connectionAlerting</i> method.

2. In case of third party call (terminating party setup)

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the terminating party.
1	The Java Call Control Implementation receives ALERTING message.
2	The state of connection object transits to ALERTING.
3	The Java Call Control Implementation invokes the <i>connectionAlerting</i> method.

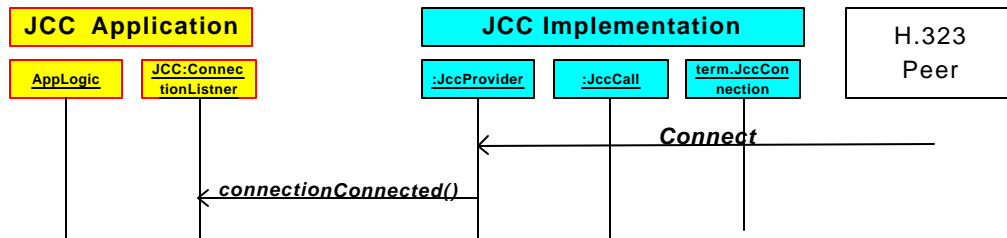
3. In case of incoming call.

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the terminating party.
1	The Java Call Control Implementation receives ALERTING message.
2	The state of the terminating connection transits to ALERTING.
3	The Java Call Control Implementation invokes the <i>connectionAlerting</i> method.
4	The state of originating connection transits to ALERTING.
5	The Java Call Control Implementation invokes the <i>connectionAlerting</i> method.

```
void connectionConnected(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.CONNECTED state

Call Flow



Normal Operation

1. In case of third party call (originating party setup)

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the originating party.
1	The Java Call Control Implementation receives CONNECT message.
2	The state of connection object transits to CONNECTED.
3	The Java Call Control Implementation invokes the <i>connectionConnected</i> method.

2. In case of third party call (terminating party setup)

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the terminating party.
1	The Java Call Control Implementation receives CONNECT message.
2	The state of connection object transits to CONNECTED.
3	The Java Call Control Implementation invokes the <i>connectionConnected</i> method.

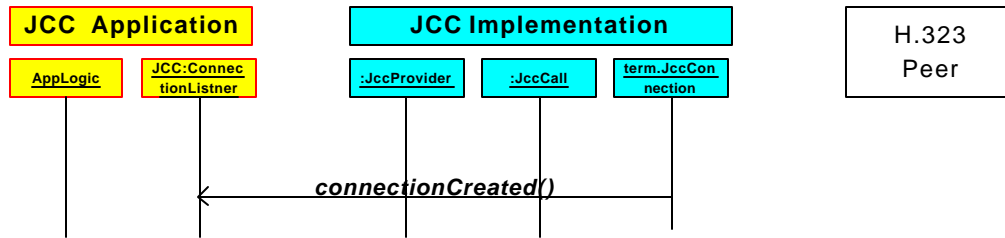
3. In case of incoming call.

Pre-conditions	The listener object for the connection has been registered. The Java Call Control Implementation has sent SETUP message to the destination corresponding to the terminating party.
1	The Java Call Control Implementation receives CONNECT message.
2	The state of the terminating connection object transits to CONNECTED.
3	The Java Call Control Implementation invokes the <i>connectionConnected</i> method.
4	The state of the originating connection object transits to CONNECTED.
5	The Java Call Control Implementation invokes the <i>connectionConnected</i> method

```
void connectionCreated(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection object has just been created.

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

1. In case of third party call.

Pre-conditions	The listener object for the connection has been registered. The call object has been created by using <code>JccProvider.createCall</code> method.
1	The Application invokes the method which creates a new connection. (Ex: The application invokes <code>JccCall.createConnection</code> methods.)
2	The Java Call Control Implementation detects the creation of a new connection object
3	The state of the connection object transits to IDLE.
4	The Java Call Control Implementation invokes the <i><code>connectionCreated</code></i> method

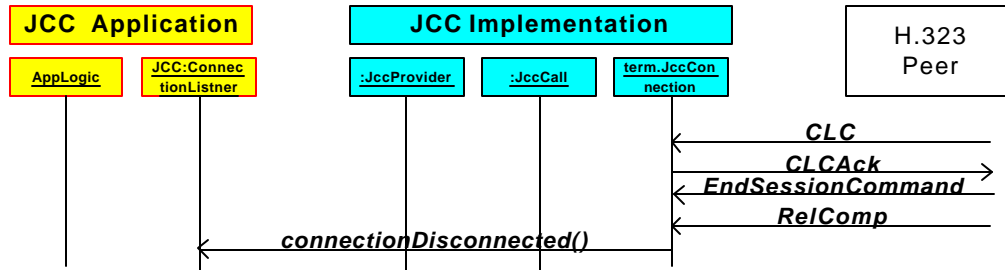
Parameter Mapping

None

```
void connectionDisconnected(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.DISCONNECTED state

Call Flow



Normal Operation

1. The Application releases an existing connection.

Pre-conditions	The listener object for the connection has been registered. Call and connection objects have been created.
1	The Application invokes the method which releases an existing connection. (Ex: The Application invokes the JccConnection.release method.)
2	The state of connection object transits to DISCONNECTED.
3	The Java Call Control Implementation invokes the <i>connectionDisconnected</i> method.

Parameter Mapping

None.

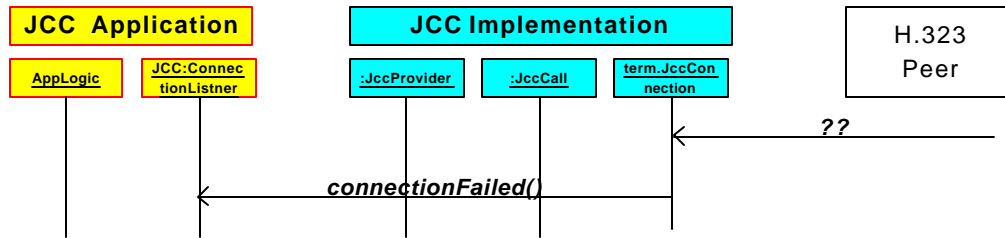
2. Receive EndSessionCommand message.

Pre-conditions	The listener object for the connection has been registered. The state of connection object is CONNECTED.
1	The Java Call Control Implementation receives CLC and EndSessionCommand message.
2	The state of originating and terminating connection object transits to DISCONNECTED.
3	The Java Call Control Implementation invokes the <i>connectionDisconnected</i> method on each connection listener.

```
void connectionFailed(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.FAILED state

Call Flow

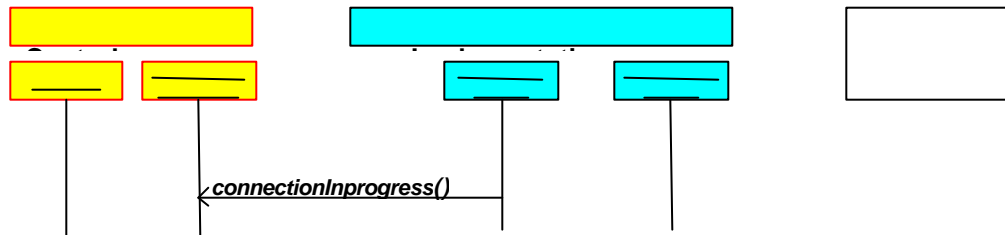


Note: This method could be called due to a failure reported back form the terminal (The message is TBD) or due to en error found in the logic itself. For example wrong address, etc.

```
void connectionInProgress(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.INPROGRESS state

Call Flow



Note: The connection state change is not directly related to any H.323 message. This would occur due to internal processing of the connection.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created, and the state of connection object is IDLE.
1	The Java Call Control Implementation starts to process the connection.
2	The state of connection object transits from IDLE to INPROGRESS.
3	The Java Call Control Implementation invokes the <i>connectionInProgress</i> method.

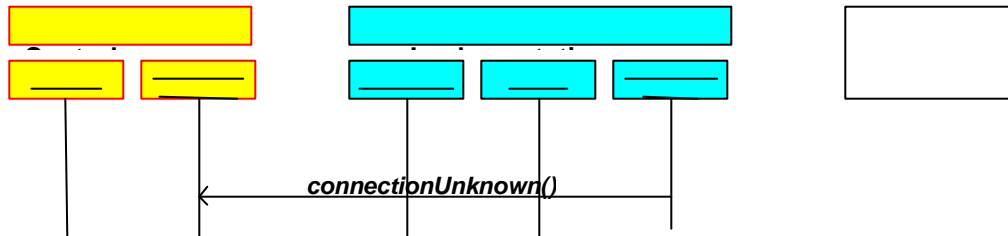
Parameter Mapping

None

```
void connectionUnknown(JcpConnectionEvent connectionevent)
```

Indicates that the JcpConnection has just been placed in the JcpConnection.UNKNOWN state

Call Flow



Note: At time of writing, it is not sure what message form the H.323 protocol will cause this method to be called. It is presumed that some H.323 message that could not be mapped in to a state defined by the spec would cause this.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection objects have been created.
1	The Java Call Control Implementation becomes to be unable to determine the current state of the connection.
2.	The Application invokes the <i>connectionUnknown</i> method.

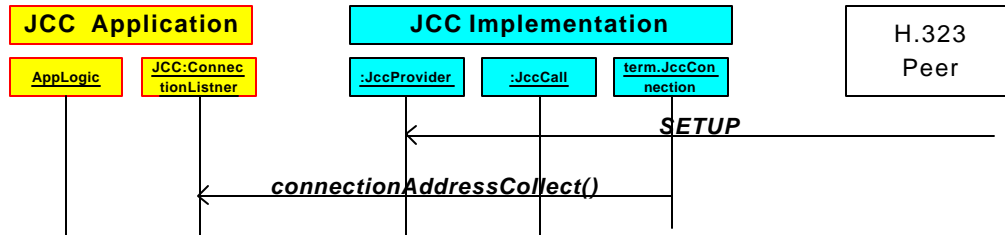
Parameter Mapping

None

```
void connectionAddressAnalyze(JccConnectionEvent connectionevent)
```

Indicates that the JccConnection has just been placed in the JccConnection.ADDRESS_ANALYZE state

Call Flow



Note: The SETUP message would not directly cause the connection to go to ADDRESS_ANALYZE state. The internal processing on the connection after receiving the SETUP will cause the connection to change the state.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created, and the state of connection object is AUTHORIZE_CALL_ATTEMPT or ADDRESS_COLLECT.
1	The Java Call Control Implementation collects complete initial information package/dialing string from the originating party.
2	The state of connection object transits to ADDRESS_ANALYZE.
3	The Java Call Control Implementation invokes the <i>connectionAddressAnalyze</i> method.

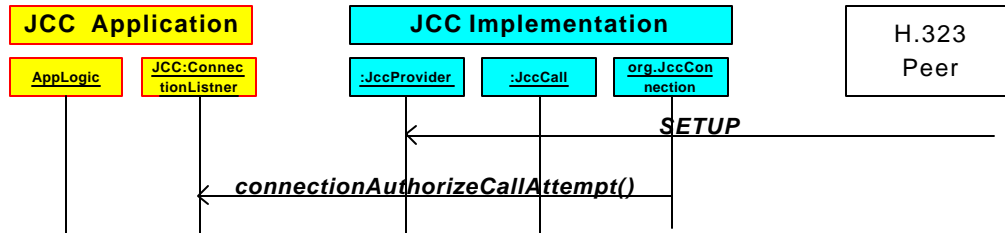
Parameter Mapping

None

```
void connectionAddressCollect(JccConnectionEvent connectionevent)
```

Indicates that the JccConnection has just been placed in the JccConnection.ADDRESS_COLLECT state

Call Flow



Note: The SETUP message would not directly cause the connection to go to ADDRESS_COLLECT state. The internal processing on the connection after receiving the SETUP will cause the connection to change the state.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created, and the state of connection object is AUTHORIZE_CALL_ATTEMPT or CALL_DELIVERY.
1	The Java Call Control Implementation authorizes the originating address for this call.
2	The state of connection object transits to ADDRESS_COLLECT.
3	The Java Call Control Implementation invokes the <i>connectionAddressCollect</i> method.

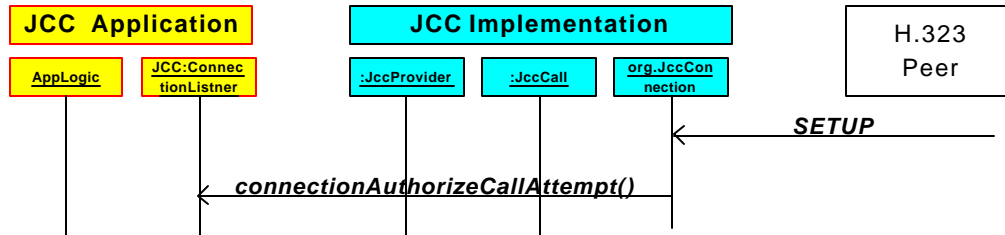
Parameter Mapping

None

```
void connectionAuthorizeCallAttempt(JccConnectionEvent connectionevent)
```

Indicates that the JccConnection has just been placed in the JccConnection.AUTHORIZE_CALL_ATTEMPT state

Call Flow



Note: The SETUP message would not directly cause the connection to go to AUTHORIZE_CALL_ATTEMPT state. The internal processing on the connection after receiving the SETUP will cause the connection to change the state.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created, and the state of connection object is IDLE.
1	The Java Call Control Implementation starts to authorize originating or terminating terminal for the call.
2	The state of connection object transits from IDLE to AUTHORIZE_CALL_ATTEMPT.
3	The Java Call Control Implementation invokes the <i>connectionAuthorizeCallAttempt</i> method.

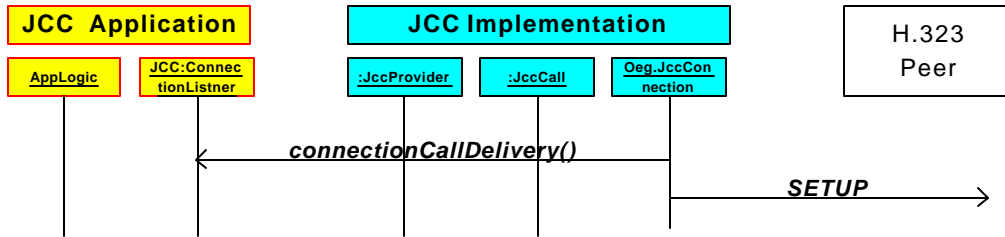
Parameter Mapping

None

```
void connectionCallDelivery(JccConnectionEvent connectionevent)
```

Indicates that the JccConnection has just been placed in the JccConnection.CALL_DELIVERY state

Call Flow



Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created, and the state of connection object is AUTHORIZE_CALL_ATTEMPT or ADDRESS_ANALYZE.
1	The Java Call Control Implementation becomes to be able to use the routing address and call type on the originating connection, or The Java Call Control Implementation authorize the terminating connection
2	The state of connection object transits to CALL_DELIVERY.
3	The Java Call Control Implementation invokes the <i>connectionCallDelivery</i> method.

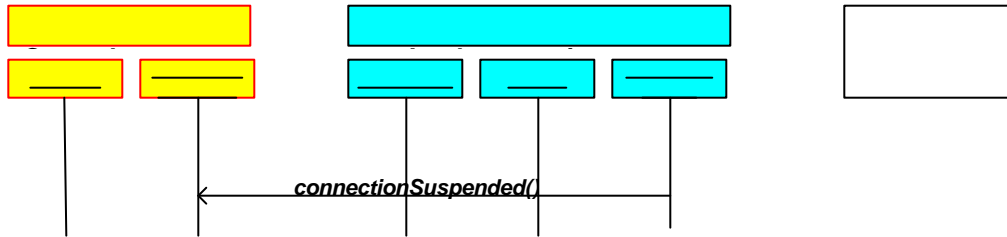
Parameter Mapping

None

```
void connectionSuspended(JccConnectionEvent connectionevent)
```

Indicates that the JccConnection has just been placed in the JccConnection.SUSPENDED state

Call Flow



Note: This is not associated with any particular H.323 message.

Normal Operation

Pre-conditions	The listener object for the connection has been registered. Call and connection object has been created and connected.
1	The Java Call Control Implementation receives the message, which indicates transition to SUSPEND.
2	The state of connection object transits to CALL_SUSPEND.
3	The Java Call Control Implementation invokes the <i>connectionSuspend</i> method.

Parameter Mapping

None

3.0 Methods not mapped

3.1 *JccProvider*

```
void addProviderListener(JcpProviderListener providerlistener)
```

Adds a listener to this provider.

```
void removeProviderListener(JcpProviderListener providerlistener)
```

Removes the given listener from the provider.

```
String getName()
```

Returns the unique string name of this JcpProvider instance.

```
int getState()
```

Returns the state of the JcpProvider.

```
void addCallListener(JccCallListener calllistener, EventFilter filter)
```

Add a call listener to all (future and current) call objects within the domain of this provider.

```
void addCallListener(JcpCallListener calllistener)
```

Add a call listener to all (future and current) call objects within the domain of this provider.

```
void addCallLoadControlListener(CallLoadControlListener  
loadcontrollistener, EventFilter filter)
```

Adds a listener to listen to load control related events.

```
void addConnectionListener(JccConnectionListener connectionlistener,  
EventFilter filter)
```

Add a connection listener to all connections under this JcpProvider. Add a connection listener to all connections under this JcpProvider.

```
void removeCallListener(JcpCallListener calllistener)
```

Removes a call listener that was previously registered.

```
void removeCallLoadControlListener(CallLoadControlListener  
loadcontrollistener)
```

Deregisters the load control listener.

```
void removeConnectionListener(JcpConnectionListener connectionlistener)
```

Removes a connection listener that was registered previously.

3.2 JccCall

```
void addCallListener(JcpCallListener calllistener)
```

Add a listener to this call.

```
JcpConnection[] getConnections()
```

Retrieves an array of connections associated with this call.

```
JcpProvider getProvider()
```

Retrieves the provider handling this call object.

```
int getState()
```

Retrieves the state of the call.

```
void removeCallListener(JcpCallListener calllistener)
```

Removes a listener from this call.

```
void addCallListener(JccCallListener calllistener, EventFilter filter)
```

Add a listener to this call.

```
void addConnectionListener(JccConnectionListener cl, EventFilter  
filter)
```

Add a connection listener to all connections under this call.

```
void removeConnectionListener(JccConnectionListener cl)
```

Removes the connection listener from all connections under this call.

3.3 JccConnection

`JcpCall getCall()`

Retrieves the JcpCall that is associated with this JcpConnection.

`int getState()`

Retrieves the state of the JcpConnection object.

`int getJccState()`

Retrieves the state of the JccConnection object.

`boolean isBlocked()`

Returns a boolean value indicating if the JccConnection is currently blocked due to a blocking event having been fired to a listener registered for that blocking event.

`void attachMedia()`

This method will allow transmission on all associated bearer connections or media channels to and from other parties in the call.

`void detachMedia()`

This method will detach the JccConnection from the call, i.e., this will prevent transmission on any associated bearer connections or media channels to and from other parties in the call.

4.0 References

- [1] Java Call Control (the Java Call Control) Specification JCP JSR21 – <http://jcp.org/jsr/detail/21.jsp>
- [2] JAIN H323 Specification JCP JSR32 - <http://jcp.org/jsr/detail/32.jsp>
- [3] Recommendation H323 - <http://www.itu.int/itudoc/itu-t/>
- [4] the Java Call Control to INAP Mapping – <http://java.sun.com/products/jain>
- [5] the Java Call Control to SIP Mapping - <http://java.sun.com/products/jain>

5.0 Appendix

5.1 the Java Call Control Connection Object FSM

