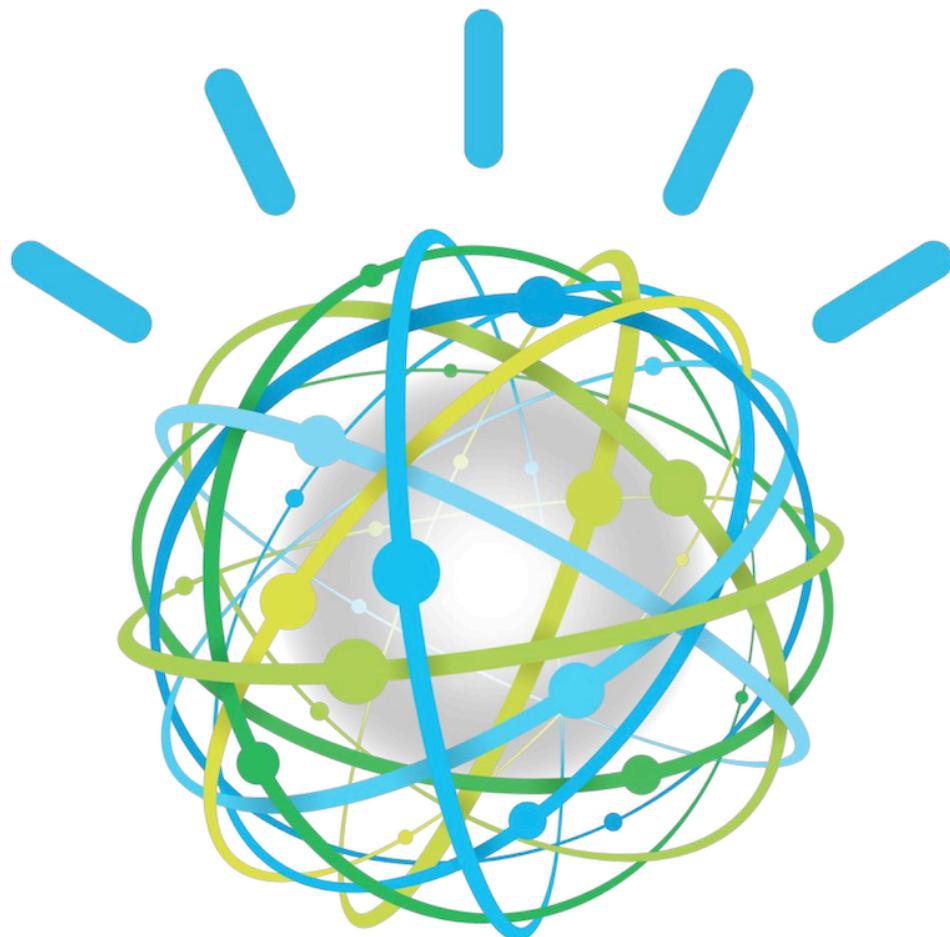


IBM Watson Solutions

Business and Academic Partners



Developing a Chat Bot Using the IBM Watson Conversation Service

Prepared by Armen Pischdotchian

Version 1.0 July 2016

Overview

What is Bluemix you ask? [Bluemix](#) is an implementation of IBM's Open Cloud Architecture, leveraging Cloud Foundry to enable developers to rapidly build, deploy, and manage their cloud applications, while tapping a growing ecosystem of available services and runtime frameworks. You can view a short introductory video here:

<http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html>

Additionally, for our academic partners, there are no-charge 12-month licenses for faculty and no-charge 6-month licenses for students - all renewable and NO CREDIT CARD required!

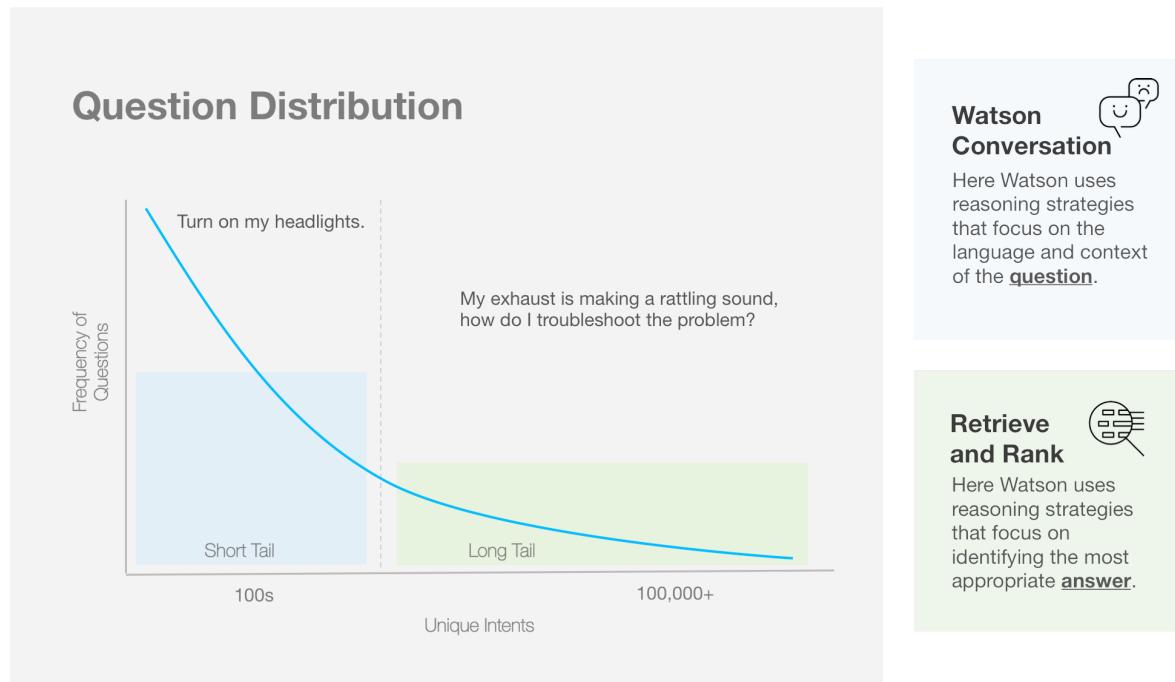
To get started, you will need to become an Academic Initiative member. Refer to this document for details:

http://it.husc.edu.vn/Media/TaiLieu/IBM_Academic_Initiative_for_Cloud_Process.pdf

The purpose of this guide is not to introduce you to Bluemix, that foundational knowledge is a prerequisite study on your part and you can obtain it from the links mentioned above. This guide is more of an instructional approach to working with the IBM Watson™ Conversation service where you can create virtual agents and bots that combine machine learning, natural language understanding, and integrated dialog tools to provide automated customer engagements. Watson Conversation provides an easy-to-use graphical environment to create natural conversation flows between your apps and your users. Creating your first conversation using the IBM Watson™ Conversation service entails the following steps:

1. Train Watson to understand your users' input with example utterances: Intents and Examples
2. Identify the terms that may vary in your users' input: Entities
3. Create the responses to your user's questions: Dialog Builder
4. Test and Improve

IBM Watson Conversation service is designed to answer the short tail of typical question distribution per below.



Consider the following scenario used in this guide: You are driving a cognitive enabled car. You ask in a natural way, the way you speak, for the car to do things for you, such as turn on wipers, play music and so forth. At one point, you ask the about the weather. In this workshop, you will connect the Conversation service with an external service from the Weather Company (<http://api.wunderground.com/?MR=1>) to inform you of real time actual weather in the exact location that you are sitting and performing this lab. The service obtains your location using the browser's geo-location capabilities.

It is strongly recommended that you watch this 14-minute video: <https://youtu.be/ELwWhJGE2P8>

At the end of this workshop, spend some time and consider what other services you can use to augment a better approach for bringing further cognition to your app; for example, Retrieve and Rank to extract information from specific documents (the long tail), or, you may want to include Speech to Text upfront and Text to Speech for the returned responses. Enjoy the cognitive journey you are about to undertake.

Prerequisite

This section provides instructions to help you get started quickly with the IBM Watson™ Developer Cloud services using Node.js as your programming runtime environment. To make it easy to get up and running with a functional application that uses the REST Application Programming Interface (API) for any Watson service, IBM provides a Node.js package with wrappers that simplify application development. The package includes simple command-line example applications to let you experiment with any of the available services. Complete the following steps:

- **Obtain Bluemix credentials:**

1. Direct your browser to the Bluemix home page: <https://console.ng.bluemix.net/home/>
2. Click **Sign Up** on the top right.
3. Enter requested information and click **Create Account**.
4. If you have received a promo code as a member of the Academic Initiative, enter it at this time; otherwise, you have 30 days to include the promo code.

- **Install the Node.js runtime:**

The default installation includes both the runtime and package manager. Make sure to include the installed binaries on your PATH environment variable after installation (typically, the default installation locations that the installer selects does the inclusion).

1. Direct your browser to the nodejs.org web site: <https://nodejs.org/download/>
2. Click **Downloads**.
3. Select and install the installer (not the binary) appropriate for your operating system.

- **Install the cf command line**

1. Direct your browser to a GitHub repository: <https://github.com/cloudfoundry/cli/releases>
2. Download and install the most recent installer appropriate for your operating system.
3. You may need to open Preferences → Security and Privacy → General tab (in Mac); unlock and change the Allow applications downloaded from **Anywhere**.

- **Download OS appropriate code-friendly editing tool:**

- If you are using a PC, we recommend using **Notepad ++**
- If you are using Mac, we recommend **Sublime Text**

- **Obtain the Weather Company API**

Currently, the following desktop browsers support the W3C Geolocation API:

- Firefox 3.5+
- Chrome 5.0+
- Safari 5.0+
- Opera 10.60+
- Internet Explorer 9.0+

1. Point your browser to the following site: <http://api.wunderground.com/>
2. From the **More** pull down menu, click **Weather API for Developers**.

The screenshot shows the Weather Underground homepage. At the top, there's an advertisement for 'Saatva' mattresses. Below it, the main navigation bar includes links for 'Maps & Radar', 'Severe Weather', 'News & Blogs', 'Photos & Video', 'Activities', and 'More'. The 'More' menu is open, showing options like 'Historical Weather', 'Personal Weather Station Network', 'Register Your PWS', 'WU Store', 'Mobile Apps', 'Daily Forecast Flyer', 'Weather API for Developers', and 'Site Map'. A banner at the bottom of the page reads 'Dangerously Hot Temperatures Expected This Week as Massive'.

3. Follow the on screen instructions to obtain the key. All fields must be filled. Below is an example from my experience (use <http://localhost:3000> for the purposes of this lab). You need your own key, which appears under the **Key Settings** tab. Refer to your email for verification.

The screenshot shows the 'GET YOUR API KEY' page. At the top, there's a 'Select a Key to Customize' dropdown set to 'Saa2ca764b80f41a - Chatbot'. Below it, the 'Edit API Key' form contains fields for 'Key ID' (Saa2ca764b80f41a), 'Project Name' (Chatbot), 'Company Website' (<http://localhost:3000>), 'Contact Phone', 'Contact Email' (apischdo@us.ibm.com), and an 'Update >' button. To the right, there's a 'Regenerate API Key' section with a warning about consequences (changing apps, resetting stats, and being unable to undo) and a checkbox for understanding. A 'Regenerate Key >' button is available. Below this is a 'Billing Information' section with 'Modify', 'Upgrade', 'Downgrade', and 'Cancel' buttons, and a 'View Billing >' button. At the bottom, there's a plan selection section with three options: 'STRATUS PLAN', 'CUMULUS PLAN', and 'ANVIL PLAN'. The 'TOTAL: \$0 USD per month' and a 'Purchase Key >' button are also present.

- **Extract the ConversationMaster.zip package from Github**

This package contains basic code that you will need to build

1. Direct your browser to a GitHub repository (no need to sign up): <https://github.com/>
2. Search for **bluemix-workshop**.
3. Scroll down and select: apischdo/*Bluemix-workshop*-assets
4. Download just the ConversationMaster.zip package (or perhaps all the workshops)
5. Extract the contents of the **ConversationMaster.zip** file in a temporary location on your local system.

Overview of the steps required to complete the workshop

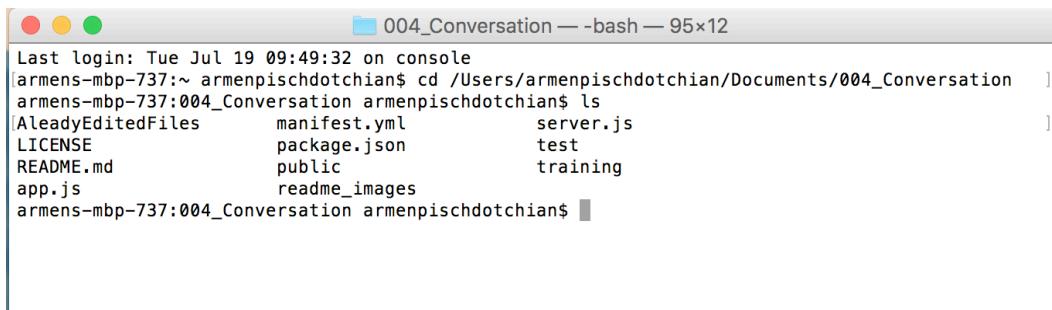
This table summarizes the steps that you need to complete to stand up an instance of the Watson Conversation service and integrate it an external service.

Step	Where do I go?	What do I do?
1	Github: https://github.com/apischdo/Bluemix-workshop-assets	Download the ConversationMaster.zip (you can also download everything that you see there). Alternatively, can also use the following command from a command prompt or your terminal: <code>git clone https://github.com/apischdo/Bluemix-workshop-assets.git</code>
2	Your machine, the terminal window	Run the command <code>npm install</code> from the same directory where you extracted or cloned the package
3	Bluemix	Create a Conversation service and launch the tooling
4	Conversation service tooling	Import the <code>car_workspace.js</code> file
6	Your machine: create a new system internal file	This <code>.env</code> file will contain the conversation ID and the username/password for the Conversation service
7	Your machine, the terminal window	From the same directory, run the <code>node server.js</code> command
8	Your machine: use the provided <code>weather.js</code> file	Update the <code>conversation.js</code> and the <code>app.js</code> files. You include the API from the Weather Company in this step.
9	Your machine	From the same directory, run the <code>node server.js</code> command to reflect your changes.
10	Bluemix:	Specify custom credentials
11	Your machine	Run the <code>cf push</code> command
12	Bluemix	Invoke the Routes URL and test your app on Bluemix

Using the Conversation service

Let's begin our journey with a few good housekeeping practices:

1. Create a top-level directory on your system.
2. Extract the contents of the ConversationMaster.zip file to your top level-working directory.
3. Open a command window or a terminal and change directory to the working directory (in this example, I named my top directory 004_Conversation (you pick your own directory name).

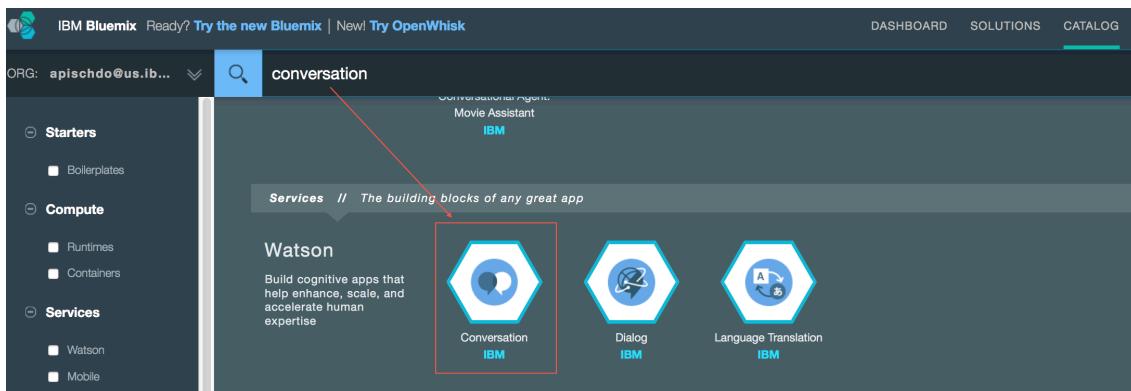


```
Last login: Tue Jul 19 09:49:32 on console
armens-mbp-737:~ armenpischdotchian$ cd /Users/armenpischdotchian/Documents/004_Conversation
armens-mbp-737:004_Conversation armenpischdotchian$ ls
AlreadyEditedFiles      manifest.yml          server.js
LICENSE                  package.json         test
README.md                public                 training
app.js                   readme_images
armens-mbp-737:004_Conversation armenpischdotchian$
```

4. From the same terminal, run the node package manager (npm) to install dependencies that node.js relies on for further processing of the code. Notice that you have a new folder named node_modules.

```
npm install
```

5. Login into Bluemix: <https://console.ng.bluemix.net> and use the Classic view.
6. Search for the **Conversation** service and click that tile.



7. Edit the Service name to something meaningful to you and click **CREATE**.

The screenshot shows the IBM Bluemix Catalog interface. On the left, there's a sidebar with service details: Conversation, IBM, Publish Date 07/12/2016, Author IBM, Type Service, Location US South, and a 'VIEW DOCS' button. The main area displays a brief description of the Watson Conversation service, three screenshots of its interface, and a 'Pick a plan' section. The 'Plan' table shows a 'Free' plan with 1000 API queries per month, up to 3 Workspaces, up to 25 Intents, 7 Days Production Data, and Shared Public Cloud. To the right, a modal window titled 'Add Service' is open, showing the 'Space' dropdown set to 'dev', 'App' dropdown set to 'Leave unbound', and the 'Service name' input field circled in red and containing 'Conversation-mycar'. Below it are 'Credential name' (Credentials-1), 'Selected Plan' (Standard), and a large green 'CREATE' button.

8. Creation of the service may take up to a minute or two. If seems to be spinning for a while, click the **Service Credentials** link in the left pane and copy and paste the username and password in your text pad.

The screenshot shows the 'Conversation-mycar' service page in the IBM Bluemix catalog. The sidebar has links for 'Manage', 'Service Credentials' (which is highlighted with a red arrow), and 'Service Access Authorization'. The main content area is titled 'Service Credentials' and contains a JSON snippet representing the service credentials. The 'NAME' column shows 'Credentials-1'. The 'SERVICE CREDENTIALS' section contains the following JSON:

```
{
  "credentials": {
    "url": "https://gateway.watsonplatform.net/conversation/api",
    "password": "0WYvfcmMohoy",
    "username": "0f1f8ecc8-1701-4ed1-a1f9-0e76bd6b0d69"
  }
}
```

A red circle highlights the 'username' and 'password' fields in the JSON snippet.

9. Click **Manage** just above the Service Credentials and you will see the front page of the service. Note that creating the service may take up to 3 minutes or so.
10. Click the **Launch Tool** button.
11. Login (if you had not logged in earlier) using the same credentials that you use to login Bluemix.
12. Click **Import**.
13. Navigate to the **car_workspace.json** file in the **training** folder and click **Open** to import the file.

The screenshot shows the Watson Conversation service instance page. At the top, it says 'Instances: Conversation-mycar'. Below that is a 'Import a workspace' dialog box with the instruction 'Select a JSON file then choose which elements from the workspace to import.' A file browser window is overlaid on the page, showing the file 'car_workspace.json' selected in the 'Documents' folder under 'Import'.

You must now provide three parameters to the code: conversation ID and the service credentials (username and password). Bear in mind that files that start with a dot “.” are hidden system files, but with Sublime or Notepad++, by opening the folder, instead of just a file, you can see those hidden files. And if you just can't see internal files, then no worries, just save the file and trust it is there.

14. In this example, select the top-level directory (004_Conversation) and open it with Sublime.

15. Open a new file with sublime and copy paste the below in that file:

```
#environment variables
WORKSPACE_ID=
CONVERSATION_USERNAME=
CONVERSATION_PASSWORD=
```

You are now ready to populate the environment variables:

16. Obtain the workspace ID by clicking the Workspace box and then click **View Details**.

17. Obtain the username and password of the Conversation service from the environment variables link.

Watson Conversation

Workspaces

Created: 7/20/2016, 2:24:18 PM
Last modified: 7/20/2016, 2:24:18 PM
Documentation
Bluemix

Workspace ID: 5df54f2f-1f3a-4cb1-aaef-f0a4427178b6

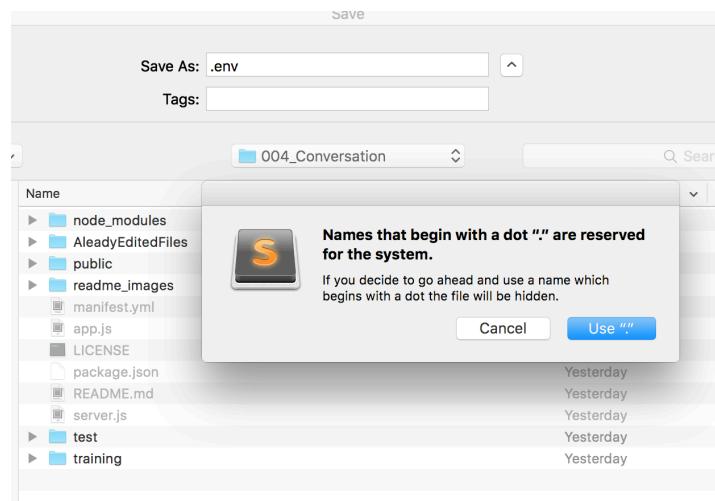
13 Intents 8 Entities 64 Dialog nodes

.env

```
1 # Environment variables
2 WORKSPACE_ID=5df54f2f-1f3a-4cb1-aaef-f0a4427178b6
3 CONVERSATION_USERNAME=01f8ecc8-1701-4ed1-a1f9-0e76bd6b0d69
4 CONVERSATION_PASSWORD=0NYvfcnMohoy
```

Get these values from the Conversation service tile: from environment variables

18. Save the file as **.env** (notice, there is a dot in front of the file name, this is a system internal file, typically hidden) and click **Use “.”**



19. Go back to the terminal window and run the following command:

```
Node server.js
```

20. Open a new browser tab and enter **localhost:3000** in the URL field and run a conversation similar to what you see in the screen capture below:

The screenshot shows a Watson Conversation interface with two main sections: "User input" and "Watson understands".

User input:

```

1 {
2   "input": {
3     "text": "What is the weather like tomorrow?"
4   },
5   "context": {
6     "conversation_id": "b05f7569-3df3-4695-bc77-ca36d9057c1c",
7     "system": {
8       "dialog_stack": [
9         "root"
10      ],
11      "dialog_turn_counter": 5,
12      "dialog_request_counter": 5
13    },
14    "defaultCounter": 0,
15    "reprompt": true
16  }
17 }
```

Watson understands:

```

1 {
2   "input": {
3     "text": "What is the weather like tomorrow?"
4   },
5   "context": {
6     "conversation_id": "b05f7569-3df3-4695-bc77-ca36d9057c1c",
7     "system": {
8       "dialog_stack": [
9         "root"
10      ],
11      "dialog_turn_counter": 5,
12      "dialog_request_counter": 5
13    },
14    "defaultCounter": 0,
15    "reprompt": true
16  }
17 }
```

The "User input" section contains a history of messages between a user and a bot. The user asks about the weather, turns on lights, turns off wipers, asks for music preferences, and asks about the weather again. The bot responds to each message. The "Watson understands" section shows the JSON representation of the user's messages and the bot's responses.

Notice that you have some work to do for the service to tell you how the weather is for your location, actual forecast exactly where you are sitting right now (well, rather where you will be when the next part of the workshop is complete).

Binding the Conversation service with an external API

For this next section you are going to use the geo-location capabilities of your browser to check the actual weather forecast and for that you will need to include the API key from the Weather Company web site that you obtained from the Pre-requisites section.

1. From your top-level directory navigate to the **AlreadyEditedFiles** folder and open the **weather.js** file. There are two distinct sections here: the initialization function (line 6 through line 33) is used in the **conversation.js** file and the remainder of the code (line 53 through line 103) is used in the **app.js** code plus *two* additional edits in the **app.js** file.
 2. Navigate to the **/public/js/** folder and open the **conversation.js** file.
 3. Copy the **init()** function from the **weather.js** file replacing the **init()** function in the **conversation.js** file.
 4. Save the **conversation.js** file and close it.

```
weather.js
1 //Client JS Code.. Add this to conversation.js, overwriting the init function
2 //new init function
3 // Initialize the module
4
5 function init() {
6   chatUpdateSetup();
7   if(navigator.geolocation){
8     navigator.geolocation.getCurrentPosition(geoSuccess, geoError);
9   }else{
10   console.log("Browser geolocation isn't supported.");
11   geoSuccess(position)
12 }
13 setupInputBox();
14
15 }
16
17 //private functions
18 function geoSuccess(position){
19   var context = null;
20   if(position && position.coords){
21     context = {};
22     context.long = position.coords.longitude;
23     context.lat = position.coords.latitude;
24   }
25   // The client displays the initial message to the end user
26   Api.sendRequest("", context);
27 }
28
29
30 //Sends in null to ask for zip code
31 function geoError(){
32   geoSuccess(null);
33 }
34
35
36 //modify the callback from conversation to call updateResponse(res, data)
37
38 // Line 1, Column 1
39
40 // Line 27, Column 20
41
42
43
44
45
46
47
48
49
50
51
52
53
54

conversation.js
18   }
19
20   // Publicly accessible methods defined
21   return {
22     init: init,
23     [inputKeyDown]: inputKeyDown
24   };
25
26   // Initialize the module
27 function init() {
28   chatUpdateSetup();
29   Api.sendRequest('', null);
30   setupInputBox();
31 }
32
33 // Set up callbacks on payload setters in Api module
34 // This causes the displayMessage function to be called when messages are sent
35 var currentRequestPayloadSetter = Api.setRequestPayload;
36 Api.setRequestPayload = function(newPayloadStr) {
37   currentRequestPayloadSetter.call(Api, newPayloadStr);
38   displayMessage(JSON.parse(newPayloadStr), settings.authorTypes.user);
39 }
40
41 var currentResponsePayloadSetter = Api.setResponsePayload;
42 Api.setResponsePayload = function(newPayloadStr) {
43   currentResponsePayloadSetter.call(Api, newPayloadStr);
44   displayMessage(JSON.parse(newPayloadStr), settings.authorTypes.viewer);
45 }
46
47
48 function chatUpdateSetup() {
49   var input = document.getElementById('textInput');
50   var dummy = document.getElementById('textInputDummy');
51   var padding = 3;
52
53   if (dummy === null) {
54     var dummyJson = {
```

5. You are now ready to edit the app.js file. Open the **app.js** file.
 6. Include the `var http = require('http');` variable with the other variables at the top of the **app.js** file (line 24 will do nicely).

7. Copy lines 53 to the end (line 119) from the `weather.js` file replacing lines 76 through 100 (the `updateMessage` function) in the `app.js` file. If the line numbers differ, search for `updateMessage` (Cntrl+F).

The screenshot shows two code editors side-by-side. The left editor contains `weather.js` with code related to fetching weather data from `wunderground.com`. The right editor contains `app.js`, which handles the response from `weather.js` and adds confidence levels to the output. A red box highlights the `function updateMessage(response)` block in `app.js`.

```
weather.js
40 /**
41 * Updates the response text using the intent confidence
42 * @param {Object} input The request to the Conversation service
43 * @param {Object} response The response from the Conversation service
44 * @return {Object} The response with the updated message
45 */
46
47 function updateMessage(res, input, data) {
48   if(checkWeather(data)){
49     var path = getLocationURL(data.context.long, data.context.lat);
50
51     var options = {
52       host: 'api.wunderground.com',
53       path: path
54     };
55
56     http.get(options, function(resp){
57       var chunkText = '';
58       resp.on('data', function(chunk){
59         chunkText += chunk.toString('utf8');
60       });
61       resp.on('end', function(){
62         var chunkJSON = JSON.parse(chunkText);
63         var params = [];
64         if(chunkJSON.location) {
65           var when = data.entities[0].value;
66           params.push( chunkJSON.location.city );
67           var forecast = null;
68           if( when == 'today' ) {
69             forecast = chunkJSON.forecast.txt_forecast.forecastday[0].fct
70           } else if ( when == 'tomorrow' ) {
71             forecast = chunkJSON.forecast.txt_forecast.forecastday[1].fct
72           } else{
73             forecast = chunkJSON.forecast.txt_forecast.forecastday[2].fct
74           }
75           params.push ( forecast );
76
77           data.output.text = replaceParams ( data.output.text, params );
78
79         }
80       });
81     });
82   }
83 }

app.js
67   return res.json(updateMessage(data));
68 }
69 });
70 */
71 /**
72 * Updates the response text using the intent confidence
73 * @param {Object} response The response from the Conversation service
74 * @return {Object} The response with the updated message
75 */
76 function updateMessage(response) {
77   var responseText = null;
78   if (!response.output) {
79     response.output = {};
80   } else {
81     return response;
82   }
83   if (response.intents && response.intents[0]) {
84     var intent = response.intents[0];
85     // Depending on the confidence of the response the app can return different responses
86     // The confidence will vary depending on how well the system is trained
87     // a class/intent to the input. If the confidence is low, then it suggests
88     // user's intent . In these cases it is usually best to return a disclaimer
89     // ("I did not understand your intent, please rephrase your question")
90     if (intent.confidence >= 0.75) {
91       responseText = 'I understand your intent was ' + intent.intent;
92     } else if (intent.confidence >= 0.5) {
93       responseText = 'I think your intent was ' + intent.intent;
94     } else {
95       responseText = 'I did not understand your intent';
96     }
97   }
98   response.output.text = responseText;
99
100 }
101
102 module.exports = app;
```

8. Search for **return res.json** in the app.js file and edit it to reflect the following code:
`updateMessage(res, payload, data);`

```
weather.js
44      if(position && position.coords) {
45        context = {};
46        context.long = position.coords.longitude;
47        context.lat = position.coords.latitude;
48      }
49      // The client displays the initial message to the end user
50      Api.sendRequest("", context);
51    };
52  }
53  //Sends in null to ask for zip code
54  function geoError(){
55    geoSuccess(null);
56  }
57
58 //modify the callback from conversation to call updateResponse(res, data)
59
60 //NODE JS CODE.. ADD this to app.js
61 // Add var http = require('http'); at the top of the file..
62 //update "return res.json( updateMessage( payload, data ) );" to be....
63
64 //Add the following new functions
65
66 /**
67 * Updates the response text using the intent confidence
68 * @param {Object} input The request to the Conversation service
69 * @param {Object} response The response from the Conversation service
70 * @return {Object} The response with the updated message
71 */
72
73 function updateMessage(res, input, data) {
74   if(checkWeather(data)){
75     var path = getLocationURL(data.context.long, data.context.lat);
76
77     var options = {
78       host: 'api.wunderground.com',
79
80
81     var options = {
82       host: 'api.wunderground.com',
83       path: path
84
85     http.get(options, function(resp){
86       var chunkText = '';
87       resp.on('data', function(chunk){
```

```
app.js
50
51   workspace_id: workspace,
52   context: {}
53 };
54 if (req.body) {
55   if (req.body.input) {
56     payload.input = req.body.input;
57   }
58   if (req.body.context) {
59     // The client must maintain context/state
60     payload.context = req.body.context;
61   }
62 }
63 // Send the input to the conversation service
64 conversation.message(payload, function(err, data) {
65   if (err) {
66     return res.status(err.code || 500).json(err);
67   }
68   return res.json(updateMessage(payload,data));
69 });
70
71 /**
72 * Updates the response text using the intent confidence
73 * @param {Object} response The response from the Conversation service
74 * @return {Object} The response with the updated message
75 */
76
77 function updateMessage(res, input, data) {
78   if(checkWeather(data)){
79     var path = getLocationURL(data.context.long, data.context.lat);
80
81     var options = {
82       host: 'api.wunderground.com',
83       path: path
84
85     http.get(options, function(resp){
86       var chunkText = '';
87       resp.on('data', function(chunk){
```

9. Include the API key from the Weather Company in the bottom of the **app.js** file.
 10. Save the **app.js** file.
 11. Run the node `server.js` command again and if need be, (Ctrl+C) to end the current terminal session.

12. Ask the following questions:

- Turn on my wipers
- What is the weather like today?

The screenshot shows a Watson Assistant interface. On the left, there is a text input field containing "type something". Above it, a purple sidebar displays the text: "Hi. It looks like a nice drive today. What would you like me to do?". Below the sidebar, two blue rounded rectangular buttons are shown: one labeled "turn on the wipers" and another labeled "what is the weather like today". To the right of these buttons, the text "Ok. Turning on the wipers." is displayed. Further down, the text "Unfortunately I don't know much about the weather..I'm still learning." is shown. At the top right of the interface, there is a circular icon with a left and right arrow symbol.

User input

```
1 {
2   "input": {
3     "text": "what is the weather like today"
4   },
5   "context": {
6     "long": -71.4708425,
7     "lat": 42.549521999999996,
8     "conversation_id": "d56af184-882f-4b18-967a-069bc116084e",
9     "system": {
10       "dialog_stack": [
11         "root"
12       ],
13       "dialog_turn_counter": 2,
14       "dialog_request_counter": 2
15     },
16     "defaultCounter": 0,
17     "reprompt": true
18   }
19 }
```

Watson understands

```
1 {
2   "input": {
3     "text": "what is the weather like today"
4   },
5   "context": {
6     "long": -71.4708425,
7     "lat": 42.549521999999996,
8     "conversation_id": "d56af184-882f-4b18-967a-069bc116084e",
9   }
10 }
```

Notice that it understands the concept of weather, but the app cannot fetch that data at this point.

You are now ready to incorporate these changes in the dialog structure.

Updating Dialog

Working with Intents, Entities and Dialog is entirely separate discipline and subsequent workshops will elaborate on all facets of creating a robust dialog, for the purposes of this lab, you will edit the Entities and the Dialog.

1. Go to the **Car_Dashboard** conversation and click the **Entity** tab.
2. Click the plus sign and add an *Entity* of **day** (case matters, use lower case).
3. Add the value of **today** for the Entity of day (case matters, use lower case).
4. Add synonyms such as **this afternoon** and **tonight** (case does not matter here, pick your own synonyms).

The screenshot shows the 'Entities' tab selected in the Car_Dashboard conversation. A new entity '@day' is being created. The 'Value' field contains 'today'. In the 'Synonyms' field, 'this afternoon' and 'tonight' are listed, separated by a plus sign. A green 'Done' button is visible at the top right.

5. Click the plus sign and add another entity *value* of **tomorrow** (case matters, use lower case)
6. Add synonyms such as **tomorrow evening** and **tomorrow night**.

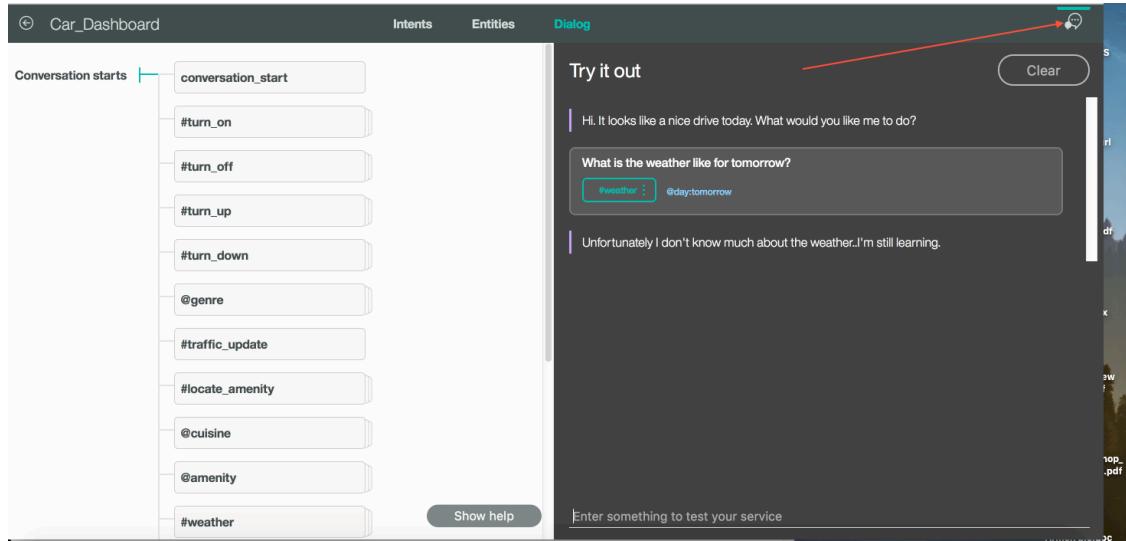
The screenshot shows the 'Entities' tab selected in the Car_Dashboard conversation. The entity '@day' now has a value 'tomorrow'. In the 'Synonyms' field, 'tomorrow evening' and 'tomorrow night' are listed, separated by a plus sign. Other entities like 'this afternoon' and 'tonight' are also visible in the list.

7. Click **Done**.
8. Note, # sign signifies Intent and @ sign signifies Entity. The end result looks like the image below:

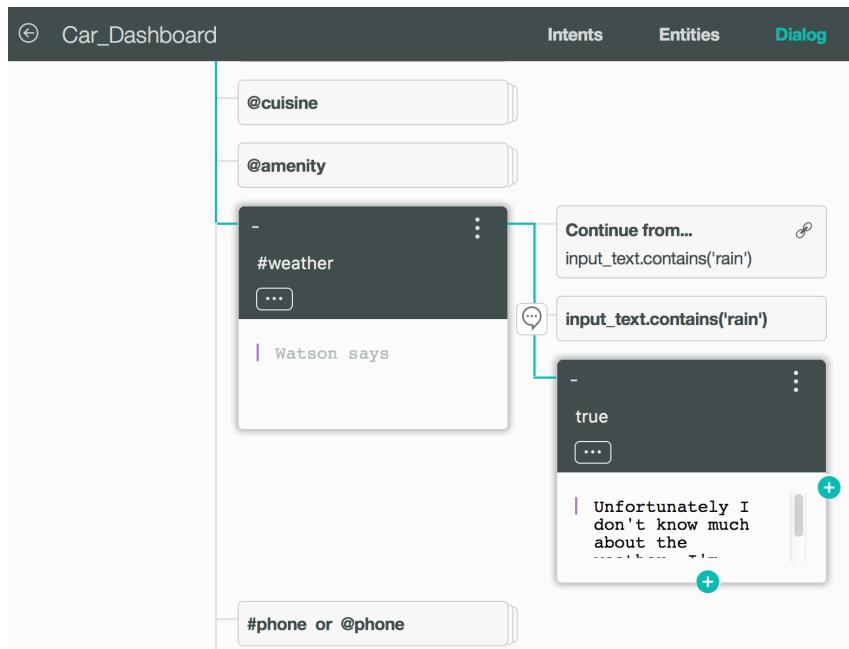
The screenshot shows the 'Entities' tab selected in the Car_Dashboard conversation. The entity '@day' is listed with values 'today, tomorrow'. A green 'Create' button with a plus sign is visible at the top left. At the bottom right, there are filters for '9 entities' and 'Sort by: Newest'.

9. Let's update the Dialog flow. Click the **Dialog** tab.

10. Click the chat icon to the top right corner of the page and do a trial run and ask the system “how is the weather tomorrow?” The result should appear as below:

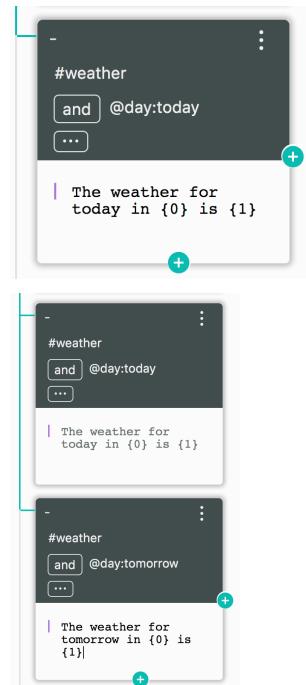


11. Close the chat window and click the #weather Dialog node and expand the tree, notice where the chat got its response!



12. Click the three dots in the #weather node and delete the #weather node.
 13. Click the amenity node (or any node) and then click the plus sign in the bottom of the node, to add another Dialog node.

14. Type and then select the **#weather** for Intent.
15. Click the three dots and specify the condition of **@day:today**
16. Add the dialog: The weather for today in {0} is {1}
17. There is the concept of conditions, if the condition is true then the dialog code get's executed.
18. Click the plus sign underneath the dialog node you just created and add another node: **#weather**
19. Add the condition of tomorrow: **@day:tomorrow**
20. Include the same script:



Keep an eye on the message where it says **Watson is training on your recent changes**. It will become green and state that training has finished and disappear shortly thereafter.

You are now ready to deploy your app to Bluemix.

Deploying your app to Bluemix

An easy way to deploy your app is by including an app name in the manifest.yml file and then using the cf push command to deploy the app onto Bluemix. However, this guide will make greater use of Bluemix capabilities in ensuring that your app is deployed properly.

1. Back to Bluemix, click Dashboard and click **CREATE APP**.
2. Follow the onscreen instructions until your placeholder app fully stages and is available, that means the Route link is live. This may take a minute or two.
3. Click **BIND A SERVICE OR API**.
4. Select the conversation service you created in the beginning of this workshop. Allow the restaging process to complete.
5. Click the **Environment Variables** link in the left pane.
6. Click **USER-DEFINED**.

- For name, specify: **WORKSPACE_ID** (uppercase please) and for Value, copy and paste the **WORKSPACE_ID** from the Conversation tooling UI, from the Details section of the work space.

Name	Value	Actions
WORKSPACE_ID	1d8fe748-e521-49d3-baa5-0fb3ef65ea65	

- Click the **Start Coding** link from the left panel
- Follow the instructions from Step 6 onward and you perform these in the same terminal or Command prompt where the app.js and the server.js files reside.
- Allow enough time for the app to upload and restage
- Click the **Routes** URL and test your app.

Congratulations, you just deployed a chat bot onto Bluemix.