```
 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 RaspiHomeSpeechNSynthetize
13  {
14      public class Synthetizer
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 Synthetizer(Speecher paramSpeecher)
```

```
113         {
114             this.RhSpeech = paramSpeecher;
115             this.RhCommands = new Commands();
116             this._rnd = new Random();
117             this._voice = new SpeechSynthesizer();
118             this._lstSentenceSplited = 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 synthetize 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
                  synth.SynthesizeTextToStreamAsync(messageToSay);
197
198             mediaElement.SetSource(synthStream, synthStream.ContentType);
199             // 0 = min / 1 = max
200             mediaElement.Volume = 1;
201             mediaElement.Play();
202
203             // Work like Thread.Sleep(TIME_TO_WAIT)
204             await Task.Delay(TimeSpan.FromSeconds(TIME_TO_WAIT));
205         }
206
207         /// <summary>
208         /// Called when there is any error
209         /// </summary>
210         public void WrongCommand()
211         {
212             // Reach the error resquest sentences to say
213             this.RaspiCalled(this.RhCommands.SpeecherRespondingRequestError);
214         }
215
216         /// <summary>
217         /// Allow the Raspi, to let her talk with list of information
218         /// </summary>
219         /// <param name="informationsToGive"></param>
220         public void RaspiSayInformation(List<string> informationsToGive)
221         {
```

```
222             foreach (string informationToSay in informationsToGive)
223             {
224                 if (informationToSay != "")
225                 {
226                     this.RaspiTalk(informationToSay);
227                 }
228             }
229         }
230 
231         /// <summary>
232         /// Set information to sythetize
233         /// </summary>
234         public List<string> SetProprelyInformations(string messageReply,
              string messageCommand)
235         {
236             List<string> result = new List<string>();
237 
238             bool temp = false, humi = false, pres = false;
239 
240             if (messageCommand.Contains("température"))
241                 temp = true;
242             if (messageCommand.Contains("humidité"))
243                 humi = true;
244             if (messageCommand.Contains("pression"))
245                 pres = true;
246             if (messageCommand.Contains("état"))
247             {
248                 temp = true;
249                 humi = true;
250                 pres = true;
251             }
252 
253             string[] informationSplited = messageReply.Split(';');
254             if (informationSplited[0] != "")
255             {
256                 foreach (string information in informationSplited)
257                 {
258                     switch (information.Split('=').First())
259                     {
260                         case "TEMP":
261                             if (temp)
262                             {
263                                 result.Add
                        (this.RhCommands.SpeecherRespondingEtatRequest[0] +
264                                         information.Split('=').Last() +
265 
    this.RhCommands.SpeecherRespondingEtatRequest[1]);
266                             }
267                             break;
268                         case "HUMI":
269                             if (humi)
270                             {
271                                 result.Add
                        (this.RhCommands.SpeecherRespondingEtatRequest[4] +
272                                         information.Split('=').Last() +
273 
```

```
    this.RhCommands.SpeecherRespondingEtatRequest[5]);
274                             }
275                             break;
276                         case "PRES":
277                             if (pres)
278                             {
279                                 result.Add
                        (this.RhCommands.SpeecherRespondingEtatRequest[6] +
280                                     information.Split('=').Last() +
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
```