

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Windows.Media.SpeechRecognition;
5 using Windows.Devices.Spi;
6 using System.Text;
7 using System.Threading;
8 using System.Threading.Tasks;
9 using Windows.Devices.Enumeration;
10
11 namespace RaspiHomeSpeechNSynthesize
12 {
13     public class Speecher
14     {
15         #region Fields
16         #region Constants
17         #endregion
18
19         #region Variable
20         private Synthesizer _rhSynt;
21         private Commands _rhCommands;
22         private CommunicationWithServer _comWithServer;
23
24         private SpeechRecognizer _recoEngine = null;
25         private SpiDevice _mcp3202;
26
27         private bool _isRaspiCalled = false;
28         #endregion
29         #endregion
30
31         #region Properties
32         public Synthesizer RhSynt
33         {
34             get
35             {
36                 return _rhSynt;
37             }
38
39             set
40             {
41                 _rhSynt = value;
42             }
43         }
44
45         public bool IsRaspiCalled
46         {
47             get
48             {
49                 return _isRaspiCalled;
50             }
51
52             set
53             {
54                 _isRaspiCalled = value;
55             }
56         }
57     }
58 }
```

```
57
58     public CommunicationWithServer ComWithServer
59     {
60         get
61         {
62             return _comWithServer;
63         }
64
65         set
66         {
67             _comWithServer = value;
68         }
69     }
70
71     public Commands RhCommands
72     {
73         get
74         {
75             return _rhCommands;
76         }
77
78         set
79         {
80             _rhCommands = value;
81         }
82     }
83 #endregion
84
85 #region Constructor
86 /// <summary>
87 /// Constructor: Initialize
88 /// </summary>
89 public Speecher()
90 {
91     // Initialize the synthesizer
92     this.RhSynt = new Synthesizer(this);
93     this.RhCommands = new Commands();
94
95     // Initialize the communication with the server
96     this.ComWithServer = new CommunicationWithServer(this);
97
98     // Create the speech recognition object
99     this._recoEngine = new SpeechRecognizer();
100
101     // Initialize the recognition
102     InitializeSpeechRecognizer();
103 }
104 #endregion
105
106 #region Methods
107 /// <summary>
108 /// Initialize the Spi communication
109 /// </summary>
110 private async void InitializeSpi()
111 {
112     //using SPI0 on the Pi
```

```
113         var spiSettings = new SpiConnectionSettings(0); //for spi bus index 0
114         spiSettings.ClockFrequency = 3600000; //3.6 MHz
115         spiSettings.Mode = SpiMode.Mode0;
116
117         string spiQuery = SpiDevice.GetDeviceSelector("SPI0");
118
119         var deviceInfo = await DeviceInformation.FindAllAsync(spiQuery);
120         if (deviceInfo != null && deviceInfo.Count > 0)
121         {
122             _mcp3202 = await SpiDevice.FromIdAsync(deviceInfo[0].Id, spiSettings);
123         }
124     }
125
126     /// <summary>
127     /// Read digital input with SPI
128     /// </summary>
129     /// <returns></returns>
130     private string ReadSpiData()
131     {
132         byte[] transmitBuffer = new byte[3] { 0x01, 0x80, 0 };
133         byte[] receiveBuffer = new byte[3];
134
135         string result = "";
136
137         _mcp3202.TransferFullDuplex(transmitBuffer, receiveBuffer);
138
139         return result = Encoding.UTF8.GetString(receiveBuffer);
140     }
141
142     /// <summary>
143     /// (obsolete on UWP) Set the configuration of the speecher
144     /// </summary>
145     private void InitializeSpeechRecognizer()
146     {
147
148     }
149
150     /// <summary>
151     /// (obsolete on UWP) Enable the speech, used when raspi is not talking
152     /// </summary>
153     public void EnableSpeech()
154     {
155         //this._recoEngine.RecognizeAsync(RecognizeMode.Multiple);
156     }
157
158     /// <summary>
159     /// (obsolete on UWP) Disable the speech, used when raspi is talking
160     /// </summary>
161     public void DisableSpeech()
162     {
163         // this._recoEngine.RecognizeAsyncStop();
164     }
165
```

```
166     /// <summary>
167     /// Used to call raspi
168     /// </summary>
169     /// <param name="nameMentioned"></param>
170     private void CallRaspi(string nameMentioned)
171     {
172         DisableSpeech();
173
174         this.RhSynt.RaspiCalled(nameMentioned);
175
176         EnableSpeech();
177     }
178
179     /// <summary>
180     /// Used to send the command after raspi called
181     /// </summary>
182     /// <param name="brutCommand"></param>
183     public void SendBrutCommand(string brutCommand)
184     {
185         DisableSpeech();
186
187         this.ComWithServer.SendCommandToServer(brutCommand);
188     }
189
190     /// <summary>
191     /// Reply message to the sender (reply message error if command
192     /// unknown)
193     /// </summary>
194     /// <param name="messageReply"></param>
195     /// <param name="messageCommand"></param>
196     public void ReplyForSynthesize(string messageReply, string
197     messageCommand)
198     {
199         if (messageReply == "ERROR_MESSAGE")
200         {
201             this.RhSynt.WrongCommand();
202         }
203         else
204         {
205             this.RhSynt.RaspiSayInformation
206                 (this.RhSynt.SetProperlyInformations(messageReply,
207                 messageCommand));
208             this.IsRaspiCalled = false;
209             EnableSpeech();
210         }
211     }
212 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Globalization;
4 using System.Linq;
5 using System.Text;
6 using Windows.Media.SpeechSynthesis;
7 using System.Threading;
8 using System.Threading.Tasks;
9 using Windows.UI.Xaml.Controls;
10 using Windows.UI.Core;
11
12 namespace RaspiHomeSpeechNSynthesize
13 {
14     public class Synthesizer
15     {
16         #region Fields
17         #region Constants
18         private const string RASPI_NAME = "raspi";
19         private const char SEPARATOR = ' ';
20         // Change value when new update ("en" to "fr")
21         private const string LANGUAGE_SELECTION = "en";
22         private const int TIME_TO_WAIT = 3;
23         #endregion
24
25         #region Variable
26         private Speecher _rhSpeech;
27
28         private Commands _rhCommands;
29
30         private Random _rnd;
31         private SpeechSynthesizer _voice;
32         private string _commandReceiveStr = "";
33         private string _commandToSend = "";
34
35         private bool _isCalled = false;
36         private bool _isCompleted = false;
37         private List<string> _lstSentenceSplited;
38         #endregion
39         #endregion
40
41         #region Properties
42         public Speecher RhSpeech
43         {
44             get
45             {
46                 return _rhSpeech;
47             }
48
49             set
50             {
51                 _rhSpeech = value;
52             }
53         }
54
55         public string CommandReceiveStr
56         {
```

```
57         get
58         {
59             return _commandReceiveStr;
60         }
61
62         set
63         {
64             _commandReceiveStr = value;
65         }
66     }
67
68     public bool IsCalled
69     {
70         get
71         {
72             return _isCalled;
73         }
74
75         set
76         {
77             _isCalled = value;
78         }
79     }
80
81     public bool IsCompleted
82     {
83         get
84         {
85             return _isCompleted;
86         }
87
88         set
89         {
90             _isCompleted = value;
91         }
92     }
93
94     public Commands RhCommands
95     {
96         get
97         {
98             return _rhCommands;
99         }
100
101         set
102         {
103             _rhCommands = value;
104         }
105     }
106     #endregion
107
108     #region Constructor
109     /// <summary>
110     /// Constructor: Initialize
111     /// </summary>
112     public Synthesizer(Speecher paramSpeecher)
```

```
113     {
114         this.RhSpeech = paramSpeecher;
115         this.RhCommands = new Commands();
116         this._rnd = new Random();
117         this._voice = new SpeechSynthesizer();
118         this._1stSentenceSplited = new List<string>();
119     }
120     #endregion
121
122     #region Methods
123     /// <summary>
124     /// Raspberry processus, wait calling to start communication with server
125     /// </summary>
126     private void RaspiProcessus()
127     {
128         if (this.IsCalled)
129         {
130             SendCommand();
131         }
132     }
133
134     /// <summary>
135     /// Send to the synthesize method the order to reply
136     /// </summary>
137     /// <param name="name"></param>
138     public void RaspiCalled(string name)
139     {
140         string raspiName = RemoveDiacritics(name).ToLower();
141
142         if (raspiName == RASPI_NAME.ToLower())
143         {
144             RaspiCalled(this.RhCommands.WhenCalling);
145             this.RhSpeech.IsRaspiCalled = true;
146             this.IsCalled = true;
147         }
148     }
149
150     /// <summary>
151     /// Send the instruction for the Raspberry
152     /// </summary>
153     /// <returns></returns>
154     public string SendCommand()
155     {
156         this.RhSpeech.IsRaspiCalled = false;
157         this.IsCalled = false;
158
159         this.IsCompleted = true;
160         return this._commandToSend;
161     }
162
163     /// <summary>
164     /// Processus to choose the sentence to say
165     /// </summary>
166     /// <param name="repertory"> List of sentence to say </param>
167     private void RaspiCalled(List<string> repertory)
```

```
168     {
169         string messageToSay = repertory[_rnd.Next(0, repertory.Count - 1)];
170
171         this.RaspiTalk(messageToSay);
172     }
173
174     /// <summary>
175     /// Allow the raspi to let her talk
176     /// </summary>
177     /// <param name="messageToSay"> sentence to say </param>
178     private async void RaspiTalk(string messageToSay)
179     {
180         // Get the output element (audio jack)
181         MediaElement mediaElement = new MediaElement();
182         SpeechSynthesizer synth = new SpeechSynthesizer();
183
184         // Set the default language
185         foreach (VoiceInformation vInfo in SpeechSynthesizer.AllVoices)
186         {
187             if (vInfo.Language.Contains(LANGUAGE_SELECTION))
188             {
189                 synth.Voice = vInfo;
190                 break;
191             }
192             else
193                 synth.Voice = vInfo;
194         }
195
196         SpeechSynthesisStream synthStream = await
197             synth.SynthesizeTextToStreamAsync(messageToSay);
198
199         mediaElement.SetSource(synthStream, synthStream.ContentType);
200         // 0 = min / 1 = max
201         mediaElement.Volume = 1;
202         mediaElement.Play();
203
204         // Work like Thread.Sleep(TIME_TO_WAIT)
205         await Task.Delay(TimeSpan.FromSeconds(TIME_TO_WAIT));
206     }
207
208     /// <summary>
209     /// Called when there is any error
210     /// </summary>
211     public void WrongCommand()
212     {
213         // Reach the error resquest sentences to say
214         this.RaspiCalled(this.RhCommands.SpeecherRespondingRequestError);
215     }
216
217     /// <summary>
218     /// Allow the Raspi, to let her talk with list of information
219     /// </summary>
220     /// <param name="informationsToGive"></param>
221     public void RaspiSayInformation(List<string> informationsToGive)
222     {
```



[illegible]

```
this.RhCommands.SpeecherRespondingEtatRequest[5]);
274         }
275         break;
276         case "PRES":
277             if (pres)
278             {
279                 result.Add
                                     (this.RhCommands.SpeecherRespondingEtatRequest[6] +
                                     information.Split('=').Last() +
280                                     281
this.RhCommands.SpeecherRespondingEtatRequest[7]);
282             }
283             break;
284         }
285     }
286 }
287 else
288     result.Add("");
289 return result;
290 }
291
292 /// <summary>
293 /// Stack Overflow solution to delete accents in strings
294 /// http://stackoverflow.com/questions/249087/how-do-i-remove-
                                     diacritics-accents-from-a-string-in-net
295 /// </summary>
296 /// <param name="str"></param>
297 /// <returns></returns>
298 static string RemoveDiacritics(string str)
299 {
300     var normalizedString = str.Normalize(NormalizationForm.FormD);
301     var stringBuilder = new StringBuilder();
302
303     foreach (var c in normalizedString)
304     {
305         var unicodeCategory = CharUnicodeInfo.GetUnicodeCategory(c);
306         if (unicodeCategory != UnicodeCategory.NonSpacingMark)
307         {
308             stringBuilder.Append(c);
309         }
310     }
311
312     return stringBuilder.ToString().Normalize
                                     (NormalizationForm.FormC);
313 }
314 #endregion
315 }
316 }
317
```

```
1 using RaspiHomeSpeechNSynthesize;
2 using System;
3 using System.Collections.Generic;
4 using System.Diagnostics;
5 using System.Linq;
6 using System.Net;
7 using System.Net.Sockets;
8 using System.Text;
9 using System.Threading;
10 using System.Threading.Tasks;
11 using Windows.ApplicationModel.Core;
12 using Windows.Networking;
13 using Windows.Networking.Connectivity;
14 using Windows.Networking.Sockets;
15 using Windows.Storage.Streams;
16
17 namespace RaspiHomeSpeechNSynthesize
18 {
19     public class CommunicationWithServer
20     {
21         #region Fields
22         #region Constants
23         // Default information to connect on the server
24         private const int PORT = 54565;
25         //// Need to be changed fo each configuration
26         private const string IPSEVER = "10.134.97.117";// "192.168.2.8";
27
28         private const string FORMATSTRING = "IPRasp={0};Location=
29             {1};Component={2}";
30         private const string COMMUNICATIONSEPARATOR = "@";
31
32         // Important need to be changed if it's another room!
33         private const string LOCATION = "Salon";
34         private const string COMPONENT = "Microphone";
35         private const string RPINAME = "Microphone_" + LOCATION;//
36             "192.168.2.8";
37
38         private const int MESSAGE_FULL LENGHT = 512;
39         #endregion
40
41         #region Variables
42         private Speecher _speecher;
43
44         private StreamSocket _socket = new StreamSocket();
45         private StreamSocketListener _listener = new StreamSocketListener();
46         private List<StreamSocket> _connections = new List<StreamSocket>();
47         private bool _isConnected = false;
48         private bool _connecting = false;
49
50         private string _messageCommand = "";
51         #endregion
52         #endregion
53
54         #region Properties
55         public Speecher Speecher
56         {
57             {
```

```
55         get
56         {
57             return _speecher;
58         }
59
60         set
61         {
62             _speecher = value;
63         }
64     }
65
66     public StreamSocket Socket
67     {
68         get
69         {
70             return _socket;
71         }
72
73         set
74         {
75             _socket = value;
76         }
77     }
78
79     public StreamSocketListener Listener
80     {
81         get
82         {
83             return _listener;
84         }
85
86         set
87         {
88             _listener = value;
89         }
90     }
91
92     public List<StreamSocket> Connections
93     {
94         get
95         {
96             return _connections;
97         }
98
99         set
100        {
101            _connections = value;
102        }
103    }
104
105     public bool IsConnected
106     {
107         get
108         {
109             return _isConnected;
110         }
111     }
```

```
111
112         set
113         {
114             _isConnected = value;
115         }
116     }
117
118     public bool Connecting
119     {
120         get
121         {
122             return _connecting;
123         }
124
125         set
126         {
127             _connecting = value;
128         }
129     }
130
131     public string MessageCommand
132     {
133         get
134         {
135             return _messageCommand;
136         }
137
138         set
139         {
140             _messageCommand = value;
141         }
142     }
143     #endregion
144
145     #region Constructors
146     /// <summary>
147     /// Constructor: Initializer
148     /// </summary>
149     /// <param name="paramModel"></param>
150     public CommunicationWithServer(Speecher paramModel)
151     {
152         this.Speecher = paramModel;
153
154         Connect();
155     }
156     #endregion
157
158     #region Methods
159     /// <summary>
160     /// Connect the raspberry to the server
161     /// </summary>
162     private async void Connect()
163     {
164         try
165         {
166             this.Connecting = true;
```

```

167         await this.Socket.ConnectAsync(new HostName(IPSERVER),
168             PORT.ToString());
169         SendForInitialize();
170         this.Connecting = false;
171         this.IsConnected = true;
172
173         WaitForData(this.Socket);
174     }
175     catch (Exception)
176     {
177         this.Connecting = false;
178         this.IsConnected = false;
179     }
180 }
181
182 /// <summary>
183 /// Listen the traffic on the port
184 /// </summary>
185 private async void Listen()
186 {
187     this.Listener.ConnectionReceived += listenerConnectionReceived;
188     await this.Listener.BindServiceNameAsync(PORT.ToString());
189 }
190
191 void listenerConnectionReceived(StreamSocketListener sender,
192     StreamSocketListenerConnectionReceivedEventArgs args)
193 {
194     this.Connections.Add(args.Socket);
195
196     WaitForData(args.Socket);
197 }
198
199 /// <summary>
200 /// Send the message in input to output
201 /// </summary>
202 /// <param name="socket"></param>
203 /// <param name="message"></param>
204 private async void SendMessage(StreamSocket socket, string message)
205 {
206     DataWriter dataWriter = new DataWriter(socket.OutputStream);
207     var len = dataWriter.MeasureString(message); // Gets the UTF-8
208         string length.
209     dataWriter.WriteInt32((int)len);
210     dataWriter.WriteString(message);
211     var ret = await dataWriter.StoreAsync();
212     dataWriter.DetachStream();
213 }
214
215 /// <summary>
216 /// Send to initialize the raspberry to the server
217 /// </summary>
218 private void SendForInitialize()
219 {
220     SendMessage(this.Socket, string.Format(COMMUNICATIONSEPARATOR +
221         RPINAME + COMMUNICATIONSEPARATOR + "Connection:" + FORMATSTRING,
222         GetHostName(), LOCATION, COMPONENT));

```

```
218     }
219
220     /// <summary>
221     /// Send the command to the server
222     /// </summary>
223     public void SendCommandToServer(string message)
224     {
225         SendMessage(this.Socket, COMMUNICATIONSEPARATOR + "Send:" +
226             message);
227         this.MessageCommand = message;
228     }
229
230     /// <summary>
231     /// Wait data readed if exist
232     /// </summary>
233     /// <param name="socket"></param>
234     private async void WaitForData(StreamSocket socket)
235     {
236         DataReader dataReader = new DataReader(socket.InputStream);
237         dataReader.InputStreamOptions = InputStreamOptions.Partial;
238         var messageLenght = dataReader.UnconsumedBufferLength;
239         uint stringBytes = messageLenght;
240
241         try
242         {
243             // Read modification in the stream
244             stringBytes = await dataReader.LoadAsync(MESSAGE_FULL LENGHT);
245
246             // read message
247             string messageRead = dataReader.ReadString(stringBytes);
248
249             // Send in return if the value exist
250             if (messageRead != "")
251             {
252                 this.Speecher.ReplyForSynthesize(messageRead,
253                     this.MessageCommand);
254             }
255
256             messageRead = "";
257         }
258         catch (Exception ) { }
259
260         WaitForData(socket);
261     }
262
263     /// <summary>
264     /// Get the ip of the raspberry
265     /// </summary>
266     /// <returns>return a string like 192.168.1.2</returns>
267     public string GetHostName()
268     {
269         List<string> IpAddress = new List<string>();
270         var Hosts =
271             Windows.Networking.Connectivity.NetworkInformation.GetHostNames
272             ().ToList();
273         foreach (var Host in Hosts)
```

```
270         {
271             string IP = Host.DisplayName;
272             IPAddress.Add(IP);
273         }
274         return IPAddress.Last();
275     }
276 #endregion
277 }
278 }
279
```



```
1 <Page
2   x:Class="RaspiHomeSpeechNSynthesize.MainPage"
3   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5   xmlns:local="using:RaspiHomeSpeechNSynthesize"
6   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8   mc:Ignorable="d">
9
10  <StackPanel Orientation="Horizontal" Background="White"
11    HorizontalAlignment="Center" VerticalAlignment="Center">
12    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
13      Margin="0,0,50,0">
14      <StackPanel Orientation="Horizontal">
15        <TextBox x:Name="tbxLightCommand1" Width="250" Text="Allume la
16          lumière du salon"/>
17        <Button x:Name="btnLightCommand1" Width="55" Content="Send"
18          Margin="40,0,0,0" Click="btnLightCommand1_Click"/>
19      </StackPanel>
20
21      <StackPanel Orientation="Horizontal" Margin="0,40,0,0">
22        <TextBox x:Name="tbxLightCommand2" Width="250" Text="Éteins la
23          lumière du salon"/>
24        <Button x:Name="btnLightCommand2" Width="55" Content="Send"
25          Margin="40,0,0,0" Click="btnLightCommand2_Click"/>
26      </StackPanel>
27
28      <StackPanel Orientation="Horizontal" Margin="0,40,0,0">
29        <TextBox x:Name="tbxState" Width="250" Text="Quel est la
30          température du salon"/>
31        <Button x:Name="btnState" Width="55" Content="Send"
32          Margin="40,0,0,0" Click="btnState_Click"/>
33      </StackPanel>
34    </StackPanel>
35
36    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
37      Margin="0,0,50,0">
38      <StackPanel Orientation="Horizontal">
39        <TextBox x:Name="tbxStore1" Width="250" Text="Monte le store du
40          salon"/>
41        <Button x:Name="btnSendCommand" Width="55" Content="Send"
42          Margin="40,0,0,0" Click="btnStore1_Click"/>
43      </StackPanel>
44
45      <StackPanel Orientation="Horizontal" Margin="0,40,0,0">
46        <TextBox x:Name="tbxStore2" Width="250" Text="Descends le store
47          du salon"/>
48        <Button x:Name="btnStore2" Width="55" Content="Send"
49          Margin="40,0,0,0" Click="btnStore2_Click"/>
50      </StackPanel>
51
52      <StackPanel Orientation="Horizontal" Margin="0,40,0,0">
53        <TextBox x:Name="tbxStore3" Width="250" Text="Stop le store du
54          salon"/>
55        <Button x:Name="btnStore3" Width="55" Content="Send"
56          Margin="40,0,0,0" Click="btnStore3_Click"/>
57      </StackPanel>
58    </StackPanel>
59  </StackPanel>
60 </Page>
```

```
42         </StackPanel>
43     </StackPanel>
44 </StackPanel>
45 </Page>
46
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Runtime.InteropServices.WindowsRuntime;
6 using Windows.Foundation;
7 using Windows.Foundation.Collections;
8 using Windows.UI.Xaml;
9 using Windows.UI.Xaml.Controls;
10 using Windows.UI.Xaml.Controls.Primitives;
11 using Windows.UI.Xaml.Data;
12 using Windows.UI.Xaml.Input;
13 using Windows.UI.Xaml.Media;
14 using Windows.UI.Xaml.Navigation;
15
16 // Pour plus d'informations sur le modèle d'élément Page vierge, consultez la page http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409
17
18 namespace RaspiHomeSpeechNSynthetize
19 {
20     /// <summary>
21     /// Une page vide peut être utilisée seule ou constituer une page de destination au sein d'un frame.
22     /// </summary>
23     public sealed partial class MainPage : Page
24     {
25         #region Fields
26         private Speecher _speecher;
27         #endregion
28
29         #region Constructor
30         public MainPage()
31         {
32             this.InitializeComponent();
33
34             this._speecher = new Speecher();
35         }
36         #endregion
37
38         #region Event
39         private void btnLightCommand1_Click(object sender, RoutedEventArgs e)
40         {
41             if (this.tbxLightCommand1.Text != "")
42                 this._speecher.SendBrutCommand(this.tbxLightCommand1.Text);
43             else
44                 this._speecher.SendBrutCommand("Allumer lumière");
45         }
46
47         private void btnLightCommand2_Click(object sender, RoutedEventArgs e)
48         {
49             if (this.tbxLightCommand2.Text != "")
50                 this._speecher.SendBrutCommand(this.tbxLightCommand2.Text);
51             else
52                 this._speecher.SendBrutCommand("Eteindre lumière");
53         }
54     }
```

```
55     private void btnState_Click(object sender, RoutedEventArgs e)
56     {
57         if (this.tbxState.Text != "")
58             this._speecher.SendBrutCommand(this.tbxState.Text);
59         else
60             this._speecher.SendBrutCommand("Temperature du salon");
61     }
62
63     private void btnStore1_Click(object sender, RoutedEventArgs e)
64     {
65         if (this.tbxStore1.Text != "")
66             this._speecher.SendBrutCommand(this.tbxStore1.Text);
67         else
68             this._speecher.SendBrutCommand("Monter store");
69     }
70
71     private void btnStore2_Click(object sender, RoutedEventArgs e)
72     {
73         if (this.tbxStore2.Text != "")
74             this._speecher.SendBrutCommand(this.tbxStore2.Text);
75         else
76             this._speecher.SendBrutCommand("Descendre store");
77     }
78
79     private void btnStore3_Click(object sender, RoutedEventArgs e)
80     {
81         if (this.tbxStore3.Text != "")
82             this._speecher.SendBrutCommand(this.tbxStore3.Text);
83         else
84             this._speecher.SendBrutCommand("Stopper store");
85     }
86     #endregion
87 }
88 }
89
```

```
1 using System.Collections.Generic;
2
3 namespace RaspiHomeSpeechNSynthesize
4 {
5     public class Commands
6     {
7         #region Commands variable
8         private List<string> _whenCalling = new List<string>()
9         {
10             "Oui, que puis-je faire pour vous?", "Comment puis-je vous servir?", "Je suis toute ouïe"
11         };
12
13         private List<string> _whenCallingError = new List<string>()
14         {
15             "Je ne m'appel pas comme ça!", "Je crois que vous vous êtes trompé de personne!"
16         };
17
18         private List<string> _speakerRespondingRequest = new List<string>()
19         {
20             "Tout de suite", "Je m'y mets de ce pas!", "Ne quittez pas!"
21         };
22
23         private List<string> _speakerRespondingRequestError = new List<string>()
24         {
25             "Je n'ai pas compris ce que vous avez demandé"
26         };
27
28         private List<string> _raspiHomeObjectKnown = new List<string>()
29         {
30             "lumiere", "lumières",
31             "store", "stores",
32             "television", "televisions",
33             "porte", "portes",
34             "fenetre", "fenêtres",
35         };
36
37         private List<string> _raspiHomeActionKnown = new List<string>()
38         {
39             "allumer", "allume",
40             "eteindre", "eteins",
41             "monter", "monte",
42             "descendre", "descends",
43             "stopper", "stop",
44             "ouvrir", "ouvre",
45             "fermer", "ferme",
46         };
47
48         private List<string> _speakerRespondingEtatRequest = new List<string>()
49         {
50             "Il fait ", " degrés Celsius ", " degrés Farad", " degrés Kelvin", "Le taux d'humidité est de ", " pourcent", "La pression est actuellement de ", " hecto Pascal"
```

```

51     };
52
53     private List<string> _raspiHomeActionWithoutObjectKnown = new List<string>()
54     {
55         "temperature", "humidite", "pression", "etat"
56     };
57
58     private List<string> _raspiHomeLocationKnown = new List<string>()
59     {
60         "maison", "salon", "cuisine", "parent", "enfant", "bureau", "toilette", "bain"
61     };
62
63     private List<string> _raspiHomeCommandUselessConnecter = new List<string>()
64     {
65         "le", "la", "les", "un", "une", "des", "mon", "ma", "mes", "son", "sa", "ses", "ce", "cet", "cette", "ces", "cel", "celle", "du", "de"
66     };
67
68     private List<string> _raspiHomeGrammarCommand = new List<string>()
69     {
70         "allume la lumière", "allume les lumières", "allumer la lumière", "allumer les lumières",
71         "éteint la lumière", "éteint les lumières", "éteindre la lumière", "éteindre les lumières",
72         "monte le store", "monte les stores", "monter le store", "monter les stores",
73         "descend le store", "descend les stores", "descendre le store", "descendre les stores",
74         "ouvre le store", "ouvre les stores", "ouvrir le store", "ouvrir les stores",
75         "ferme le store", "ferme les stores", "fermer le store", "fermer les stores",
76         "quelle est la température", "quelle est l'humidité", "quelle est la pression", "quelle est l'état",
77         "de la maison", "de la cuisine", "du salon", "de la salle de bain", "de la chambre", "de la pièce",
78     };
79     #endregion
80
81     #region Properties
82     public List<string> WhenCalling
83     {
84         get
85         {
86             return _whenCalling;
87         }
88         set
89         {
90             _whenCalling = value;
91         }
92     }
93 
```

```
94
95     public List<string> WhenCallingError
96     {
97         get
98         {
99             return _whenCallingError;
100         }
101
102         set
103         {
104             _whenCallingError = value;
105         }
106     }
107
108     public List<string> SpeecherRespondingRequest
109     {
110         get
111         {
112             return _speecherRespondingRequest;
113         }
114
115         set
116         {
117             _speecherRespondingRequest = value;
118         }
119     }
120
121     public List<string> SpeecherRespondingRequestError
122     {
123         get
124         {
125             return _speecherRespondingRequestError;
126         }
127
128         set
129         {
130             _speecherRespondingRequestError = value;
131         }
132     }
133
134     public List<string> RaspiHomeObjectKnown
135     {
136         get
137         {
138             return _raspiHomeObjectKnown;
139         }
140
141         set
142         {
143             _raspiHomeObjectKnown = value;
144         }
145     }
146
147     public List<string> RaspiHomeActionKnown
148     {
149         get
```

```
150         {
151             return _raspiHomeActionKnown;
152         }
153
154         set
155         {
156             _raspiHomeActionKnown = value;
157         }
158     }
159
160     public List<string> SpeecherRespondingEtatRequest
161     {
162         get
163         {
164             return _speecherRespondingEtatRequest;
165         }
166
167         set
168         {
169             _speecherRespondingEtatRequest = value;
170         }
171     }
172
173     public List<string> RaspiHomeActionWithoutObjectKnown
174     {
175         get
176         {
177             return _raspiHomeActionWithoutObjectKnown;
178         }
179
180         set
181         {
182             _raspiHomeActionWithoutObjectKnown = value;
183         }
184     }
185
186     public List<string> RaspiHomeLocationKnown
187     {
188         get
189         {
190             return _raspiHomeLocationKnown;
191         }
192
193         set
194         {
195             _raspiHomeLocationKnown = value;
196         }
197     }
198
199     public List<string> RaspiHomeCommandUselessConnector
200     {
201         get
202         {
203             return _raspiHomeCommandUselessConnector;
204         }
205     }
```



```
206         set
207         {
208             _raspiHomeCommandUselessConnector = value;
209         }
210     }
211
212     public List<string> RaspiHomeGrammarCommand
213     {
214         get
215         {
216             return _raspiHomeGrammarCommand;
217         }
218
219         set
220         {
221             _raspiHomeGrammarCommand = value;
222         }
223     }
224     #endregion
225 }
226 }
227
```

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <grammar version="1.0" xml:lang="fr-FR" mode="voice" root="toplevel"
3 tag-format="semantics/1.0"
4 xml:base="http://www.contoso.com/"
5 xmlns="http://www.w3.org/2001/06/grammar"
6 xmlns:sapi="http://schemas.microsoft.com/Speech/2002/06/SGSExtensions">
7 <rule id="toplevel" scope="public">
8   <one-of>
9     <item>
10       <ruleref uri="#Raspi"/>
11     </item>
12     <item>
13       <ruleref uri="#Meteo"/>
14     </item>
15     <item>
16       <ruleref uri="#Action"/>
17     </item>
18   </one-of>
19 </rule>
20
21 <rule id="Raspi">
22   <example> raspi </example>
23
24   <item> raspi </item>
25 </rule>
26
27 <rule id="Meteo">
28   <example> quel est la temperature </example>
29   <example> quel est le taux d'humidite </example>
30   <example> quel est le niveau de pression </example>
31   <example> quel est l'etat</example>
32
33   <one-of>
34     <item> temperature </item>
35     <item> humidite </item>
36     <item> pression </item>
37     <item> etat </item>
38   </one-of>
39   <ruleref uri="#Localisation"/>
40 </rule>
41
42 <rule id="Action">
43   <example> allume la lumiere</example>
44   <example> eteint la lumiere</example>
45   <example> monte le store</example>
46   <example> descend le store</example>
47   <example> ouvre le store</example>
48   <example> ferme le store</example>
49
50   <one-of>
51     <item> allume </item>
52     <item> eteint </item>
53     <item> monte </item>
54     <item> descend </item>
55     <item> ouvre </item>
56     <item> ferme </item>
```

```
57     </one-of>
58
59     <ruleref uri="#Composant"/>
60 </rule>
61
62 <rule id="Composant">
63     <example> la lumiere </example>
64     <example> les lumieres </example>
65     <example> le store </example>
66     <example> les stores </example>
67
68     <one-of>
69         <item> lumiere </item>
70         <item> store </item>
71     </one-of>
72     <ruleref uri="#Localisation"/>
73 </rule>
74
75 <rule id="Localisation">
76     <example> du salon </example>
77     <example> de la cuisine </example>
78     <example> du bureau </example>
79     <example> de la chambre </example>
80     <example> des toilettes </example>
81     <example> de la salle de bain </example>
82     <example> de la maison </example>
83
84     <item repeat="0-1"> salon </item>
85     <item repeat="0-1"> cuisine </item>
86     <item repeat="0-1"> bureau </item>
87     <item repeat="0-1"> chambre </item>
88     <item repeat="0-1"> toilette </item>
89     <item repeat="0-1"> salle de bain </item>
90     <item repeat="0-1"> maison </item>
91 </rule>
92 </grammar>
```