Platform SDK Developer's Guide

Telephony (T-Server)

4/30/2025

Telephony (T-Server)

Java

You can use the Voice Platform SDK to write Java or .NET applications that monitor and handle voice interactions from a traditional or IP-based telephony device. These applications can range from the simple to the advanced. This document shows how to implement the basic functions you will need to write a simple voice application. It is organized to show the kind of structure you will probably use to write your own applications.

Setting Up a TServerProtocol Object

The first thing you need to do to use the Voice Platform SDK is instantiate a `TServerProtocol` object. To do that, you must supply information about the T-Server you want to connect with. This example provides the server's name, host, and port information:

```
[Java]

TServerProtocol tServerProtocol =
        new TServerProtocol(
                new Endpoint(
                        tServerName, host, port));
```

After instantiating the `TServerProtocol` object, you need to open the connection to the T-Server:

```
[Java]

tServerProtocol.open();
```

Registering an Address

Now you need to register a DN for your agent to use. To do this, you must send a `RequestRegisterAddress` request to the server.

Here is how to create this request:

```
[Java]

RequestRegisterAddress requestRegisterAddress =
        RequestRegisterAddress.create(
                thisDn,
                RegisterMode.ModeShare,
                ControlMode.RegisterDefault,
                AddressType.DN);
```

The `thisDn` argument refers to the DN you want to associate with your agent, while `RegisterMode.ModeShare` tells the T-Server to share information about the DN with other applications. The next argument asks to use the switch's default value for deciding whether to let the switch know that you have registered this DN. And finally, you are specifying that the object you are registering is a DN.

After you create the request, you will need to send it to the T-Server:

```
[Java]

Message response =
        tServerProtocol.request(requestRegisterAddress);
```

Remember that the `request()` method is synchronous. If you use this method, your application will block until you hear back from the server. When you get the response, you can execute code to handle the response. In this case, you probably don't need to do anything if the request is successful:

```
[Java]

switch(response.messageId())
{
        case EventRegistered.ID:
        case EventUnregistered.ID:
                break;
        .
        .
        .
}
```

Logging in an Agent

Once you have registered a DN to your agent, you can log him or her in. To do this, you need to create a `RequestAgentLogin` request:

```
[Java]

RequestAgentLogin requestAgentLogin =
        RequestAgentLogin.create(
                thisDn,
                AgentWorkMode.AutoIn);
```

After you create the request, you will need to indicate the queue the agent will be using, and you may need to supply the agent's user name and password. Once you have done this, you can send the request to the server:

```
[Java]

requestAgentLogin.setThisQueue(thisQueue);
// Your switch may not need a user name and password:
requestAgentLogin.setAgentID(userName);
requestAgentLogin.setPassword(password);
Message response = tServerProtocol.request(requestAgentLogin);
```

If your request is successful, the server will respond with an `EventAgentLogin` event. At that point, you may need to update the state of your user interface to indicate that the agent can no longer log in, but that, for example, he or she can now log out.

Answering a Call

Now that your agent is logged in, he or she can handle calls. Let's start by answering a call.

When a call comes in, your application will receive an EventRinging message. When you get this message, you will probably want to enable an answer button. Here is how to do that:

```
[Java]

switch(response.messageId())
{
        .
        .
        .
        case EventRinging.ID:
                EventRinging eventRinging = (EventRinging) response;
                connId = eventRinging.getConnID();
                if (eventRinging.getThisDN() == thisDn)
                {
                        AnswerButton.enabled = true;
                }
                break;
        .
        .
        .
}
```

It is important to note that an EventRinging event will also be triggered when you are sending an outbound call. So this particular snippet is only enabling the answer button if the call is ringing on thisDN. As you can also see, when you receive an EventRinging you will want to store the ConnID of the call associated with it.

After the agent clicks the answer button, you need to send a request to answer the call, using your DN and the ConnID of the call:

```
[Java]

RequestAnswerCall requestAnswerCall =
        RequestAnswerCall.create(
        thisDn,
        connId);
Message response = tServerProtocol.request(requestAnswerCall);
```

If the request is successful, you will receive an EventEstablished.

Releasing a Call

When your agent is finished with the call, he or she will need to release it:

```
[Java]

RequestReleaseCall requestReleaseCall =
        RequestReleaseCall.create(
                thisDn,
                connId);
```

```
Message response = tServerProtocol.request(requestReleaseCall);
```

If the request is successful, you will receive an EventReleased.

Making a Call

Here is how to make a call:

```
[Java]

RequestMakeCall requestMakeCall =
        RequestMakeCall.create(
                thisDn,
                thatDn,
                MakeCallType.DirectAgent);
Message response = tServerProtocol.request(requestMakeCall);
```

If the request is successful, you will receive an EventDialing message, an EventRinging message, and then, when your party responds, an EventEstablished message.

Setting up a Conference Call

After you make or answer a call, you can add another party to the call. Here is how to perform an ordinary two-step conference call.

To start off, you need to initiate a conference call, supplying your own DN, the connection ID of the existing call, and the DN of the party you want to add to the call:

```
[Java]

RequestInitiateConference requestInitiateConference =
        RequestInitiateConference.create(
                thisDn,
                connId,
                otherDn);
Message response = tServerProtocol.request(requestInitiateConference);
```

> ### Tip
> In a real telephony application, the events you would receive in response to the kinds of conferencing requests shown here could also be generated by other requests. For example, you might receive an EventDialing or an EventEstablished in response to a RequestMakeCall or RequestInitiateTransfer. Because of this, a real-world application will need to keep track of the requests that initiate these events in order to interpret them correctly.

If the initiate request is successful, you will receive an EventDialing message and an EventHeld message. When your party picks up the call, you will also receive an EventEstablished message.

Now you need to complete the conference call.

When you received the EventDialing message from the RequestInitiateConference, you were given a new connection ID associated with the party you want to establish the conference call with. You will need that connection ID, in addition to your own DN and the original connection ID, in order to complete the conference call:

```
[Java]

RequestCompleteConference requestCompleteConference =
        RequestCompleteConference.create(
                thisDn,
                connId,
                secondConnId);
response = tServerProtocol.request(requestCompleteConference);
```

If the completion request is successful, you will receive EventReleased, EventRetrieved, EventPartyAdded, and EventAttachedDataChanged messages.

Transferring a Call

After you make or answer a call, you may also want to transfer that call. Here is how to perform an ordinary two-step transfer.

To start off, you need to initiate a transfer, supplying your own DN, the connection ID of the existing call, and the DN of the party you want to transfer the call to.

```
[Java]

RequestInitiateTransfer requestInitiateTransfer =
        RequestInitiateTransfer.create(
                thisDn,
                connId,
                otherDn);
Message response = tServerProtocol.request(requestInitiateTransfer);
```

> ### Tip
> In a real telephony application, the events you would receive in response to the kinds of transfer requests shown here could also be generated by other requests. For example, you might receive an EventDialing or an EventEstablished in response to a RequestMakeCall or RequestInitiateConference. Because of this, a real-world application will need to keep track of the requests that initiate these events in order to interpret them correctly.

If the initiate request is successful, you will receive an EventDialing message and an EventHeld message. When the party you want to transfer to picks up the call, you will also receive an EventEstablished message.

Now you need to complete the transfer.

When you received the EventDialing message from the RequestInitiateTransfer, you were given a new connection ID associated with the party you want to transfer the call to. You will need that connection ID, in addition to your own DN and the original connection ID, in order to complete the transfer:

```
[Java]

RequestCompleteTransfer requestCompleteTransfer =
        RequestCompleteTransfer.create(
                thisDn,
                connId,
                secondConnId);
response = tServerProtocol.request(requestCompleteTransfer);
```

If the completion request is successful, you will receive two EventReleased messages and you will no longer be a party to the call.

Closing the Connection

Finally, when you are finished communicating with the T-Server, you should close the connection to minimize resource utilization:

```
[Java]

tServerProtocol.close();
```

.NET

You can use the Voice Platform SDK to write Java or .NET applications that monitor and handle voice interactions from a traditional or IP-based telephony device. These applications can range from the simple to the advanced. This document shows how to implement the basic functions you will need to write a simple voice application. It is organized to show the kind of structure you will probably use to write your own applications.

Setting Up a TServerProtocol Object

The first thing you need to do to use the Voice Platform SDK is instantiate a TServerProtocol object. To do that, you must supply information about the T-Server you want to connect with. This example uses the URI of the T-Server, but you can also use name, host, and port information:

```
[C#]

TServerProtocol tServerProtocol =
        new TServerProtocol(
                new Endpoint(
                        tServerUri));
```

After instantiating the TServerProtocol object, you need to open the connection to the T-Server:

```
[C#]

tServerProtocol.Open();
```

Registering an Address

Now you need to register a DN for your agent to use. To do this, you must send a `RequestRegisterAddress` request to the server.

Here is how to create this request:

```
[C#]

RequestRegisterAddress requestRegisterAddress =
        RequestRegisterAddress.Create(
        thisDn,
        RegisterMode.ModeShare,
        ControlMode.RegisterDefault,
        AddressType.DN);
```

The `thisDn` argument refers to the DN you want to associate with your agent, while `RegisterMode.ModeShare` tells the T-Server to share information about the DN with other applications. The next argument asks to use the switch's default value for deciding whether to let the switch know that you have registered this DN. And finally, you are specifying that the object you are registering is a DN.

After you create the request, you will need to send it to the T-Server:

```
[C#]

IMessage response = tServerProtocol.Request(requestRegisterAddress);
```

Remember that the `Request()` method is synchronous. If you use this method, your application will block until you hear back from the server. When you get the response, you can execute code to handle the response. In this case, you probably don't need to do anything if the request is successful:

```
[C#]

switch(response.Id )
{
        case EventRegistered.MessageId:
        case EventUnregistered.MessageId:
                break;
        .
        .
        .
}
```

Logging in an Agent

Once you have registered a DN to your agent, you can log him or her in. To do this, you need to create a `RequestAgentLogin` request:

```
[C#]

RequestAgentLogin requestAgentLogin =
        RequestAgentLogin.Create(
        thisDn,
        AgentWorkMode.AutoIn);
```

After you create the request, you will need to indicate the queue the agent will be using, and you may need to supply the agent's user name and password. Once you have done this, you can send the request to the server:

```
[C#]

requestAgentLogin.ThisQueue = thisQueue;
// Your switch may not need a user name and password:
requestAgentLogin.AgentID = userName;
requestAgentLogin.Password = password;
IMessage response = tServerProtocol.Request(requestAgentLogin);
```

If your request is successful, the server will respond with an `EventAgentLogin` event. At that point, you may need to update the state of your user interface to indicate that the agent can no longer log in, but that, for example, he or she can now log out.

Answering a Call

Now that your agent is logged in, he or she can handle calls. Let's start by answering a call.

When a call comes in, your application will receive an `EventRinging` message. When you get this message, you will probably want to enable an answer button. Here is how to do that:

```
[C#]

switch(response.Id)
{
        .
        .
        .
        case EventRinging.MessageId:
                EventRinging eventRinging = (EventRinging) response;
                connId = eventRinging.ConnID;
                if (eventRinging.ThisDN == thisDn)
                {
                        AnswerButton.Enabled = true;
                }
                break;
        .
        .
        .
}
```

It is important to note that an `EventRinging` event will also be triggered when you are sending an outbound call. So this particular snippet is only enabling the answer button if the call is ringing on `thisDN`. As you can also see, when you receive an `EventRinging` you will want to store the `ConnID` of the call associated with it.

After the agent clicks the answer button, you need to send a request to answer the call, using your

DN and the ConnID of the call:

```
[C#]

RequestAnswerCall requestAnswerCall =
        RequestAnswerCall.Create(
        thisDn,
        connId);
IMessage response = tServerProtocol.Request(requestAnswerCall);
```

If the request is successful, you will receive an EventEstablished.

Releasing a Call

When your agent is finished with the call, he or she will need to release it:

```
[C#]

RequestReleaseCall requestReleaseCall =
        RequestReleaseCall.Create(
        thisDn,
        connId);
IMessage response = tServerProtocol.Request(requestReleaseCall);
```

If the request is successful, you will receive an EventReleased.

Making a Call

Here is how to make a call:

```
[C#]

RequestMakeCall requestMakeCall =
        RequestMakeCall.Create(
        thisDn,
        thatDn,
        MakeCallType.DirectAgent);
IMessage response = tServerProtocol.Request(requestMakeCall);
```

If the request is successful, you will receive an EventDialing message, an EventRinging message, and then, when your party responds, an EventEstablished message.

Setting up a Conference Call

After you make or answer a call, you can add another party to the call. Here is how to perform an ordinary two-step conference call.

To start off, you need to initiate a conference call, supplying your own DN, the connection ID of the existing call, and the DN of the party you want to add to the call:

```
[C#]

RequestInitiateConference requestInitiateConference =
        RequestInitiateConference.Create(
        thisDn,
        connId,
        otherDn);
IMessage response = tServerProtocol.Request(requestInitiateConference);
```

> **Tip**
>
> In a real telephony application, the events you would receive in response to the kinds of conferencing requests shown here could also be generated by other requests. For example, you might receive an `EventDialing` or an `EventEstablished` in response to a `RequestMakeCall` or `RequestInitiateTransfer`. Because of this, a real-world application will need to keep track of the requests that initiate these events in order to interpret them correctly.

If the initiate request is successful, you will receive an `EventDialing` message and an `EventHeld` message. When your party picks up the call, you will also receive an `EventEstablished` message.

Now you need to complete the conference call.

When you received the `EventDialing` message from the `RequestInitiateConference`, you were given a new connection ID associated with the party you want to establish the conference call with. You will need that connection ID, in addition to your own DN and the original connection ID, in order to complete the conference call:

```
[C#]

RequestCompleteConference requestCompleteConference =
        RequestCompleteConference.Create(
        thisDn,
        connId,
        secondConnId);
response = tServerProtocol.Request(requestCompleteConference);
```

If the completion request is successful, you will receive `EventReleased`, `EventRetrieved`, `EventPartyAdded`, and `EventAttachedDataChanged` messages.

Transferring a Call

After you make or answer a call, you may also want to transfer that call. Here is how to perform an ordinary two-step transfer.

To start off, you need to initiate a transfer, supplying your own DN, the connection ID of the existing call, and the DN of the party you want to transfer the call to.

```
[C#]

RequestInitiateTransfer requestInitiateTransfer =
```

```
        RequestInitiateTransfer.Create(
        thisDn,
        connId,
        otherDn);
IMessage response = tServerProtocol.Request(requestInitiateTransfer);
```

> **Tip**
>
> In a real telephony application, the events you would receive in response to the kinds of transfer requests shown here could also be generated by other requests. For example, you might receive an EventDialing or an EventEstablished in response to a RequestMakeCall or RequestInitiateConference. Because of this, a real-world application will need to keep track of the requests that initiate these events in order to interpret them correctly.

If the initiate request is successful, you will receive an EventDialing message and an EventHeld message. When the party you want to transfer to picks up the call, you will also receive an EventEstablished message.

Now you need to complete the transfer.

When you received the EventDialing message from the RequestInitiateTransfer, you were given a new connection ID associated with the party you want to transfer the call to. You will need that connection ID, in addition to your own DN and the original connection ID, in order to complete the transfer:

```
[C#]

RequestCompleteTransfer requestCompleteTransfer =
        RequestCompleteTransfer.Create(
        thisDn,
        connId,
        secondConnId);
response = tServerProtocol.Request(requestCompleteTransfer);
```

If the completion request is successful, you will receive two EventReleased messages and you will no longer be a party to the call.

Closing the Connection

Finally, when you are finished communicating with the T-Server, you should close the connection to minimize resource utilization:

```
[C#]

tServerProtocol.Close();
```