# Bria Desktop API Sample for Java – Quickstart Guide

This guide will introduce the Bria Desktop API Samples for Java and document how to setup and run the sample application. Readers should be familiar with the Bria Desktop API and the Bria Desktop API Developer Guide.

## RemoteApiPhoneGUI Sample Application

The Bria Desktop API Sample for Java includes a Swing-based graphical user interface that demonstrates the capabilities of the Bria Desktop API and a sample implementation written in Java for interacting with the API.

The included sample, RemoteApiPhoneGUI, is bundled as an Eclipse IDE project that can be easily imported into your own workspace. The sample can be found in the Samples directory. To import the project into Eclipse, select Import from the File menu, and then import as Existing Projects into Workspace. It was built to comply with the Java 5 APIs and should also work on any new versions of Java.

There are only two different classes included in the RemoteApiPhoneGUI sample: BriaRemote and BriaModel.

BriaRemote contains all the user interface components and also handles the communication with the Bria Desktop client. The sample utilizes the sample code included in the BriaApiWrapper directory to connect then read and write to the websocket exposed by the Bria Desktop API (this code is discussed later in this document). The most interesting methods of the BriaRemote class are described below:

void initializeClient () – Creates a new BriaApiTransferClient object and registers handlers for various message types.

void requestInitialStatuses() – Requests an initial set of information from Bria Desktop client, including: audio properties, call status, voicemail status, call history, and missed call data.

The BriaModel class holds the last known state of the Bria Desktop client and is updated by the BriaRemote class when new messages are received.

## BriaApiWrapper Sample - WebSocket Reading and Writing

This sample use a class that inherits WebSocketFactory written by Takahiko Kawasaki. For full reference, please check github(https://github.com/TakahikoKawasaki/nv-websocket-client) for the source code. It also uses NaiveSSLContext.java, written by the same author.

The BriaApiWrapper demonstrates one way to connect, read, and write to the websocket exposed by the Bria Desktop API. It is meant to provide a starting point for developers that want to connect, read and write to the Bria Desktop websocket and is not a part of the official Bria Desktop API. The RemoteApiPhoneGUI sample uses this code to interact with the Bria Desktop client.

### BriaApiWrapper Overview

The most important classes required to work with the BriaApiWrapper are found in the com.counterpath.api.bria package. They are:

BriaApiTransferClient

BriaApiTransferClient.BriaApiClientOpenListener

Message

MessageBodyParser

MessageBuilder

MessageHandler

MessageProcessor

Utilities

The BriaApiTransferClient listens for messages and responses sent by a Bria Desktop client via a web socket and it also provides a means to register handlers that are invoked when messages are received. The other components exists to simplify the process of constructing and parsing messages sent and received from a Bria Desktop client.

### Sample Usage

Here is a simple example of how these pieces work together. The numbered lines include a description below.

**public class** BriaRemote **implements** BriaApiClientOpenListener { // #1

URI serverUri = new URI("wss://cpclientapi.softphone.com:9002/counterpath/socketapi/v1/");

BriaApiTransferClient apiClient = new BriaApiTransferClient(serverUri, this);

apiClient.connectSocket(); // #2

MessageProcessor processor = apiClient.getMessageProcessor();

processor.registerHandlerForMessageType(MessageBodyType.RESPONSE\_STATUS\_AUDIO\_PROPERTIES, **new** MessageHandler() { // #3

**public** **void** handle(Message message) {

AudioProperties audioProperties = MessageBodyParser.parseAudioPropertiesStatusResponse(message.getXmlDocument()); // #4

System.out.println("Audio Properties response message");

System.out.println(" speakerMuted:" + audioProperties.isSpeakerMuted());

System.out.println(" microphoneMuted:" + audioProperties.isMicrophoneMuted());

System.out.println(" speakerModeEnabled:" + audioProperties.isSpeakerModeEnabled());

System.out.println(" speakerVolume:" + audioProperties.getSpeakerVolume());

System.out.println(" microphoneVolume:" + audioProperties.getMicrophoneVolume());

}

});

processor.startProcessing(); // #5

Message message = **new** MessageBuilder().status(StatusType.AUDIO\_PROPERTIES).build(); // #6

apiClient.writeMessage(message); // #7

**#1:** One of your classes should implement BriaApiClientOpenListener, which has abstract method onClientConnected(). BriaApiClient that you will create will call this method, when web socket connection has been established.

**#2**: Construct a new BriaApiTransferClient object. Pass on the server URI to connect to, and a class that extends BriaApiClientOpenListener.

**#3**: Registering a MessageHandler for the RESPONSE\_STATUS\_AUDIO\_PROPERTIES response type with the MessageProcessor. When the Bria Desktop client sends this type of response, this MessageHandler’s handle(Message) method will be invoked by the MessageProcessor.

**#4**: Accessing the XML response and using MessageBodyParser to extract typed information. MessageBodyParser has methods to parse each of the different message response types Bria can send.

**#5**: Tell the MessageProcessor to start processing. This should be called once you have registered all of your MessageHandlers.

**#6**: Construct a message to send to the Bria client. In this case, we’re creating a request with MessageBuilder that will ask for information about the audio properties of the client. There are methods within the MessageBuilder class to help you build the requests that are supported by the Bria Desktop API.

**#7**: Write the message to the pipe. The response will be handled with the MessageHandler we registered in #3.

There are a variety of response types you can register MessageHandlers for. See the MessageBodyType enum for a complete list. Generally only the types prefixed with RESPONSE\_ and the BODYLESS type will be useful, but in some cases it may be useful to override the default handlers provided for types prefixed with EVENT\_.

MessageBuilder provides many helpers to aid developers that need to construct requests for communicating with Bria. Explore the MessageBuilder Javadocs to discover what capabilities the BriaApiWrapper exposes for communicating with Bria.