Computer Speech

Accidentally I discovered a new API that makes it easier to interact with your site/ app using your voice. The Speech Synthesis API provide both ends of the computer conversation, the recognition to listen and the synthesis to speak.

Right now I'm more interested in the synthesis part and how we can include it as additional feedback on our sites and applications as an additional cue for user interaction.

Synthesis

The Speech Syntthesis API gives us a way to "speak" strings of text without having to record them. These 'utterances' (in API speak) can be further customized.

At the most basic the utterance is made of the following:

- An instance of `SpeechSynthesisUtterance`
- The text and language we want the voice spoken in
- The instruction to actually speak the command using `speechSynthesis.speak`

```
var msg1 = new SpeechSynthesisUtterance();
msg1.text = "I'm sorry, Dave, I can't do that";
msg1.lang = 'en-US';

speechSynthesis.speak(msg1);
```

The example below changes the content and the language to `es-cl` (Spanish as spoken in Chile). The structure of the code is the same.

```
var msg2 = new SpeechSynthesisUtterance();
msg2.text = "Dicen que el tiempo guarda en las bastillas";
msg2.lang = 'es-cl';

speechSynthesis.speak(msg2);
```

Copy and past each example in your Dev Tools (I use Chrome's and have tested in Chrome and Firefox) and notice how different the default voices are for each message and for each browser you test with.

We can further customize the utterance with additional parameters. The parameters are now

- `msg` contains a new instance of `SpeechSynthesisUtterance`
- `voices` contains an array of all the voices available to the user agent (browser in this case)
- `voice` assigns a voice from the voices array to the instance of utterance we are working with
- `voiceURI` specifies speech synthesis voice and the location of the speech synthesis service that the web application wishes to use
- `rate` indicates how fast the text is spoken. 1 is the default rate supported by the speech synthesis engine or specific voice (which should correspond to a normal speaking rate). 2 is twice as fast, and 0.5 is half as fast
- `pitch` specifies the speaking pitch for the utterance. It ranges between 0 and 2 inclusive, with 0 being the lowest pitch and 2 the highest pitch. 1 corresponds to the default pitch of the speech synthesis engine or specific voice

As before `text` holds the message we want the browser to speak, `lang` holds the language we want the browser to speak in and the `speechSynthesis.speak` command will actually make the browser speak our phrase.

```
var msg = new SpeechSynthesisUtterance();
var voices = window.speechSynthesis.getVoices();
// Note: some voices don't support altering params
msg.voice = voices[0];
msg.voiceURI = 'native';
msg.volume = 1; 'native'// 0 to 1
msg.rate = 1; // 0.1 to 10
msg.pitch = 2; //0 to 2t = 'Hello World';
msg.lang = 'en-US';

speechSynthesis.speak(msg);
'Hello World'
```

Putting speech synthesis into action: why and where would we use this?

The most obvious place for me to use speech synthesis is as additional cues and messages to end-users when there is an error and problem. We'll define three different functions to encapsulate the errors messages we want to "talk" to the user about.

For this example we'll use the following HTML code:

```html
<form id="form">
  <fieldset>
    <legend>Basic User Information</legend>
    <label for="username">Username</label>
    <input id="username" type="text" placeholder="User Name">
    <label for="password">Password</label>
    <input id="password" type="password" placeholder="password">
  </fieldset>
</form>
```

For the sake of the demo I'm only interested in the input fields and not in having a fully functional working form.

I will break the Javascript portion of the demo in two parts. The first part defines the Speech Recognition portion of the script which is very similar to the examples we've already discussed.

```javascript
// Setup the Username Empty Error function
function speakUsernameEmptyError() {
  let msg1 = new SpeechSynthesisUtterance();

  msg1.text = "The Username field can not be empty";
  m"The Username field can not be empty"s.speak(msg1);
}

// Setup the Password Empty Error function
```

```
function speakPasswordEmptyError() {
  let msg2 = new SpeechSynthesisUtterance();
  msg2.text = "The Password field can not be empty";
  msg2.lang = 'en-US';

  speechSynthesis.speak(msg2);
}
"The Password field can not be empty"
```

The second part of the script assigns `blur` event listeners to the input elements. Inside each event handler the code checks if the field is empty. If it is the code adds a 1px red border around it and plays the appropriate utterance we crafted earlier. If the field is not empty, either because the user entered a value before moving out of the field or at a later time, we set the border to a 1-pixel solid black color.

```
// Assign variables to hold the elements
let username = document.getElementById('username');
let password = document.get'username'd('password');

'password'// Add blur event listener to the username fieldaddEventListene
  // If the 'blur'// If the field is empty
  if (username.value.length <= 0) {
    // Put a 1 pixel red border on the input fieldyle.border = '1px solid
    // Speak the err'1px solid red'// Speak the error as specified in the
    // speakUsernameEmptyError function
  } else {
    username.style.border = '1px solid black';
  }
});

// Add blur event listener to t'1px solid black'// Add blur event listene
password.addEventListener('blur', function() {
  // If the field is empty
  if (password.value.length <= 0) {
    // Put a 1 pixel red border on the input field
    password.style.border = '1px solid red';
    // Speak the error as specified in the
    // speakPasswordEmptyError functionassword.style.border = '1px solid b
  }
})
'1px solid black'
```

The functions and event listeners are very basic and could stand some additional work, particularly in the validation area.

Sources

- Web apps that talk - Introduction to the Speech Synthesis API
- Talking Web Pages and the Speech Synthesis API

Recognition

Speech Recognition only works in Chrome and Opera. Firefox says it supports it but it doesn't work and returns a very criptic error message.

The second part of the Speech Synthesis API is recognition. Where in the prior post we used the Speech Synthesis API to have the browser talk to use when there was a problem on the form; in this post we'll use a demo from Mozilla to illustrate a potential use of speech recognition to make the browser change the background color of the page based on what the user speaks in to a microphone.

Because we're using the user's microphone the page/application must explicitly ask for permission and it must be granted before any of this code will work. This permission can be revoked at any time.

The HTML is simple. A place holder paragraph to hold any hints we provide the user and another paragraph to hold the output of the processes.

```
<h1>Speech color changer</h1>
```

```
<p class="hints"></p>
<div>
    <p class="output"><em>...diagnostic messages</em></p>
</div>
```

The first portion of the scrpt creates a JSGF Grammar for the elements we want to recognize. JSGF is an old Sun Microsystems W3C submission that was used as the basis for the W3C Voice Browser Working Group (closed on October, 2015).

This will create the vocabulary that the rest of the script will recognize.

```
var colors = [ 'aqua' , 'azure' , 'beige', 'b'azure' 'black', 'blue', 'bro
'chocolate', 'coral', 'cr'black' 'cyan', 'fuchsia', 'ghostwhite', 'gold',
'goldenrod', 'gray', 'green', 'indigo', 'ivory', 'khaki', 'lavender', 'cy
'linen', 'magenta', 'maroon', 'moccasin', 'navy', 'olive', 'orange', 'orch
'peru', 'pink', 'plum', 'purple', 'red', 'salmon', 'sienna', 'silver', 'sn
'tan', 'teal', 'thistle', 'tomato', 'turquoise', 'violet', 'white', 'yell
'#JSGF V1.0; grammar colors; public <color> = '// You can optionally log t
console.log('grammar');
```

We next setup the speech recognition engine. The three variables: `SpeechRecognition`, `SpeechGrammarList` and `SpeechRecognitionEvent` have two possible values, either an unprefixed version (not supported anywhere) or the webkit prefixed version (supported by Chrome and Opera). It's always a good idea to future proof your code; I suspect that when Firefox finally supports the API it will be unprefixed.

```
var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition;
var SpeechGrammarList = SpeechGrammarList || webkitSpeechGrammarList;
var SpeechRecognitionEvent = SpeechRecognitionEvent || webkitSpeechRecogni
```

The script then does assignments. First it associates variables with the speech recognition engine we set up above. Next the script configures the engine with the grammar created earlier and the attributes for the recognition engine to work. It also create placeholders for the HTML elements that will hold messages to the user and will actually change the background color.

```
var recognition = new SpeechRecognition();
var speechRecognitionList = new SpeechGrammarList();
speechRecognitionList.addFromString(grammar, 1);
recognition.grammars = speechRecognitionList;
//recognition.continuous = false;
recognition.lang = 'en-US';
recognition.interimResults = false;
recognition.maxAlternatives = 1;

var diagnostic = document.querySelector('.output');
var bg = document.querySelector('html');
var hints = document.querySelector('.hints');
'en-US'
```

For each color that we make available to the user, we use the color as the `background-color` to give the user an additional cue regarding the colors he can choose from.

```
var colorHTML= '';
colors.forEach(function(v, i){
    console.log(v, i);
    colorHTML += '<span style="background-color:' + v + ';"> ' + v + ' </s
});
```

The script is almost ready to start. It adds a message to the `.hint` container and, when the user clicks anywhere on the page, it begins the recognition process.

```
hints.innerHTML = 'Tap/click then say a color to change the background co

document.body.onclick = function() {
    recognition.start();
    console.log('Ready to receive a color command.');
};
```

When the user speaks and the script gets a result the `SpeechRecognitionResultList` object contains `SpeechRecognitionResult` objects. It has a getter so it can be accessed like an array.

The `last` variable holds the `SpeechRecognitionResult` object at the last position.

Each `SpeechRecognitionResult` object contains `SpeechRecognitionAlternative` objects that contain individual results. The `last` element represents the latest result the script obtained and, using the array notation we get the `transcript` for first result ([0]) of the latest (`last`) response the client has stored.

The script will allso display the result it received (sometimes it’s funny to see what the recognition engine thinks you said), change the background color to the specified color and provide a confidence level percentage, the higher the number the most likely the user will get a correct answer and the color will change.

```
recognition.onresult = function(event) {
    var last = event.results.length - 1;
    var color = event.results[last][0].transcript;

    diagnostic.textContent = 'Result received: ' + color + '.';
    bg.style.backgroundColor = color;
    console.log('Confidence: ' + event.results[0][0].confidence);
};
```

The final part of the script handles additional events. When the script detects the end of speech it stops the recognition. When there is no match it will notify the users in the diagnostic element. Finally, if there is an error, we also report it to the user.

```
recognition.onspeechend = function() {
    recognition.stop();
};

recognition.onnomatch = function(event) {
    diagnostic.textContent = "I didn't recognise that color.";
};

recognition.onerror = function(event) {
    diagnostic.textContent = 'Error occurred in recognition: ' + event.er
}
```

Full recognition example

Speech Recognition only works in Chrome and Opera. Firefox says it supports it but it doesn't work and returns a very criptic error message.

There is another way to work with speech recognition: dictation. Rather than work through the example that illustrates how we can use the recognition portion of the API to create a dictation application that you can then copy of email.

Code demo

All the code for this posts is available on Github

- https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API