Chapter 6: Creating a Custom Dialog

Learning Bluemix & Cognitive

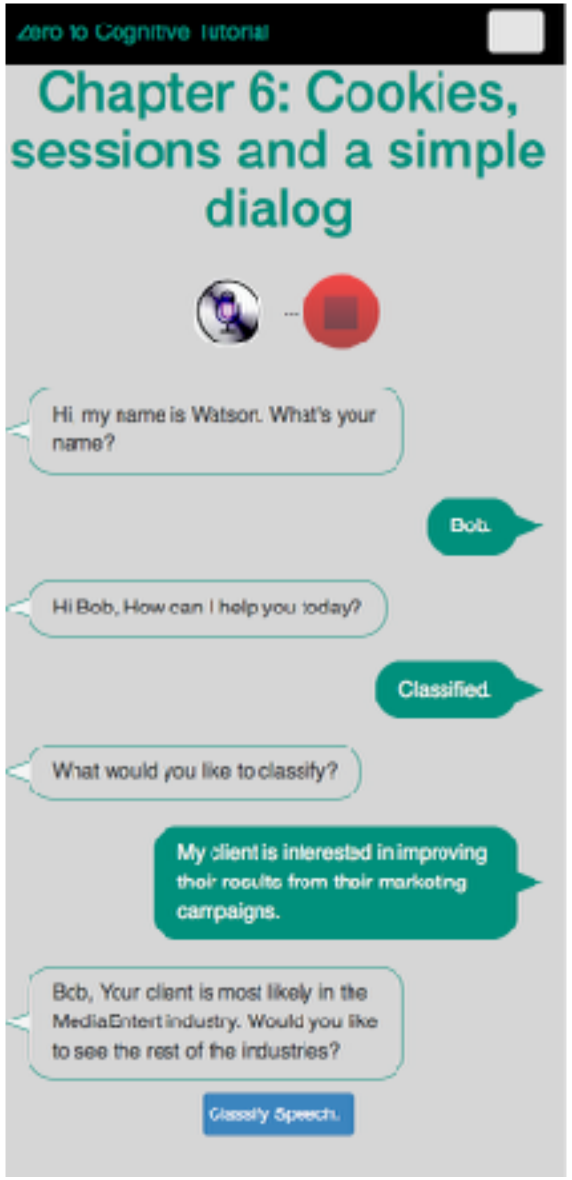
Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales

Git Repository: https://github.com/rddill-IBM/ZeroToCognitive

Chapter 6: Creating a Custom Dialog

- Identifying who we are: cookies
- Why … restful services, no authentication yet
  - How would authentication help?
- Creating and Using cookies
- The flow:
  - talk to Watson, send text for analysis
  - Identify most likely classifier and tell user.

What are we going to build today?



Zero to Cognitive Tutorial

Chapter 6: Cookies, sessions and a simple dialog

Hi my name is Watson. What's your name?

Bob.

Hi Bob, How can I help you today?

Classified.

What would you like to classify?

My client is interested in improving their results from their marketing campaigns.

Bob, Your client is most likely in the MediaEntert industry. Would you like to see the rest of the industries?

Classify Speech.

Watson greets you and asks for your name

You respond with your first name/salutation

Store Name in Cookie
Store state in Cookie

Watson responds with "Hi, {name}, …

Name extracted from cookie, passed in text to server

Store state in Cookie

You respond with 'classify'

Store state in Cookie

Watson responds with "what would you like …

Store state in Cookie

You respond with text to classify

Store state in Cookie

Watson responds with "Bob, your client is …

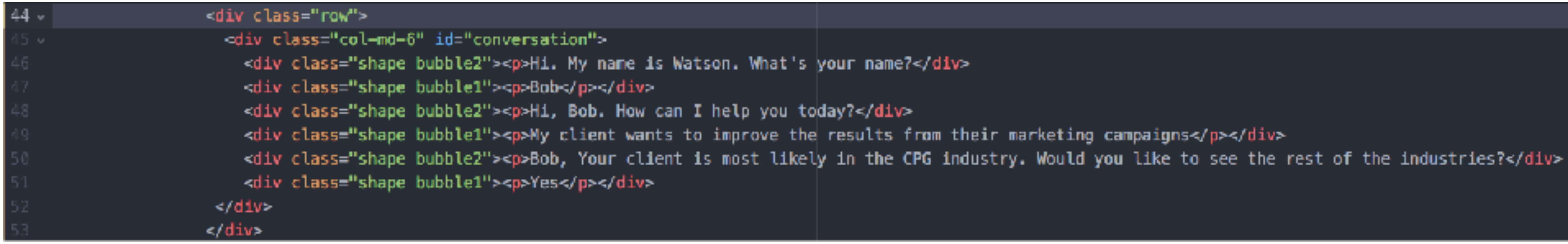
Name extracted from cookie, passed in text to server

You answer "Yes" and click stop or Classify

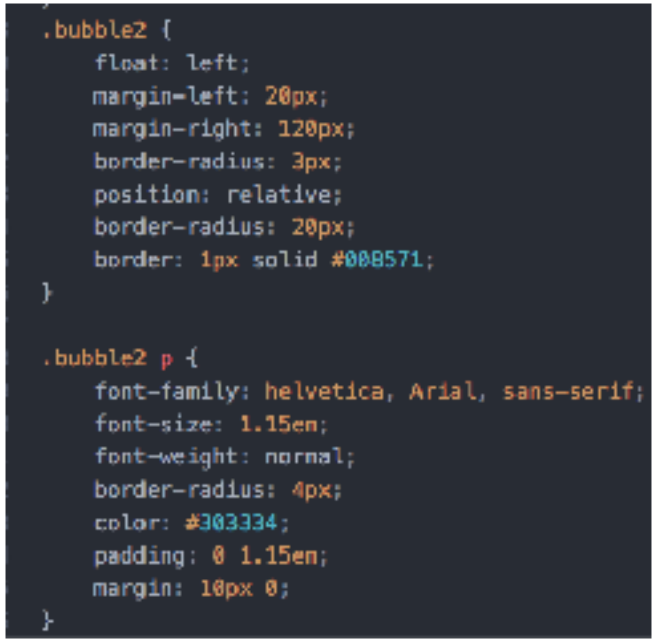
Watson responds with the popup window from lesson 5

Pop-up processed locally

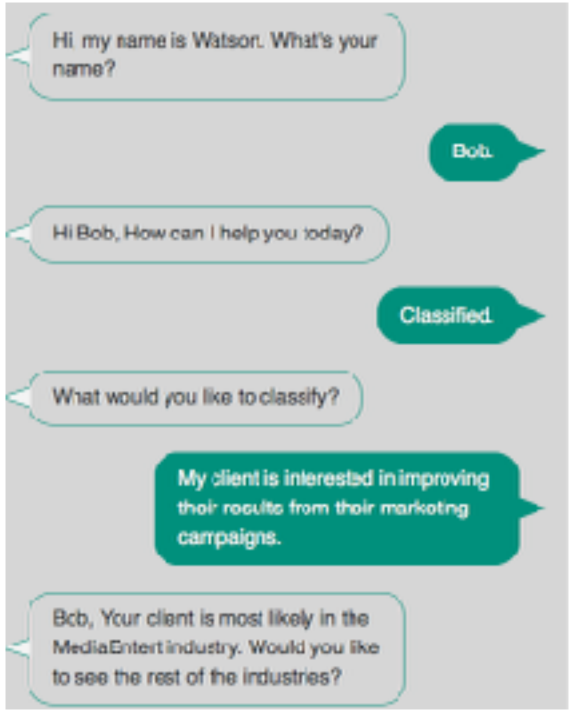
HTML and CSS



```html
<div class="row">
  <div class="col-md-6" id="conversation">
    <div class="shape bubble2"><p>Hi. My name is Watson. What's your name?</div>
    <div class="shape bubble1"><p>Bob</p></div>
    <div class="shape bubble2"><p>Hi, Bob. How can I help you today?</div>
    <div class="shape bubble1"><p>My client wants to improve the results from their marketing campaigns</p></div>
    <div class="shape bubble2"><p>Bob, Your client is most likely in the CPG industry. Would you like to see the rest of the industries?</div>
    <div class="shape bubble1"><p>Yes</p></div>
 </div>
</div>
```



```css
.bubble2 {
    float: left;
    margin-left: 20px;
    margin-right: 120px;
    border-radius: 3px;
    position: relative;
    border-radius: 20px;
    border: 1px solid #008571;
}

.bubble2 p {
    font-family: helvetica, Arial, sans-serif;
    font-size: 1.15em;
    font-weight: normal;
    border-radius: 4px;
    color: #303334;
    padding: 0 1.15em;
    margin: 10px 0;
}
```



Hi my name is Watson. What's your name?

Bob.

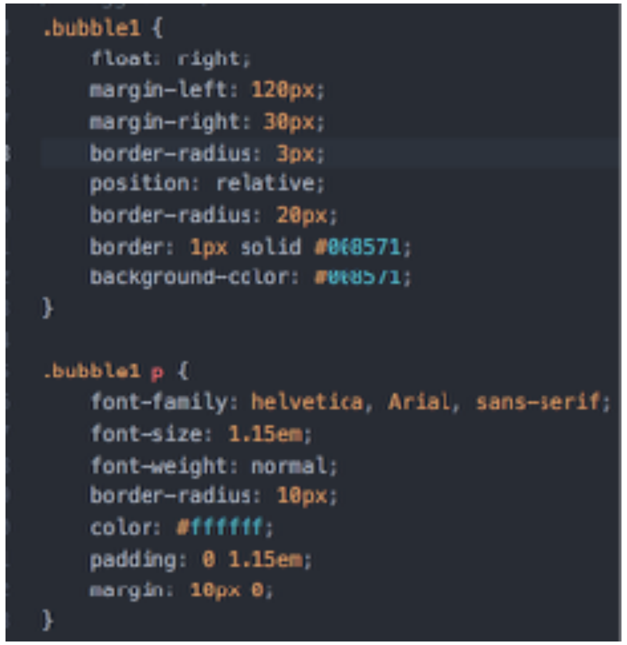
Hi Bob, How can I help you today?

Classified.

What would you like to classify?

My client is interested in improving their results from their marketing campaigns.

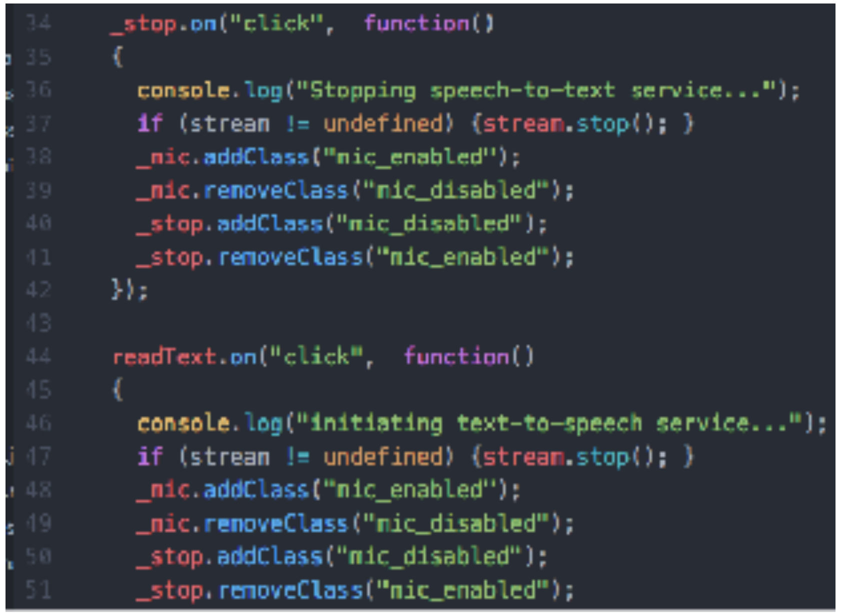
Bob, Your client is most likely in the MediaEntert industry. Would you like to see the rest of the industries?



```css
.bubble1 {
    float: right;
    margin-left: 120px;
    margin-right: 30px;
    border-radius: 3px;
    position: relative;
    border-radius: 20px;
    border: 1px solid #008571;
    background-color: #008571;
}

.bubble1 p {
    font-family: helvetica, Arial, sans-serif;
    font-size: 1.15em;
    font-weight: normal;
    border-radius: 10px;
    color: #ffffff;
    padding: 0 1.15em;
    margin: 10px 0;
}
```

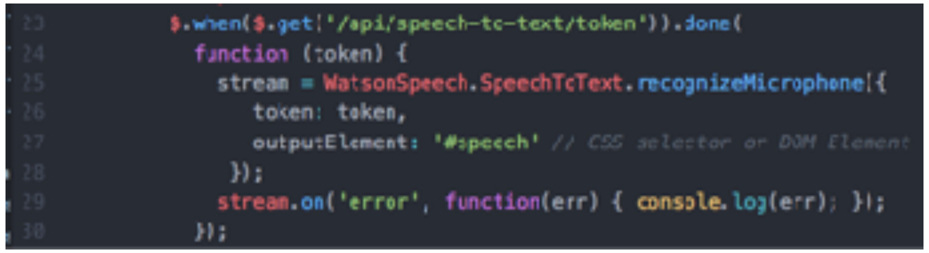
Refactoring

- The objective in refactoring any piece of code is to identify those portions which will be duplicated if they are not rewritten. In our case, both the code in the z2c-speech and z2cNLC files will benefit from refactoring.
  - The code to reset the classes for the microphone and stop button are repeated several times in the code. This creates unnecessary code and creates opportunities to insert bugs in the code.



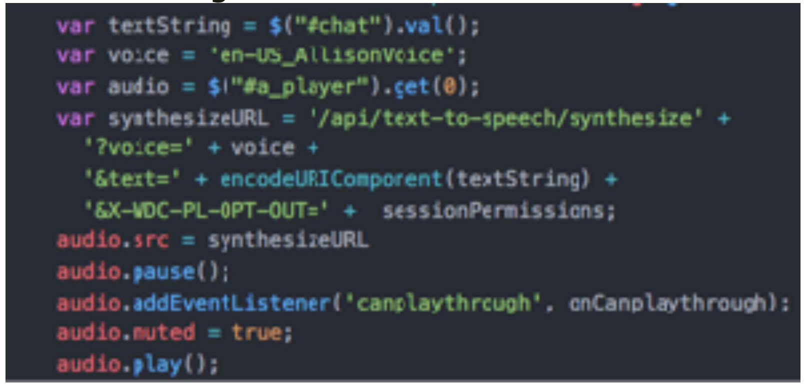
```
    _stop.on("click",  function()
    {
      console.log("Stopping speech-to-text service...");
      if (stream != undefined) {stream.stop(); }
      _mic.addClass("mic_enabled");
      _mic.removeClass("mic_disabled");
      _stop.addClass("mic_disabled");
      _stop.removeClass("mic_enabled");
    });

    readText.on("click",  function()
    {
      console.log("initiating text-to-speech service...");
      if (stream != undefined) {stream.stop(); }
      _mic.addClass("mic_enabled");
      _mic.removeClass("mic_disabled");
      _stop.addClass("mic_disabled");
      _stop.removeClass("mic_enabled");
```

  - We want to be able to use the speech to text service and dynamically tell it what html element to link to, so it makes sense to refactor this code.



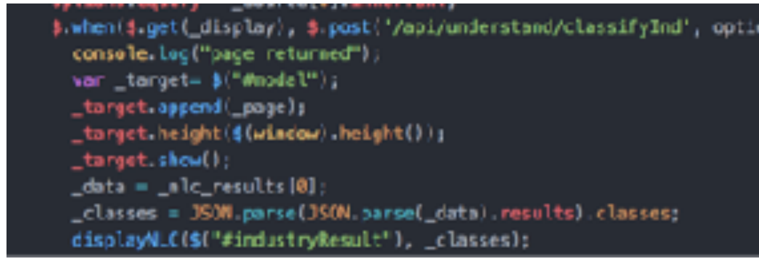
```
        $.when($.get('/api/speech-to-text/token')).done(
          function (token) {
            stream = WatsonSpeech.SpeechToText.recognizeMicrophone({
              token: token,
              outputElement: '#speech' // CSS selector or DOM Element
            });
            stream.on('error', function(err) { console.log(err); });
          });
```

  - We want to be able to use the text to speech service repeatedly and hide the audio control. Again, refactoring here makes sense.



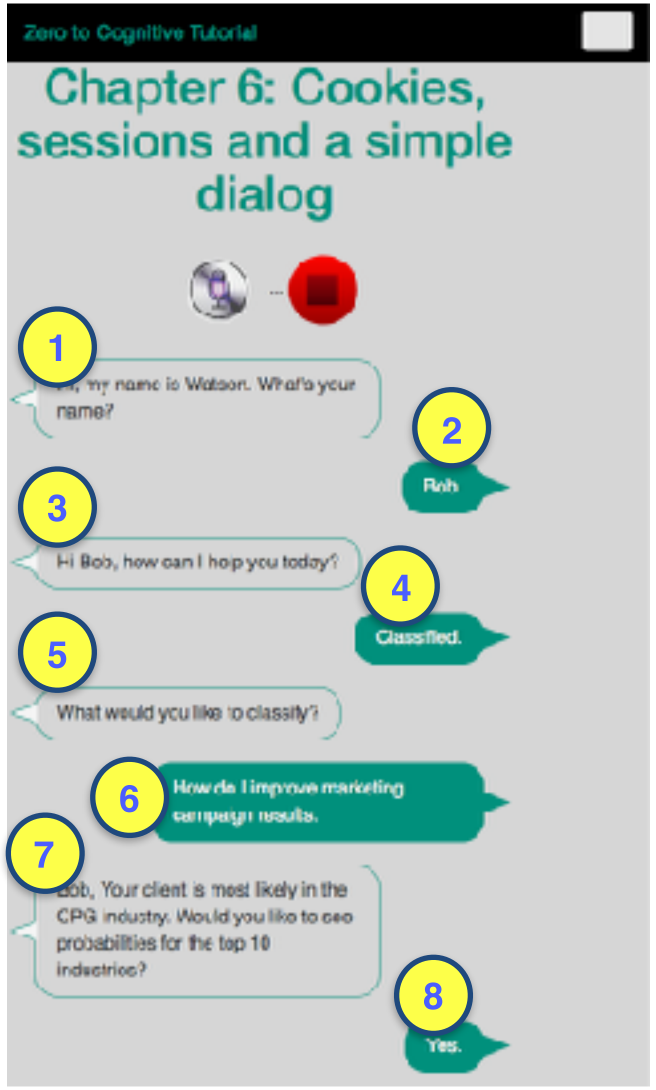
```
var textString = $("#chat").val();
var voice = 'en-US_AllisonVoice';
var audio = $("#a_player").get(0);
var synthesizeURL = '/api/text-to-speech/synthesize' +
  '?voice=' + voice +
  '&text=' + encodeURIComponent(textString) +
  '&X-WDC-PL-OPT-OUT=' +  sessionPermissions;
audio.src = synthesizeURL
audio.pause();
audio.addEventListener('canplaythrough', onCanplaythrough);
audio.muted = true;
audio.play();
```

  - In the Z2C NLC file, the function which posts the classification request to the server also automatically calls the display function which pops up the modal dialog. This is not what we want to have happen, so we need to do a little refactoring here, too.



```
$.when($.get(_display), $.post('/api/understand/classifyInd', optio
  console.log("page returned");
  var _target= $("#modal");
  _target.append(_page);
  _target.height($(window).height());
  _target.show();
  _data = _nlc_results[0];
  _classes = JSON.parse(JSON.parse(_data).results).classes;
  displayNLC($("#industryResult"), _classes);
```

The Custom Dialog



Zero to Cognitive Tutorial

Chapter 6: Cookies, sessions and a simple dialog

1 Hi, my name is Watson. What's your name?

2 Bob

3 Hi Bob, how can I help you today?

4 Classified.

5 What would you like to classify?

6 How do I improve marketing campaign results.

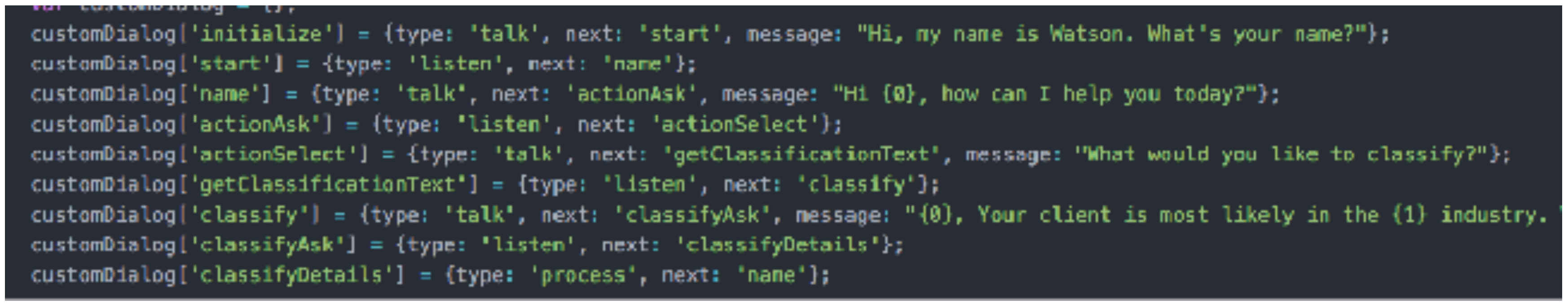
7 Bob, Your client is most likely in the CPG industry. Would you like to see probabilities for the top 10 industries?

8 Yes.

There are nine steps in this custom dialog:

1. Initialization - greeting if Watson does not know your name.
2. providing your name
3. Watson responding, asking how to help
4. You identifying what you want to do
5. Watson responding to your request for classification
6. You providing text to classify and pressing the classify button
7. Watson identifies the highest probability industry and asks if you want to see the rest of the industries
8. You answering Yes
9. Watson displaying the detail table from Chapter 5

We will manage them through a javascript object, shown below, which shows the name of each step, the kind of action to be taken, and, if Watson is to say something, what that phrase is. We could also separate the text into a separate object, which would simplify NL support.



```
customDialog['initialize'] = {type: 'talk', next: 'start', message: "Hi, my name is Watson. What's your name?"};
customDialog['start'] = {type: 'listen', next: 'name'};
customDialog['name'] = {type: 'talk', next: 'actionAsk', message: "Hi {0}, how can I help you today?"};
customDialog['actionAsk'] = {type: 'listen', next: 'actionSelect'};
customDialog['actionSelect'] = {type: 'talk', next: 'getClassificationText', message: "What would you like to classify?"};
customDialog['getClassificationText'] = {type: 'listen', next: 'classify'};
customDialog['classify'] = {type: 'talk', next: 'classifyAsk', message: "{0}, Your client is most likely in the {1} industry.
customDialog['classifyAsk'] = {type: 'listen', next: 'classifyDetails'};
customDialog['classifyDetails'] = {type: 'process', next: 'name'};
```

Cookies and Custom Dialog functions

- The application creates and uses 3 cookies: **name**, **step**, **prev**(ious)**step**
  - **name** allows Watson to sound more personal
  - **step** and **prevstep** enable the app to understand what to do next.
- startDialog()
  - initialize the dialog, figure out if we know your name, yet.
- talkToMe()
  - used to provide a message to the speech to text function, handles details of speech to text. Displays what Watson is saying.
- listenToMe()
  - used to translate my voice into text. dynamically changes to html object where text is to be displayed.
- nextStep()
  - logic for executing the next logical step. Uses all three cookies.
- incrementStep()
  - cookie management to keep track of where I'm going and what I last did. Updates step and prevstep.

The Plan: 30 minute Chapters with an hour or two of practice

1. The Story, Architecture for this app	
2. Setting up Bluemix	
3. Building your first Watson App	(Watson Speech to Text)
4. Getting Watson to talk back	(Watson Text to Speech)
5. Understanding Classifiers	(Watson NLC)
6. Creating a custom dialog with Watson	(custom Q&A, session management)
7. Authentication	(puts C2 thru 6 together)
8. Alchemy News	(Watson Alchemy)
9. Visual Recognition and Images	(Watson Visual Recognition)
10.Watson Conversations	(Watson Conversations)
11.Rank & Retrieve	(Watson Alchemy + Rank & Retrieve)

Chapter 7: Authentication

Learning Bluemix & Cognitive

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales

Git Repository: https://github.com/rddill-IBM/ZeroToCognitive