Using SSML for advanced text-to-speech on Windows Phone 8

This article demonstrates how the Speech Synthesis Markup Language (SSML) can be used to provide advanced text-to-speech (TTS) functionality on Windows Phone 8 applications.

Note: This article was a winner in the Windows Phone 8 Wiki Competition 2012Q4.

Introduction

Microsoft introduced a number of interesting APIs and features in the field of speech in Windows Phone 8, which include voice commands, speech recognition and text-to-speech (TTS).

Although this article explains how to add basic TTS functionality to an application, it mainly focuses on the use of the Speech Synthesis Markup Language (SSML) that is supported by Windows Phone 8, to allow a more advanced use of the text-to-speech engine.

Prerequisites

This article assumes a basic knowledge of the Visual Studio IDE, which includes knowledge on how to create, compile and run applications, how to add subdirectories and create and/or add new or existing items to them, and a basic knowledge of the C# language and object oriented programming in general.

TTS on Windows Phone 8

Warning: To use TTS, the ID_CAP_SPEECH_RECOGNITION capability must be set in the app manifest. If this capability is not set, the app might not work correctly or might raise runtime errors.

To add TTS functionality to a Windows Phone 8 application, the first thing that needs to be done is to include a `using` statement for the speech synthesis namespace. This is necessary in any application with TTS functionality:

```
using Windows.Phone.Speech.Synthesis;
```

Then a new object of the SpeechSynthesizer class must be instantiated:

```
SpeechSynthesizer synth = new SpeechSynthesizer();
```

And if only basic TTS functionality is needed, only one additional step is necessary, which is a call to the `SpeakTextAsync` method, to speak the text:

```
await synth.SpeakTextAsync("Testing WP8 TTS");
```

Tip: The `await` operator is used in conjunction with asynchronous methods to make sure applications remain responsive. A detailed explanation of this functionality is out of the scope of this article. (See Asynchronous Programming with Async and Await and Asynchronous Programming For Windows Phone 8)

This basic use of the TTS engine might be enough for some applications, but Windows Phone 8 provides additional mechanisms and methods to add more advanced TTS functionality, by utilizing SSML.

Speech Synthesis Markup Language (SSML)

SSML is an XML-based markup language for speech synthesis applications, and is a recommendation of the W3C's voice browser working group. It allows application developers to control various characteristics of synthesized speech, such as voice, language, pronunciation, etc.

The Microsoft implementation of SSML is based on World Wide Web Consortium Speech Synthesis Markup Language (SSML) Version 1.0.

The `SpeechSynthesizer` class provides 2 methods to speak text that includes SSML markup. The first one, `SpeakSsmlAsync`, receives the text to speak as a parameter, and the second one, `SpeakSsmlFromUriAsync`, speaks text from a standalone SSML document. The first one will be used to show the different speech characteristics that can be controlled by using SSML, and at the end of the article a short explanation of how to speak the contents of a standalone document will be included.

Each SSML document or string requires a speak element. It is the root element of the document, and can be used without any other elements. The `speak` element also specifies the language to be used.

This is the syntax:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="string">
</speak>
```

The three attributes (`version`, `xmlns` and `xml:lang`) are mandatory.

Here is an example:

```
synth = new SpeechSynthesizer();
string ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += " Testing Windows Phone 8 TTS";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

Using SSML this way produces the same results as the basic way to make applications speak described at the beginning of this article, but there are many other elements that can be used to add more advanced features, like the ones explained below.

Inserting sound files

The audio element can be used to insert sound files into the speech. It supports files in PCM, a-law and u-law format, with 8 bits or 16 bits depth, and non-stereo (mono only).

This is the syntax:

```
<audio src="string" /></audio>
```

And here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "Here comes the dog, <audio src=\"ms-appx:///Assets/cats.wav\">Dog
</audio>";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

In this example, the file **cats.wav** that is located under the application's **Assets** subdirectory will be played after speaking the "Here comes the dog" text. If the sound file does not conform to the supported formats or if for any other reason the file cannot be played, the text that is included between the <audio> and </audio> tags will be spoken (in this case, "Dog").


Tip: The location of the sound file could have been defined simply as "**Assets/cats.wav**" instead of "**ms-appx:///Assets/cats.wav**". The difference is the inclusion of the URI scheme. This is the recommended way to do it, since in

some circumstances the application could fail when trying to locate the files if the scheme is not included. The `ms-appx` scheme is used to refer to files that come from the application's package.

Inserting pauses

The break element can be used to insert pauses.

This is the syntax:

```
<break />
<break strength="string" />
<break time="string" />
```

- The `strength` attribute is optional, and can accept any of these values: `none`, `x-weak`, `weak`, `medium`, `strong`, or `x-strong`.
- The `time` attribute is also optional, and it is used to define the break duration in seconds or milliseconds.

Here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "There is a pause <break time=\"500ms\" /> here, and another one <break
strength=\"x-strong\" /> here";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

Defining or changing the pronunciation of words

There are two ways to instruct the speech synthesizer to pronounce a word in a specific way. To define a one-time pronunciation for a word, the phoneme element is a good option, but for words that need to be pronounced in a specific way every time they appear in the same document, or to define the pronunciation of more than one word in a single place, the lexicon element is the right option.

It is important to take into account that when a document includes `phoneme` and `lexicon` elements, the speech synthesizer gives precedence to phonemes.

Phonemes can be defined in-line, as the other elements that have been mentioned in this article.

This is the syntax:

```
<phoneme alphabet="string" ph="string">word</phoneme>
```

The pronunciation of the word "word" is defined in the `ph` attribute, according to the specified `alphabet`.

Here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "<phoneme alphabet=\"x-microsoft-ups\" ph=\"O L AA\">hello</phoneme>, I mean
hello";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

Using a `lexicon` element requires an additional step, which is the creation and/or addition of an external lexicon file.

Lexicon files are XML documents that contain the mapping between the pronunciations and the written representations of words according to a predefined specification.

For this example the lexicon file used is named **lexicon1.xml**, and includes the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  alphabet="x-microsoft-ups" xml:lang="en-US">
  <lexeme>
    <grapheme>wife</grapheme>
    <phoneme> W AI F AI</phoneme>
  </lexeme>
</lexicon>
```

Each word is defined in a `lexeme` element, whose content includes a `grapheme` (the word for which the pronunciation is being defined) and a `phoneme` (the definition of how the word must be pronounced according to the `alphabet` specified in its root `lexicon` element).

In this example, we are instructing the speech synthesizer to pronounce the word "wife" like "WiFi".

Here is an example of how this lexicon file can be used:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "<lexicon uri=\"ms-appx:///Assets/lexicon1.xml\"
type=\"application/pls+xml\"/>";
ssmlText += "She is not my wife";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

It is important to take into account that the lexicon will not be used if the current selected language is different from the one specified in the lexicon, for example if a different language was specified by code using the `SetVoice` method from the `SpeechSynthesizer` class, or a different language was defined in the root speak element of the SSML document.

Note: The Microsoft's official documentation of the `lexicon` element (which is not specific for Windows Phone, (link)) used to research and test some of the topics included in this article includes an example of a lexicon file with a **pls** extension, but using that extension causes an `IO.FileNotFoundException`, whereas files with an **xml** extension work well.

Changing the voice used to speak

There are several ways to change the voice that the speech synthesizer uses to speak text.

The voice can be changed by code, using the `SpeechSynthesizer`'s `SetVoice` method.

Here is an example that uses a LINQ query to look for a female voice that speaks British English.

```
IEnumerable<VoiceInformation> englishVoices = from voice in InstalledVoices.All
                                    where voice.Language == "en-GB"
                                    && voice.Gender.Equals(VoiceGender.Female)
if(englishVoices.Count() > 0)
    synth.SetVoice(englishVoices.ElementAt(0));
await synth.SpeakTextAsync("Testing Windows Phone 8 TTS");
```

The voice can also be changed by using the voice SSML element.

This is the syntax:

```
<voice name="string" gender="string" age="integer" xml:lang="string" variant="integer">
</voice>
```

All of the attributes are optional, but at least one needs to be included. It is important to mention that most of these attributes are considered as "preferred values" by the speech synthesizer, so if an appropriate voice that complies with those values is not available, a voice with different attributes will be used.

Here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "<voice name=\"Microsoft Susan Mobile\" gender=\"female\" age=\"30\"
xml:lang=\"en-US\">";
ssmlText += "This is another test </voice>";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

Additionally, the voice's language can be changed when using the p and s elements, which are used to mark the paragraph and sentence structure of the SSML document when the automatically detected structure is not satisfactory to the developer.

This is the syntax:

```
<p xml:lang="string"> </p>
<s xml:lang="string"> </s>
```

And here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-GB\">";
ssmlText += "<p>";
ssmlText += "<s>First sentence of a paragraph</s>";
ssmlText += "<s xml:lang=\"en-US\">And this is the second sentence</s>";
ssmlText += "</p>";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

Notice that every element that can define a language can define a different one from the one defined in the root `speak` element, because SSML supports the use of different languages in the same document.

Changing the prosody of the speech

Some changes to the prosody of the speech can be made by adding pauses with the `break` element, and defining the document structure with the `p` and `s` elements (all 3 have been explained in previous sections), but the element that provides the more fine-grained control over the prosody of the speech is the prosody element. It can be used to control the pitch and the volume of the voice, among other things.

This is the syntax:

```
<prosody pitch="value" contour="value" range="value" rate="value" duration="value"
volume="value"> </prosody>
```

- The `pitch` attribute can be set as an absolute value expressed in Hertz (for example "500Hz") and a relative value, expressed as the change to the pitch in Hertz or semitones (for example "+10Hz" or "-1st") . It can also be defined as an specific value from an enumeration that includes the following values: `x-low`, `low`, `medium`, `high`, `x-high`, or `default`.
- The `contour` and `range` attributes are supposed to be used to define different pitches at different parts of the speech, and to define a pitch range, respectively, but at the time of writing this article, they don't produce any effect on the speech produced in Windows Phone 8.
- The `rate` attribute defines the speed at which the speech must be executed, expressed as a multiplier of the default (for example "2" for double speed) or a specific value from an enumeration that includes this values: `x-slow`, `slow`, `medium`, `fast`, `x-`

`fast`, or `default` .

- The `duration` attribute is supposed to be used to alter the speech speed as well, but at the time of writing this article, it does not produce any effect on the speech produced in Windows Phone 8.
- The `volume` attribute sets the volume and can take absolute values from (0.0 to 100.0), relative values (expressed as a volume change, for example "+5) or specific values from an enumeration that includes the following values: `x-soft`, `soft`, `medium`, `loud`, `x-loud`, or `default`.

Here is an example showing some of the attributes:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "Testing the ";
ssmlText += "<prosody pitch=\"+100Hz\" volume=\"70.0\" >Prosody</prosody>";
ssmlText += " element";
ssmlText += "Normal,<prosody rate=\"2\"> Very Fast,</prosody>";
ssmlText += "<prosody rate=\"0.4\"> now slow,</prosody>";
ssmlText += "and normal again";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

All of the attributes of the prosody element are considered as "suggestions" by the speech synthesizer, and because of that some of them can be ignored or substituted if the speech processor does not support them.

Monitoring speech progress

If an application needs to take specific actions depending on the progress of a speech, it can use the mark element. It is an empty element used to mark a specific position in a speech. When the speech synthesizer is playing the speech and finds a mark, it raises a `SpeechBookmarkReached` event that includes information to identify the `mark` element that triggered it, which the developer can use to perform any actions required.

This is the syntax:

```
<mark name="string" />
```

And here is an example of how it can be used:

```
public MainPage()
{
    InitializeComponent();
    synth = new SpeechSynthesizer();
    // Add the event handler for the speech progress events
    synth.BookmarkReached += new TypedEventHandler<SpeechSynthesizer,
SpeechBookmarkReachedEventArgs>(synth_BookmarkReached);
}

private async void Button7_Click(object sender, RoutedEventArgs e)
{
    ssmlText = "<speak version=\"1.0\" ";
    ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
    ssmlText += "<mark name=\"START\"/>";
    ssmlText += "This is the first half of the speech.";
    ssmlText += "<mark name=\"HALF\"/>";
    ssmlText += "and this the second half. Ending now";
    ssmlText += "<mark name=\"END\"/>";
    ssmlText += "</speak>";
```

```
    await synth.SpeakSsmlAsync(ssmlText);
}

static void synth_BookmarkReached(object sender, SpeechBookmarkReachedEventArgs e)
{
    Debugger.Log(1, "Info", e.Bookmark + " mark reached\n");
}
```

- The `Windows.Foundation` and `System.Diagnostics` namespaces need to be used for this code to compile.

To appreciate the results of this example, press F5 to debug your application and keep an eye on the output window as you click the button to start the speech.

Notice that two additional steps were required to take advantage of this progress monitoring capability: First, a new typed event handler was added to the synthesizer's `BookmarkReached` event, and then the handler method itself had to be implemented.

Specifying content type and aliasing parts of a speech

The say-as element can be used to specify the content type of the text contained in the element, so the speech synthesizer knows how to pronounce it correctly.

This is the syntax:

```
<say-as interpret-as="string" format="digit string" detail="string"> <say-as>
```

- The `interpret-as` attribute is required. It defines the type of content, and can take any of the following strings: `date`, `cardinal`, `ordinal`,`characters`, `time` and `telephone`.
- The `date`, `time` and `telephone` attributes are supposed to be used to specify the appropriate content type and a specific format, but at the time of writing this article, these 3 attributes are just ignored by the speech synthesizer in Windows Phone 8.
- The `cardinal` and `ordinal` attributes instruct the synthesizer on the way it should say numbers.
- The `characters` attribute instructs it to read the contained text as individual characters.

Here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "<p>This is an ordinal number: <say-as interpret-as=\"ordinal\">121</say-
as></p>";
ssmlText += "<p>This is a cardinal number: <say-as interpret-as=\"cardinal\">121</say-
as></p>";
ssmlText += "<p>And these are just individual numbers: <say-as interpret-
as=\"characters\">121</say-as></p>";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

The sub element is used to define an alias for a string. It is probably not particularly useful as it works just as a one-time alias, so its intention is probably to give more clarity to the SSML document by providing a way to have a written and spoken form of the related text.

This is the syntax:

```
<sub alias="string"> </sub>
```

And here is an example:

```
ssmlText = "<speak version=\"1.0\" ";
ssmlText += "xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">";
ssmlText += "This code runs on <sub alias=\"Windows Phone 8\">WP8</sub>";
ssmlText += "</speak>";
await synth.SpeakSsmlAsync(ssmlText);
```

In this example the words "Windows Phone 8" are read in place of WP8.

Playing an external SSML document

External SSML documents are XML files that follow the structure and syntax of the SSML language that has been shown in all of the previous examples.

To play SSML files the `SpeechSynthesizer` class provides the `SpeakSsmlFromUriAsync` method, which is very similar to `SpeakSsmlAsync` and `SpeakTextAsync` but it receives an Uri object as a parameter.

For this example, we will use a file named **SSML1.xml**, which contains the following text:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice gender="male" xml:lang="en-US">
    <prosody rate="0.8">
      <p>Thanks for reading the article, and thanks for trying the examples</p>
      <p>Now be creative, and create amazing applications for this fantastic
platform</p>
      <voice gender="male" xml:lang="es">Adios</voice>
    </prosody>
  </voice>
</speak>
```

After the file has been created (or added to the project), the final step is to call the `SpeakSsmlFromUriAsync` method to play it:

```
await synth.SpeakSsmlFromUriAsync(new Uri("ms-appx:///Assets/SSML1.xml"));
```

Note: The Microsoft's official documentation on TTS for Windows Phone (link) used to research and test some of the topics included in this article includes an example of an external SSML file with an **ssml** extension, but using that extension causes an `IO.FileNotFoundException`, whereas files with **xml** extension work well.

Source Code

The project with the code from all the examples included in the article can be downloaded here: Media:AdvancedTTS_test.zip

Reference material and additional information

More info on Speech Synthesis Markup Language (SSML)

- Speech Synthesis Markup Language Reference
- Speech Synthesis Markup Language (SSML) Version 1.0 (W3C Recommendation)

More info on lexicon files

- Pronunciation Lexicon Specification (PLS) Version 1.0

More info on alphabets

- Phonetic Alphabet Reference

More info on text-to-speech for Windows Phone

- Text-to-speech (TTS) for Windows Phone

Version Hint

Windows Phone: [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

Nokia Asha: [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

Series 40: [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]
[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

Symbian: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]] [[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]