

# **JAIN<sup>tm</sup>, a set of Java<sup>tm</sup> APIs for Integrated Networks**

## **JAIN JAVA CALL CONTROL (JCC) APPLICATION PROGRAMMING INTERFACE (API) Version 0.8.4**

### **Example Sequence Diagrams**

**September 21, 2000**

© Copyright 2000 Sun Microsystems, Inc.

---

## 1. Introduction

This document presents several example call flows to illustrate use of the JAIN Java Call Control (JCC) API. This document is a companion to the JCC Overview and the JCC specification. Note that this document assumes the JCC package (and not the JCP package) of the specification is being used.

The call flows are represented using message sequence charts. The goal is to illustrate typical sequences based on the current version of the JCC specification, which the application programmer might use in order to provide the named service. Messages signaled between objects are shown as numbered lines with arrows indicating the direction of the signal. Temporal ordering of messages is indicated by the sequence number. Each message reflects the operation name to be invoked by the object receiving the message. The general format of the call flow diagrams is described below.

### 1.1 Service overview

Before each sequence diagram, a brief description of the service is given.

### 1.2 Objects

These are given at the top of each call flow. Where given, the names of the objects implementing the interfaces are given before the colon. The interface types are given after the colon.

### 1.3 API method invocations

API method invocations are shown as messages flowing between objects representing the application and the objects representing the JCC platform. These messages are shown as numbered arrows. Sequence numbers indicate the ordering of messages. Each message is given a message name, which indicates the method call to be invoked on the object receiving the message.

Intra-application or intra-service flows are implementation dependent and are also shown in some instances.

### 1.4 Description of service

At the end of each sequence diagram, a detailed description, describing each of the call flows, for the service is given.

---

## 2. The JCC call flow sequence diagrams

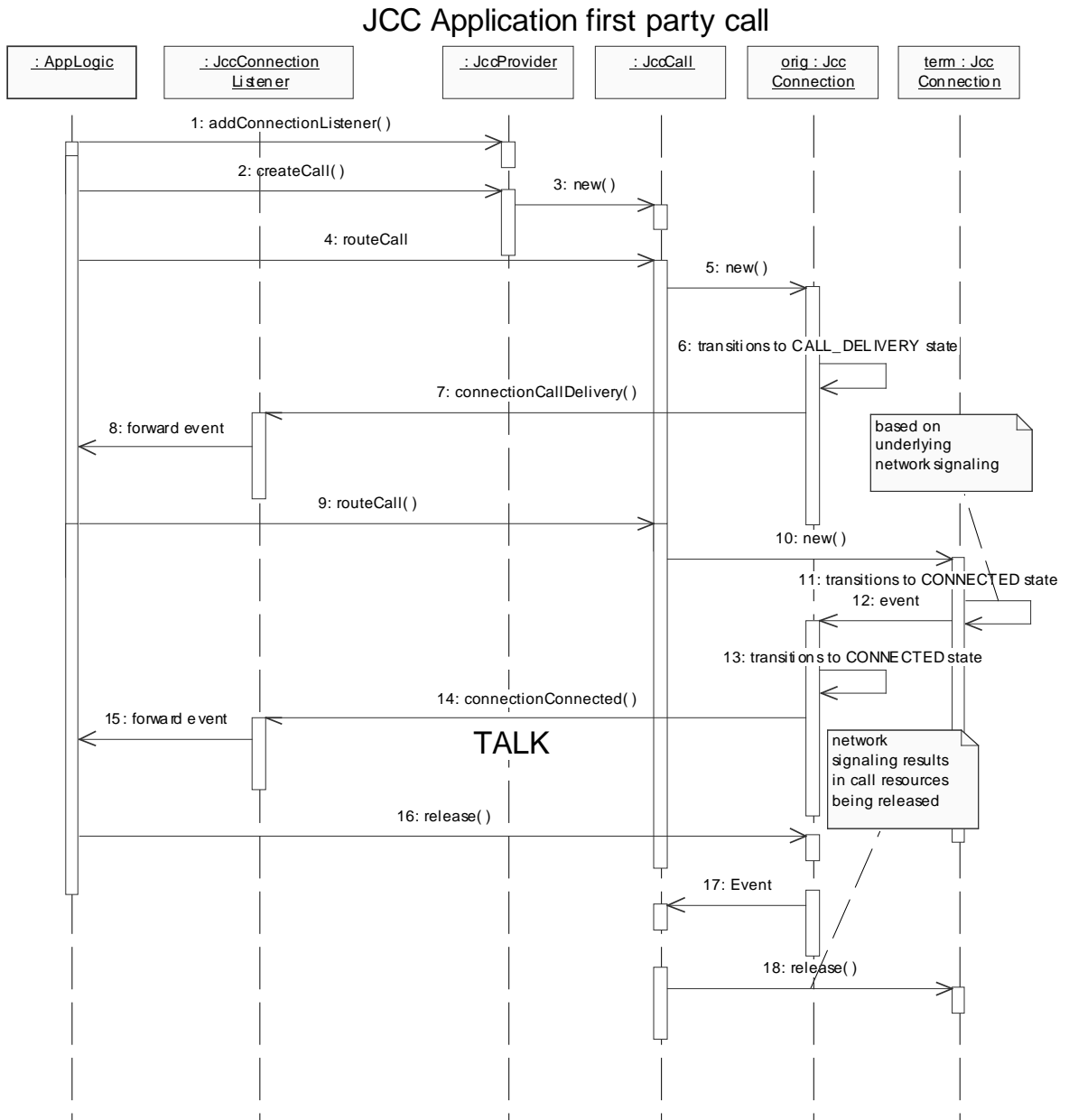
In all the sequence diagrams AppLogic is referring to the Application Logic written by the programmer. The application developer is also expected to provide an implementation of the relevant Listener classes (JcpProviderListener, JcpCallListener, JcpConnectionListener etc) that the application expects to use. Normally, we show only the JccConnectionListener since we assume that in our call flows the application would need the services of only the JccConnectionListener. Note also that the JccConnectionListener is a subclass of JccCallListener.

The objects provided by the JCC implementation are the JccProvider, JccCall, JccConnection and the JccAddress objects. In the call flows we may show only some of these objects for the purpose of avoiding confusion.

We would also like to point out that in the call flows shown before the call flow corresponding to the Virtual Private Network, we do not show the EventFilter interfaces so as to keep the explanation and flows simple. Note that EventFilter interface is expected to provide the event filtering functionality. Note that the later call flows do show the EventFilters. Finally, we would also like to point out that in these call flows we do not show the occurrence of all the events again from the point of view of simplicity. Hence, only the significant events are explicitly shown.

## 2.1 First Party Call--Origination

The following diagram depicts a simple call flow. The scenario considered is that of an application at an end point originating a call.



1. The application adds a listener to the call. The application can use the addCallListener(callListener) method if it desires to receive all events on all addresses

---

associated with this JccProvider. Note that this is okay since the JccConnectionListener interface extends the JccCallListener interface. With the call flow shown, an EventFilter to filter the events is also expected to be provided. The EventFilter is an object which implements the EventFilter interface and is expected to be provided by the application. In order to improve performance a set of standard EventFilters will be provided by the JCC implementation. The application if it desires can choose this set of standard EventFilters. In this flow though we do not show the EventFilter to keep the flow simple as explained earlier.

2. This message is then used by the application to request the JccProvider to create an object implementing the JccCall interface.
3. The message is used by the JccProvider to create an object implementing the JccCall interface.
4. This message instructs the object implementing the JccCall interface to create a JccConnection object representing the originating party and route it to the party. Note that using this API method both the creation and routing of the JccConnection object is done. It is also possible for the application to have these tasks done separately by using APIs like JccCall.createConnection and JccConnection.routeConnection.
5. This message is used to create an object implementing the JccConnection interface representing the originating party hereafter referred to as the originating JccConnection. The JccConnection passes through different states while proceeding with the different steps of basic call processing such as analyzing the digits dialed (for special processing such as on 1-800 calls etc), authorizing whether call with the given number can be setup etc.
6. This results in the JccConnection object finally transitioning to the CALL\_DELIVERY state.
7. The registered JccConnectionListener is then informed of the originating JccConnection being in the CALL\_DELIVERY state. The assumption here is that the EventFilter has advised the JCC implementation to inform the listener of this event on the given address. This is done by sending an appropriate JccConnectionEvent using the connectionCallDelivery(connectionevent) method which is to be implemented by the registered JccConnectionListener object.
8. The JccConnectionListener then informs the application about the occurrence of the event specified.
9. The application in order to complete the call, then sets to “dial” the terminating party. It does this by specifying the “number” of the destination party using the routeCall() method.
10. This results in the creation of a new JccConnection object representing the terminating party hereafter referred to as the terminating JccConnection.
11. The terminating JccConnection object transits to the CONNECTED state possibly as a result of underlying network signaling causing this to happen. Note that many intervening events such as the alerting of the destination party etc have not been shown.
12. An event internal to the implementation then informs the originating JccConnection that the destination terminating JccConnection is in the CONNECTED state. Note that this way of

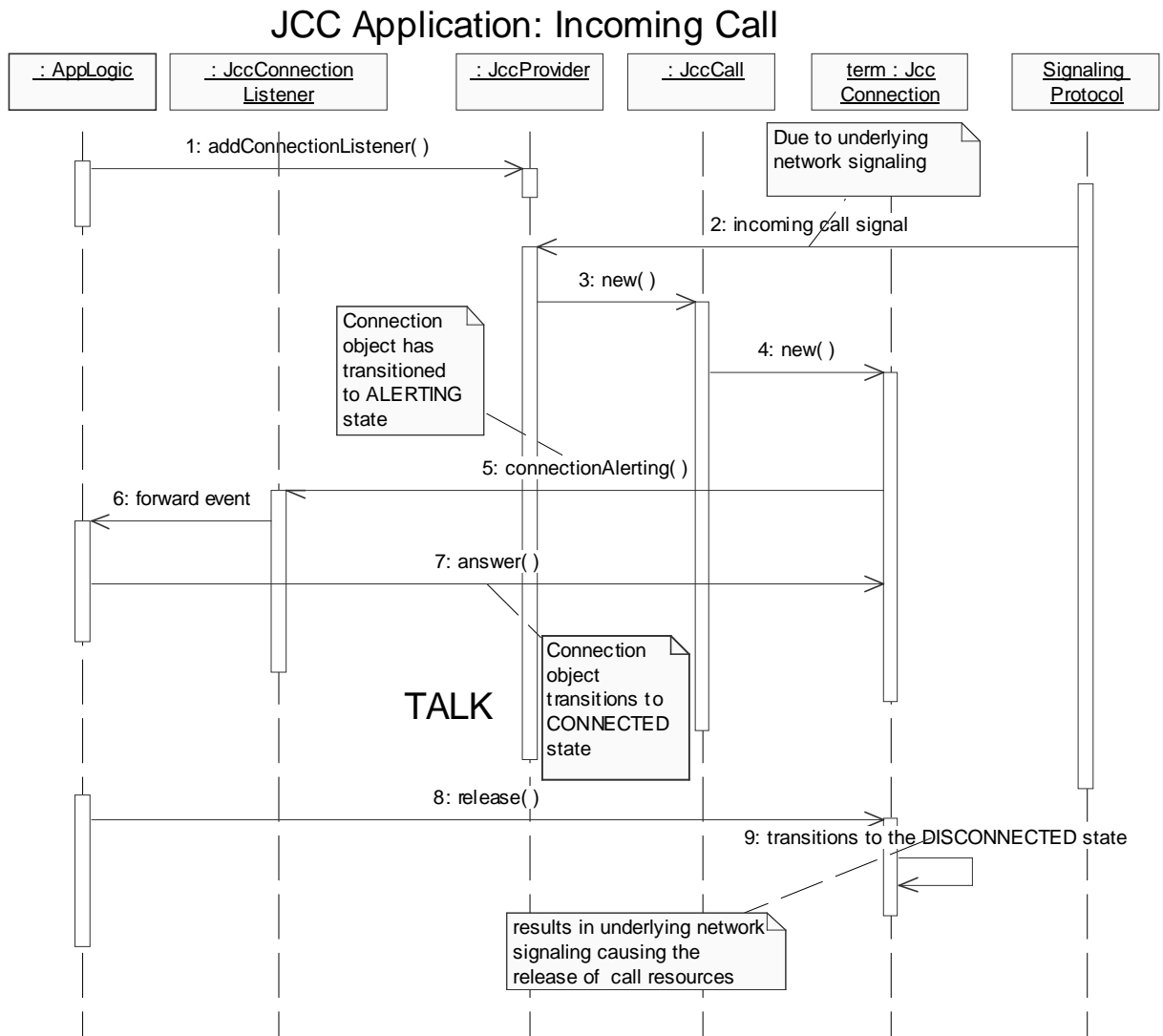
---

informing the originating JccConnection is implementation specific and is not governed in any way by the API specification.

13. The originating JccConnection transits to the CONNECTED state.
14. The JCC implementation then informs the JccConnectionListener that the original JccConnection is in the CONNECTED state.
15. The application is then informed of the terminating JccConnection being in the CONNECTED state by the JccConnectionListener.
16. When the application decides to end the conversation, it does so by using this message on the originating JccConnection. This results in the originating JccConnection transitioning from the CONNECTED state to the DISCONNECTED state. Call resources related to the originating party may also be released on account of network signaling. Note also that this is one possible way in which the call may be released. There are other ways in which the call can be disconnected and which are not shown here. Additional JccConnections may be dropped indirectly as a result of this method. For example, dropping the destination JccConnection of a two-party call as shown here may result in the entire telephone call being dropped which is what is shown in the next two flows. It is up to the implementation to determine which JccConnections are dropped as a result of this method. Implementations should not, however, drop additional JccConnections representing additional parties if it does not reflect the natural response of the underlying telephone hardware.
17. Since there are only two parties in this call, this also results in the other party namely the terminating JccConnection also transitioning to the DISCONNECTED state. This is caused by this event send internally within the JCC implementation. Note that the way events are passed around classes is highly implementation specific.
18. This message causes the terminating JccConnection to transition to the DISCONNECTED state and call resources related to the destination party used up in the network are also released as a result of network signaling.

## 2.2 First Party Call--Termination

The following diagram depicts a simple call flow. The scenario considered is that of an application at an end point terminating a call. Hence, we do not show the originating party in this call.



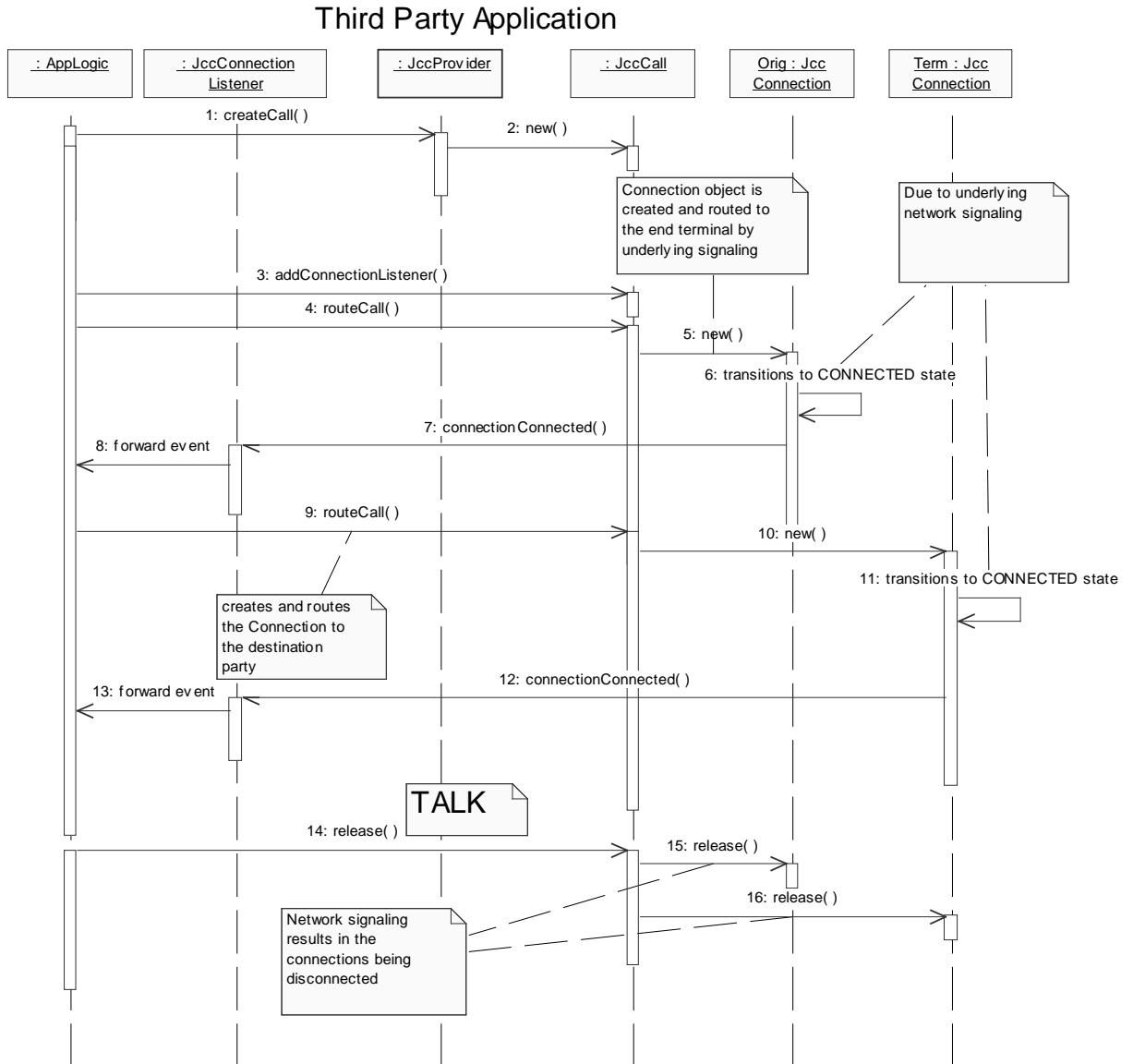
1. This message is used by the application to add a listener to the call. It is assumed that the application is interested in receiving the CONNECTION\_ALERTING event for the address of interest shown here.

- 
2. A network event (because of the signaling protocol) is delivered to the JccProvider notifying the JccProvider of an incoming call on the end point of interest.
  3. The JccProvider creates a new JccCall object representing the incoming call.
  4. A new JccConnection object is also created to represent the party receiving the call. Note that in this diagram, the JccConnection is shown as being created by the JccCall object. This fact though depends on the implementation of the JCC.
  5. An event is delivered to the JccConnectionListener alerting the listener to an incoming call. Note that this event is generated by the JCC implementation as a result of the JccConnection being in the ALERTING state.
  6. This event is used by the listener to inform the application that a JccConnection on the endpoint of interest is in the ALERTING state. This message is specific to the applications and is not specified as part of the specification.
  7. The application uses this message to tell the JCC implementation that it does want to receive the call. This results in the JccConnection transitioning to the CONNECTED state. Further, this also results in appropriate network signals being sent to the corresponding party informing it that the call has been accepted.
  8. The application uses this message to release the call and associated resources.
  9. The JccConnection object transitions to the DISCONNECTED state and this results in underlying network signaling which releases all the network resources used in this call. The corresponding party is also informed of the release of the call.



## 2.3 Third Party Application—I

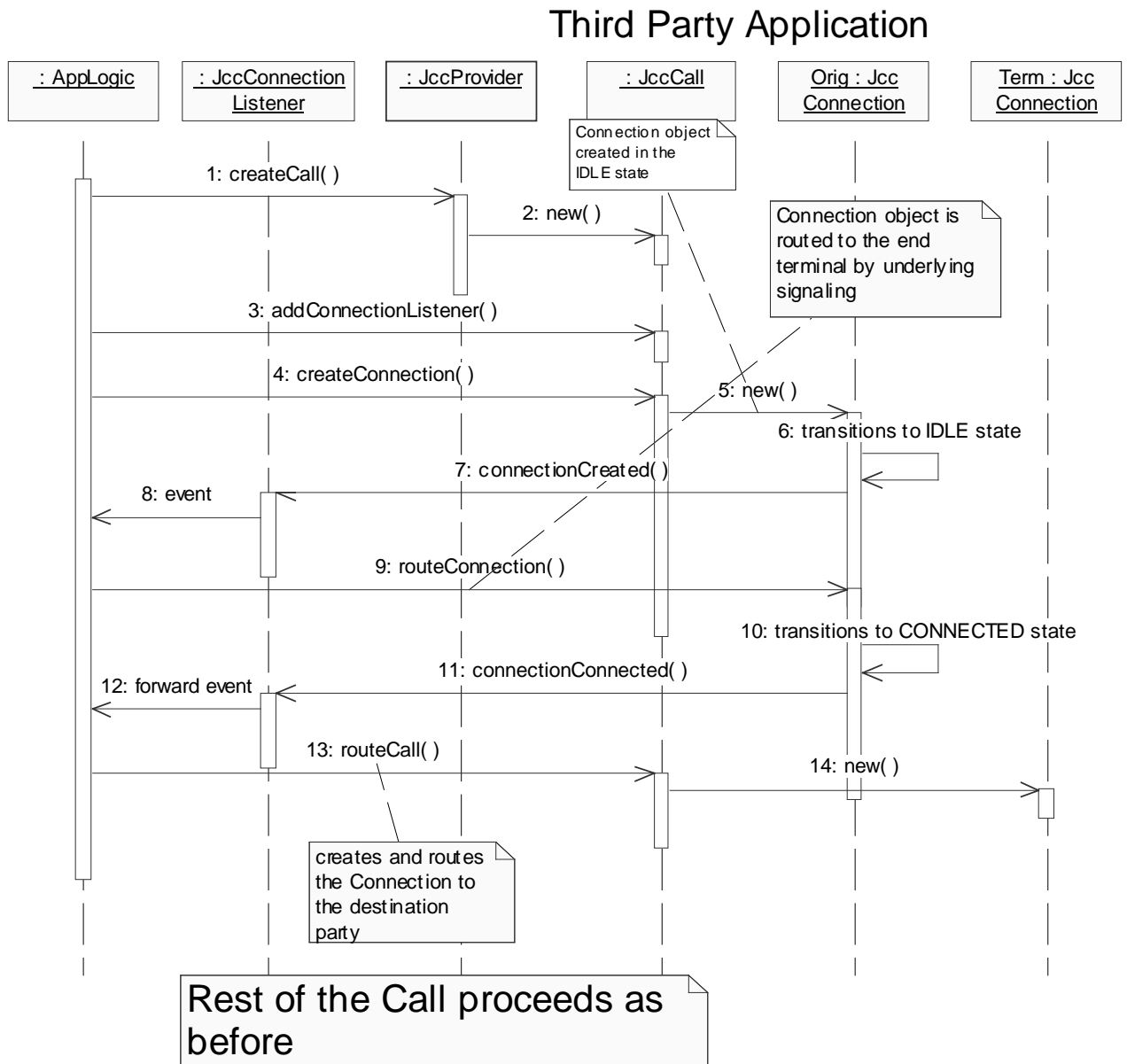
In this scenario we show the call flows associated with a third party application. It is assumed that the JCC application is responsible for dialing two parties A and B and then connecting them together. This is a third party application since the corresponding parties A and B are connected by the application. The parties themselves do not initiate the call. The parties themselves do not initiate the call.



- 
1. The application uses this to request the JCC platform to create a JccCall object so as to initiate the call between the two parties.
  2. This results in the JCC implementation creating the object implementing the JccCall interface.
  3. The application then registers a JccConnectionListener for receiving events associated with this call.
  4. The application then instructs the JCC implementation to place a call to one of the parties by invoking this method.
  5. This results in the JccConnection object representing one of the parties being created. The JccConnection object then passes through different states with corresponding signals being sent on the network to the end terminal of the party of interest.
  6. The JccConnection object transitions to the CONNECTED state.
  7. This message informs the JccConnectionListener that the JccConnection object is in the CONNECTED state.
  8. The JccConnectionListener informs the application of the occurrence of the previous event.
  9. The application then proceeds with calling the “destination”. It does so by using this method to specify to the JCC platform the address of the “destination” party.
  10. The JccConnection object representing the other party is created.
  11. The terminating JccConnection object transits to the CONNECTED state.
  12. The JccConnectionListener is notified that the other party is also connected on the call.
  13. The JccConnectionListener informs the application of the occurrence of the previous event.
  14. The application releases the connection between the two corresponding parties. Note that the connection can also be released due to action taken by either of the end parties but we do not show that here.
  15. This results in the resources used by one of the parties in the call being released and the corresponding JccConnection object transitioning to the DISCONNECTED state.
  16. This results in the resources used by the other party in the call also being released and the corresponding JccConnection object also transitioning to the DISCONNECTED state.

## 2.4 Third Party Application—II

In this scenario also we show the call flows associated with a third party application. The difference though with the earlier call flows is that in this case the first party is connected using some different API methods.



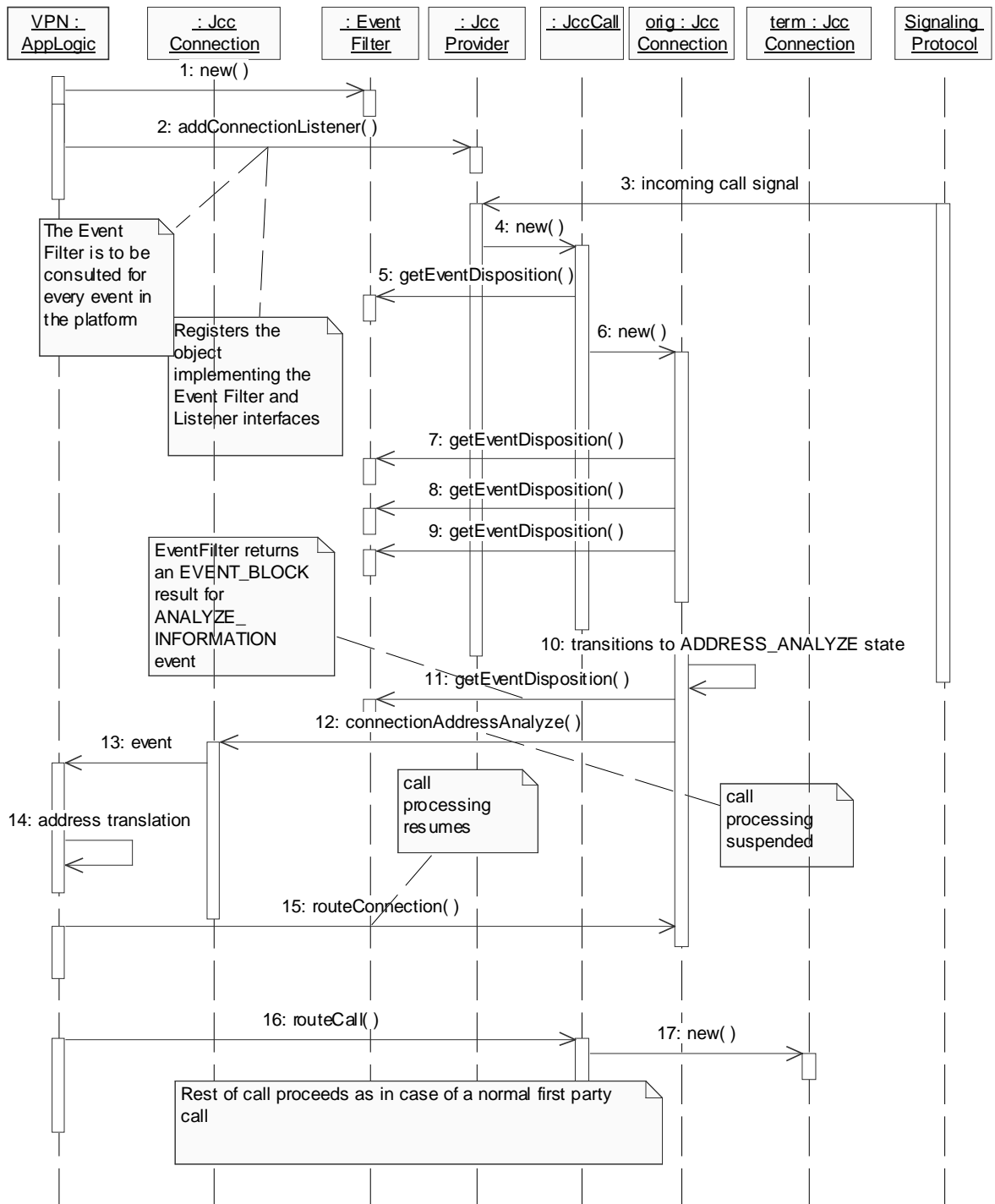
- 
1. The application uses this to create a JccCall object so as to initiate the call between the two parties.
  2. This results in the JCC implementation creating the object implementing the JccCall interface.
  3. The application then registers as a listener for receiving events associated with this call.
  4. The application then instructs the JCC implementation to create a JccConnection. The created JccConnection will be in the IDLE state and is associated with a JccCall and JccAddress.
  5. This results in the object implementing the JccConnection interface being created.
  6. The JccConnection object is created in the IDLE state.
  7. The JccConnectionListener is informed that the JccConnection object has been created and is in the IDLE state.
  8. The JccConnectionListener informs the application of the occurrence of the previous event.
  9. The application then asks the JCC implementation to route the connection to the corresponding end party. This results in network signaling causing the JccConnection object to pass through various states.
  10. The JccConnection object finally transitions to the CONNECTED state.
  11. This message informs the JccConnectionListener of success in connecting one of the parties of the call. This also implies that the corresponding JccConnection object is in the CONNECTED state.
  12. The JccConnectionListener informs the application of the occurrence of the previous event.
  13. The application then proceeds with calling the “destination” party since one of the parties of the call is already connected. It does so by using this method to specify to the JCC platform the address of the “destination” party.
  14. The JccConnection object representing the other party is created.

The rest of the call proceeds as earlier.

## **2.5 Virtual Private Network—with Application supplied EventFilter**

This example call flow demonstrates a basic abbreviated number translation for a VPN user. Unlike earlier call flows, we also explicitly show the EventFilter interface in this figure.

# Virtual Private Network



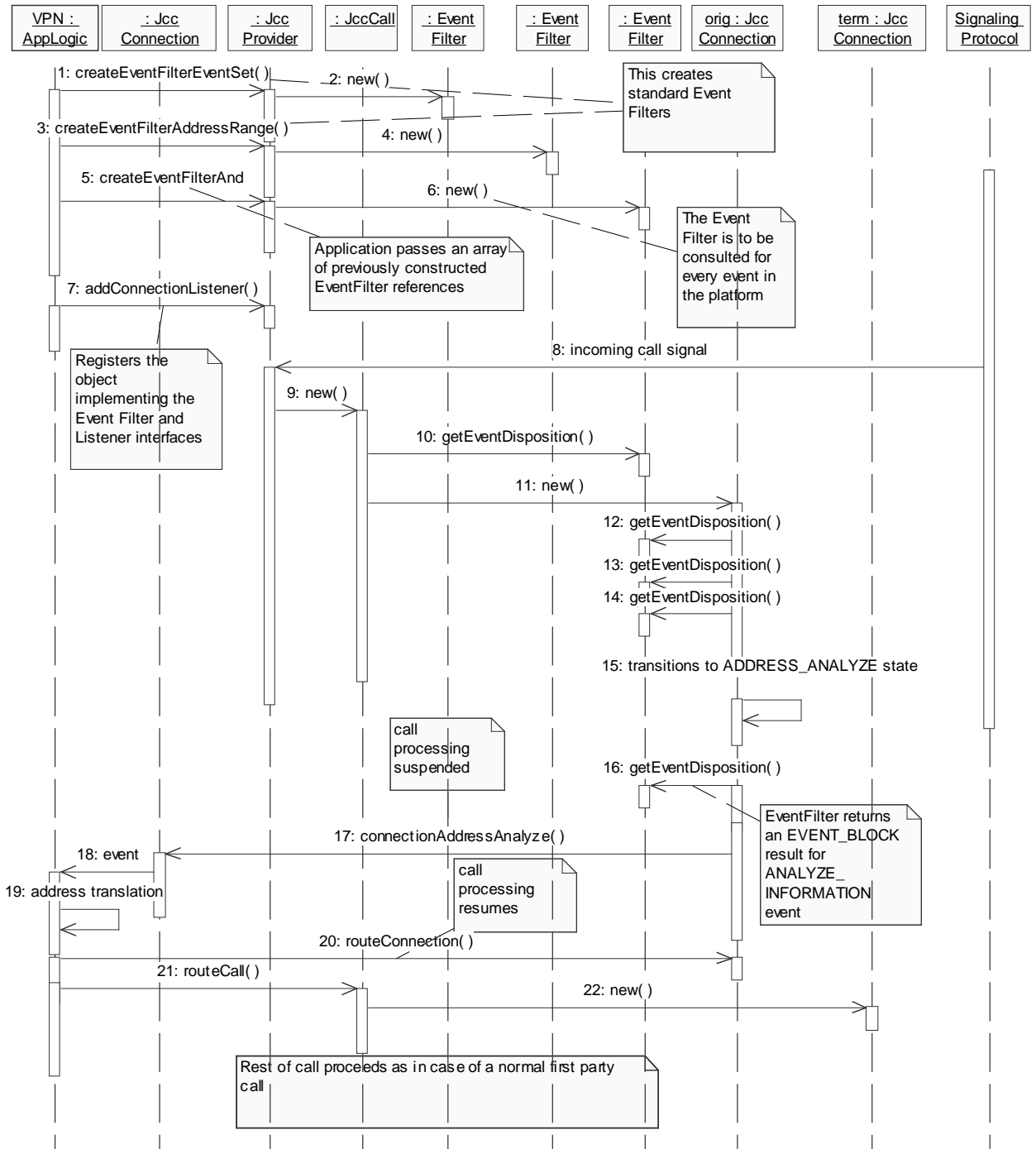
- 
1. This message is used by the application to create an object implementing the EventFilter interface. Note that we show this message explicitly in this case only to bring attention to the fact that the EventFilter interface in this case has to be provided by the application. This is in contrast to the next call flow where standard EventFilter interfaces are used and which are provided by the platform.
  2. This message is used by the VPN application to register the JccConnectionListener object and the EventFilter objects with the JCC implementation. Note that the EventFilter object registered by the VPN application is expected to contain the logic to return EVENT\_BLOCK for the ANALYZE\_INFORMATION event. This is because the VPN application is then expected to translate the address before the call processing can proceed.
  3. This message is used to inform the JccProvider of an incoming call.
  4. The object implementing the JccProvider interface then creates an object implementing the JccCall object to cater to the incoming call.
  5. For every event occurring in the platform, the EventFilter is to be consulted for indication of whether the application is interested in the event. This message shows the platform consulting the EventFilter for one such event, which in this case is the CALL\_CREATED event. It is assumed that the EventFilter returns EVENT\_DISCARD.
  6. The JccCall object then creates a new object implementing the JccConnection interface to model the incoming connection.
  7. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_CREATED). The EventFilter returns EVENT\_DISCARD.
  8. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_AUTHORIZE\_CALL\_ATTEMPT). The EventFilter returns EVENT\_DISCARD.
  9. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_ADDRESS\_COLLECT). The EventFilter returns EVENT\_DISCARD.
  10. This message indicates that the JccConnection has transitioned to the ADDRESS\_ANALYZE state which is the state of interest for the VPN application.
  11. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_ADDRESS\_ANALYZE). EventFilter is expected to return EVENT\_BLOCK.
  12. In this case, since the EVENT\_BLOCK value was returned previously, a ConnectionAddressAnalyze( ) method is used to inform the registered JccConnectionListener of the occurrence of the event. Note also that call processing is blocked as a result.
  13. The JccConnectionListener informs the application of the occurrence of the previous event.
  14. This flow shows that the VPN application is then responsible for translating the address.

- 
15. This message is then used by the *VPN* Application to instruct the platform to proceed with further processing of the call for the incoming call leg. As a result of this, the *JccConnection* object representing the incoming call leg transitions to an appropriate state.
  16. The *VPN* application having translated the digits uses this message to route the call which has come in, to the proper destination.
  17. This message results in a new *JccConnection* object being created in order to model the terminating party which in this case represents the translated number.

## **2.6 Virtual Private Network—with Standard EventFilters**

This example call flow demonstrates a basic abbreviated number translation for a *VPN* user. We show the use of standard *EventFilters* to enhance the performance of the system in this call flow.

## Virtual Private Network



1. This message is used by the application to request the JCC platform to create an EventFilter object, which will filter events, based on their types such as JccCONNECTION.ALERTING, JccCall.ACTIVE etc.



- 
2. This message is used by the JCC platform to create a new EventFilter object.
  3. This message is used by the application to request the JCC platform to create an EventFilter object, which will filter events, based on the endpoints of occurrence.
  4. This message is used by the JCC platform to create a new EventFilter object.
  5. This message is used by the application to request the JCC platform to create an EventFilter object which will use both the previously created filters. Note that this results in events being filtered based on the types as well based on the endpoints of occurrence.
  6. This message is used by the JCC platform to create a new EventFilter object.
  7. This message is used by the VPN application to register a JccConnectionListener and an EventFilter object (one created with an array of EventFilters in a previous step) with the JCC implementation. Note that the EventFilter interface registered by the VPN application is expected to contain the logic to return EVENT\_BLOCK for the ANALYZE\_INFORMATION event. This is because the VPN application is then expected to translate the address before the call processing can proceed.
  8. This message is used to inform the JccProvider of an incoming call.
  9. The object implementing the JccProvider interface then creates an object implementing the JccCall object to cater to the incoming call.
  10. For every event occurring in the platform, the registered EventFilter is to be consulted for indication of whether the application is interested in the event. This message shows the platform consulting the registered EventFilter for one such event (CALL\_CREATED). It is assumed that the EventFilter returns EVENT\_DISCARD. Note that the consultation is shown being done using the method getEventDisposition(). In reality, this might be an implementation specific method.
  11. The JccCall object then creates a new object implementing the JccConnection interface to model the incoming connection.
  12. This message also shows the platform consulting the registered EventFilter for another event (CONNECTION\_CREATED). The EventFilter returns EVENT\_DISCARD.
  13. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_AUTHORIZE\_CALL\_ATTEMPT). The EventFilter returns EVENT\_DISCARD.
  14. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_ADDRESS\_COLLECT). The EventFilter returns EVENT\_DISCARD.
  15. This message indicates that the JccConnection has transitioned to the ADDRESS\_ANALYZE state which is the state of interest for the VPN application.
  16. This message also shows the platform consulting the EventFilter for another event (CONNECTION\_ADDRESS\_ANALYZE). EventFilter is expected to return EVENT\_BLOCK.

- 
17. In this case, since the `EVENT_BLOCK` value was returned previously, a `ConnectionAddressAnalyze()` method is used to inform the registered `JccConnectionListener` of the occurrence of the event. Note also that call processing is blocked as a result.
  18. The `JccConnectionListener` informs the application of the occurrence of the previous event.
  19. This flow shows that the VPN application is then responsible for translating the address.
  20. This message is then used by the *VPN* Application to instruct the platform to proceed with further processing of the call for the incoming call leg. As a result of this, the `JccConnection` object representing the incoming call leg transitions to an appropriate state.
  21. The VPN application having translated the digits uses this message to route the call which has come in, to the proper destination.
  22. This message results in a new `JccConnection` object being created in order to model the terminating party which in this case represents the translated number.