**Title Page**

Hi and welcome to the Zero to Cognitive Series. This tutorial is designed to introduce you to building cognitive applications using the IBM Bluemix Cloud. We will use the IBM Cognitive Solution Advisor as our foundation for this tutorial. This is Chapter 4 where we start getting Watson to talk back to you.

**Page 2**

This will be a very simple session. We’re going to get Watson to talk to you and we’re going to leave the timing of Watson’s speaking up to you. The basic steps we’ll go through will be familiar because of the previous lesson.

1. Step 1 will be to create a Watson Text to Speech service, if you did not already do that in the previous lesson.
2. Step 2 will be to copy the Text to Speech credentials and update the manifest file
3. Step 3 will be to tell Watson what to say

This should take about 15 minutes.

**Page 3**

To add a service into your app, we’ll do the same thing we did to enable Speech to Text.

1. Step 1, log in to your bluemix dashboard and go to your app
2. Step 2, select “Add a Service or API”
3. Step 3, select Watson Text to Speech
4. Step 4, note the specific name of your service (Text to Speech-xx)
5. Step 5, display and copy your credentials

Let’s get started

**Page 4**

Open your env.json file and copy in the new credentials for Text to Speech

Open your manifest.yml file and add a second service “Text to Speech-xx” where the dash xx is what you wrote down in Step 4 from the previous page

**Page 5**

Open your speech\_to\_text.js controller file. Lines 23-26 have a new, empty routine to support text to speech. Conceptually, we’re going to do the same kinds of things in this routine as we did for Speech to Text. We need to authenticate with Watson, we need to provide the text to be spoken, and we need to provide you with control over the resulting audio stream. Let’s get to it.

1. Step 1, go to the answers folder and open the speech-to-text-complete.js file
2. Step 2, Let’s look at what’s in the new routine.
   1. Line 26, the variable ttsConfig retrieves your credentials and passes them to Watson
   2. Line 27 creates a variable (transcript) from the text passed in to the routine
   3. Lines 28-32 are not used in this version of the code and allow for a document to be downloaded for transcription
   4. Line 33 performs error handling
   5. Line 34 is the one you care about, which sends (pipes) the audio back to your computer.
3. Copy lines 24 to 35 inclusive from this file and paste them into your speech-to-text.js file replacing lines 23-26 inclusive.

**Page 6**

In this simple example, we’re going to create a space on the page where you can put in the text you want Watson to speak. We’ll add a button to tell Watson when to talk and then we’ll add an audio control to allow you to start and stop the speech.

1. Lines 48-51 create a text area, called “chat” to hold the text you’ll send to Watson
2. Lines 52-54 create a button, called “readText” with a label of “Watson, Read this.”
3. Lines 55-61 create an audio control call “a\_player”

None of these elements are ‘active’ yet. We need to add a little javascript code to make all this work.

**Page 7**

Open your z2c-speech.js file. Lines 42-46 create an empty function for the “Watson, Read this.” Button and lines 48-51 create an empty function called “onCanplaythrough”. Let’s look at the complete source code to understand what we need to make happen and how we talk to the text to speech service we created a few minutes ago.

Open the z2c-speech-complete.js file in your answers folder

1. Step 1 note on line 8 that a new variable, ‘readText’ has been created and refers to the html id of the same name. The fact that both the javascript variable name and the html id name are the same is for convenience and is not a requirement of either language. The requirement is that the characters inside the double quote marks be the same as the html id name.
2. Step 2, lines 71-82, define a function which activates the audio player on the html page.
   1. Line 73 creates a variable linked to the audio player id in the html file
   2. Line 74 removes this function as an event listener, to prevent it being added more than one time.
   3. Lines 75-77 set the audio play function to the beginning of the transcript
   4. Lines 78-79 cause the audio controls to be displayed and turns mute off if it was on
   5. Line 80
   6. Line 81 sets the cursor to the default (pointer/hand) in case it was still a busy cursor.
3. Step 3, lines 43-68, create the “on click” function for the Watson button.
   1. Line 46: If Watson speech to text is active, stop the service
   2. Lines 47-50: update the visual aspect of the microphone and stop buttons
   3. Line 53: get the text from the “chat” object on the html page
   4. Line 54: specify the ‘voice’ to use for speaking
   5. Line 55: specify where the audio stream is to be sent
   6. Lines 56-59: create a variable to pass to the audio player which links the text to be spoken with the Watson service
   7. Line 60: open a link between the audio player and Watson
   8. Line 61: stop the player
   9. Line 62 add the event listener
   10. Line 63-64: mute the listener, then invoke play, which invokes the event listener
   11. Lines 65-66: set the cursors to busy
4. Step 4 copy both functions and the variable from the ‘complete’ file to z2c-speech.js

**Page 8**

1. Step 1: If you’ve not already done so, Save all of your files.
2. Step 2: Open a command prompt and navigate to Chapter04
   1. Execute the command npm install
   2. Execute the command node index.js and note the port number displayed by the newly started application. On my system, you’ll see it displayed as port 6008
   3. Open your browser and direct it to the correct URL, for example: <http://localhost:6008>
   4. Test the application, talk to it, give it some text to use, tell it to talk to you.
3. Step 3: send your information to github
   1. Git status (tells you what files you’ve changed)
   2. Git add …. For each changed file
   3. Git commit –m ‘your commit message goes here’
   4. Git push … sends your data back to your git repository
4. Step 4: send your code to cloud foundry
   1. Cf login (uses your ibm id and password)
   2. Cf push Z2C (this will replace your existing Z2C app with the one from Chapter04)
   3. Log in to Bluemix and navigate to your Z2C application
   4. Click on the highlighted URL, this will open a new browser window or tab. Test the application.
5. Step 5: Woohoo!! You’ve successfully built your second Bluemix application.

**Page 9**

You have completed the first four chapters in this tutorial. In our next session, we will work with classifiers. This is how Watson figures out what you’re saying.

Thanks for your time and see you in the next session.

**Page 10**