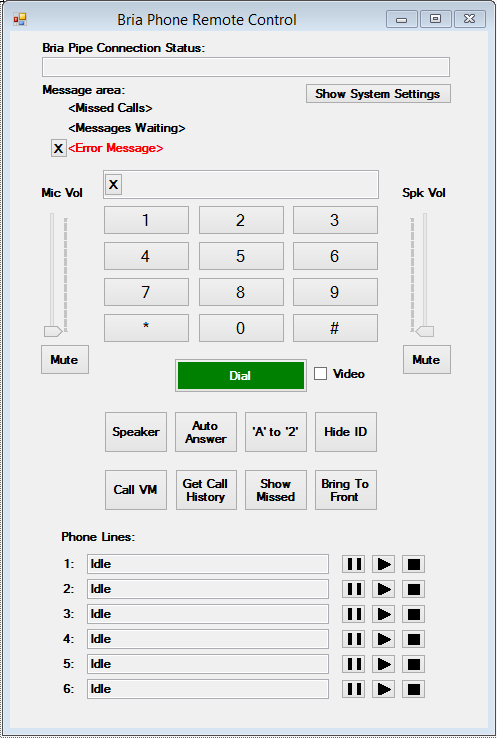
Bria\_API\_SampleApp\_Phone  
  
Sample Application Overview

* Sample application written using Microsoft Visual Studio 2012 in C# , Windows Forms and .NET
* Targeting .NET 4.5 Framework Profile
* Using 3rd party library WebSocket4Net from <https://websocket4net.codeplex.com/>
* C# Source Files in solution (solution file is: Bria\_API\_SampleApp\_Phone.sln):
  + Program.cs (100% auto-generated by project Wizard)
  + CMessageParser.cs (Sample Implementation of Bria Pipe API Message Parsing)
  + BriaPipeAPIWrapper.cs (Sample Implementation of a Wrapper for abstraction)
  + BriaPhoneRemoteControl.cs (Main Sample source, Windows Form application)
  + CallHistoryView.cs (Simple Form used to display Call History data in a List View)

The Main Form has the following areas:

At the top is a Text Box used to show the current pipe connection status against the Bria softphone.

Below that is an area where messages will appear for missed calls, voice mail notifications and error messages. Also found here is a button that will query the “System Settings” from Bria and display the result.

The next block contain a dialpad and audio controls. Numbers can be entered via the dial pad keys and the Dial button will make calls. Enabling the “Video” checkbox before hitting Dial will attempt to make Video calls.

The 8 Function Buttons that follow exercise the following options:  
- Speaker Mode on/off  
- Auto Answer Mode on/off  
- Letters To Numbers Mode on/off  
- Anonymous Mode on/off  
(These 4 all change color to indicate when a Mode is enabled)  
  
- Call VM will execute the API function to place a call to the Voice Mail number configured, if there are messages waiting (as determined by Bria receiving MWI indication)  
- Get Call History will send a query for Call History Status and open a separate window to display the result  
- Show Missed will instruct Bria via the API to switch to the History tab and filter for Missed calls  
- Bring to Front will instruct Bria via the API to come to the Foreground, without otherwise changing its state

WebSocket4Net

This Sample application connects to the Bria Desktop API using a secure web-socket connection to localhost on port 9002.

The web-socket transport functionality is provided in the DLL WebSocket4Net.dll which was written by Kerry Jiang and can be found at <https://websocket4net.codeplex.com/>. It provides a very simple and easy to use interface for web-socket connectivity, which is sufficient for the purposes of this sample. Please check the site for updates, terms of use and license information.

The Sample application uses the following methods from the WebSocket4Net library:

* Open : establishes the connection
* Send : writes a message over the socket connection
* Close : closes the connection

It also consumes the following events form the library:

* Opened : indicate that connection has been established
* MessageReceived : indicate that a message has been received
* Closed : indicate that the connection has been closed
* Error : indicate that an Error occurred (details in the associated Exception)

CMessageParser.cs

This file contain the class CMessageParser that will take a Message received from the Bria Pipe API and parse it into some easily consumable properties. To use this class the consumer needs to create an instance of the CMessageParser class, passing the Message as the only argument to the constructor. Next the consumer will execute the method ‘ParseMessage()’. When this method returns the following properties will be available for further handling of the message:

* MessageType ; *of type ApiMessageType which is an enum of UnknownMessageType, Response, Event or Error.*
* TransactionID ; *of type String, will contain any transaction ID passed in the request that triggered this response.*
* UserAgent ; *of type String, will contain the User-Agent header value in the message*.
* ContentType ; *of type String, will contain the Content-Type header value in the message.*
* ContentLength ; *of type Int32, will contain the Content-Length header value in the message*.
* Content ; *of type String, will contain the User-Agent header value in the message*.

The CMessageParser also parses the content, so it is unlikely that the raw Content will be needed by the consumer. From the content is parsed the following:  
  
(If the MessageType is Event)

* EventType ; *of type ApiEventType which is an enum of UnknownEventType or StatusChange*.

(If the MessageType is Error)

* ErrorCode ; *of type Int32, gives the 4xx or 5xx numeric error code of the error response*.
* ErrorText ; *of type String, gives the textual representation of the error in the response*.

(If the MessageType is Response)

* StatusType ; *of type ApiStatusType which is an enum of UnknownStatusType, PhoneStatus, CallStatus, CallHistoryStatus, MissedCallStatus, VoiceMailStatus, AudioPropertiesStatus, CallOptionsStatus or SystemSettingsStatus*.
* Data ; *which is of type Object, that can be either an ArrayList or a Hashtable depending on the structure of the data that has been parsed from the body of the message*.

When the response data contain a list of elements, like a list of Calls or a list of VoiceMails, then the outer container of the data will be an ArrayList. When the response data is a sequence of Key,Value pairs then the outer container of the data will be a Hashtable.

To facilitate extracting values from the key value pairs CMessageParser defines a long list of string constants in the class CMessageParser.ApiStringConstants as all elements in the hashtable is indexed by a string matching exactly the element name from the data value in the original message body.

Some elements may have other elements or attributes that can be contained in further ArrayLists or Hashtables depending again on the structure (ArrayList for any element that can be repeated more than once).

The values are also all stored as strings, and there are string constants for any enumerated values like “enabled”/”disabled” or “true/false” etc.

BriaAPIWrapper.cs

This file contain the class BriaAPI which is an abstraction layer that turns the heavily textual API messages into more easily consumable events and classes that an actual application can make use of to achieve its functionality in regards to interacting with the Bria API.

The consumer must instantiate the class, usually only once, and hook up to the events that it wishes to handle for its purpose. The events available from this abstraction layer are:

* OnConnected
* OnError
* OnDisconnected
* OnStatusChanged
* OnPhoneStatus
* OnCallStatus
* OnCallOptionsStatus
* OnAudioPropertiesStatus
* OnMissedCallsStatus
* OnVoiceMailStatus
* OnCallHistoryStatus
* OnSystemSettingsStatus
* OnErrorReceived

Details about each here below.

OnConncted, OnError, OnDisconnected

These events refer to the Web-Socket connection status and are basically direct reflections of the events from the WebSocket4Net library.

OnStatusChanged

This event is fired when a StatusChange event is received. The StatusChangedEventArgs will give access to the Property ‘StatusType’ of type *enum StatusTypes.* Typically an application would issue a query for status of whatever element has changed its status (per the StatusType) but there can be cases where the details are not required on an asynchronous basis, and the application can set the property ‘Handled’ to true before exiting the event handler method, which will instruct the API Wrapper that the application will handle this event. If the application does not set the ‘Handled’ property to true then the Wrapper will automatically issue the query for the item which status has changed.

OnPhoneStatus

This event is fired when a response to the query for “phone” status is received. Data elements in the PhoneStatusEventArgs are:

bool IsReady

bool CallsAllowed

AccountStates AccountStatus

short ErrorCode

short MaxLines

tring CallsNotAllowedReason

OnCallStatus

This event is fired when a response to the query for “call” status is received. Data elements in the CallStatusEventArgs are:

List<Call> CallList

The Call class contains:

string CallId

HoldStates HoldState

List<CallParticipant> ParticipantList ss

And the CallParticipant class contains:

string Number

string DisplayName

CallStates CallState

long TimeInitiated

OnCallOptionsStatus

This event is fired when a response to the query for “callOptions” status is received. Data elements in the CallOptionsStatusEventArgs are:

bool IsAnonymousEnabled  
 bool IsLettersToNumbersEnabled   
 bool IsAutoAnswerEnabled

OnAudioPropertiesStatus

This event is fired when a response to the query for “audioProperties” status is received. Data elements in the AudioPropertiesStatusEventArgs are:

bool IsSpeakerModeEnabled

bool IsSpeakerMuted

bool IsMicrophoneMuted

short SpeakerVolume

short MicrophoneVolume

OnMissedCallsStatus

This event is fired when a response to the query for “missedCalls” status is received. Data elements in the MissedCallsStatusEventArgs are:

int Count

OnVoiceMailStatus

This event is fired when a response to the query for “voiceMail” status is received. Data elements in the VoiceMailsStatusEventArgs are:

List<VoiceMailInfo> VoiceMailInfoList

The class VoiceMailInfo contains:

short AccountId

string AccountName

int Count

bool HasNewVoiceMail

OnCallHistoryStatus

This event is fired when a response to the query for “callHistory” status is received. Data elements in the CallHistoryStatusEventArgs are:

List<CallHistoryItem> CallHistoryItemList

The class CallHistoryItem contains:

CallHistoryEntryTypes Type

string Number

string DisplayName

int Duration

long TimeInitiated

OnSystemSettingsStatus

This event is fired when a response to the query for “systemSettings” status is received. Data elements in the SystemSettingsStatusEventArgs are:

CallTypes DefaultCallType

bool CallRightAwayOnceNumberSelected

OnErrorReceived

This event is fired when an Error is received in response to any query. Date elements in the ErrorReceivedEventArgs are:

int ErrorCode

string ErrorText

string TransactionId

In addition to the events, the following methods are available in the abstracted API:

void Start()  
void Stop()  
void RequestSetAnonymousMode(bool enabled)  
void RequestSetAutoAnswerMode(bool enabled)  
void RequestSetLettersToNumbersMode(bool enabled)  
void RequestBringPhoneToFront()  
void RequestPlaceCall(string number, CallTypes type)  
void RequestAnswerCall(string callId, bool withVideo)  
void RequestPutCallOnHold(string callId)  
void RequestResumeCall(string callId)  
void RequestEndCall(string callId)  
void RequestCheckVoiceMail(short accountId)  
void RequestSendDTMF(char key)  
void RequestSetSpeakerMode(bool enabled, bool suppressDialTone)  
void RequestSetSpeakerMute(bool enabled)  
void RequestSetMicrophoneMute(bool enabled)  
void RequestSetSpeakerVolume(short volume)  
void RequestSetMicrophoneVolume(short volume)  
void RequestPhoneStatus()  
void RequestCallStatus()  
void RequestCallHistoryStatus(int maxCount, CallHistoryFilterTypes filter)  
void RequestMissedCallStatus()  
void RequestVoiceMailStatus()  
void RequestAudioPropertiesStatus()  
void RequestCallOptionsStatus()  
void RequestSystemSettingsStatus()  
void RequestBringCallHistoryToFront(CallHistoryFilterTypes type, string filterText)  
void RequestSetCallOptions(string anonymous, string autoAnswer, string lettersToNumbers)  
void RequestSetAudioProperties(string mute, string speakerMute, string speaker, bool suppressDialTone, string speakerVolume, string micVolume)