**Multichannel Chatbot Technical Guide**

**Table Contents**

------------ **Introduction** ------------

------------ **Prerequisites** -----------

------------ **Installation** ------------

------------ **Parts of UMA Chatbot** -----------

------------ **API Handler** -----------

------------ **Project Structure** -----------

------------ **Components** ------------

------------ **Summary of chatbot** -----------

**UMA Chatbot:**

**UMA Chatbot** is a Doc retrieval multichannel chatbot which helps user

Interact with bot with the greetings as well as search relevant documents as per user query. User can see summary of the documents and there is share option.

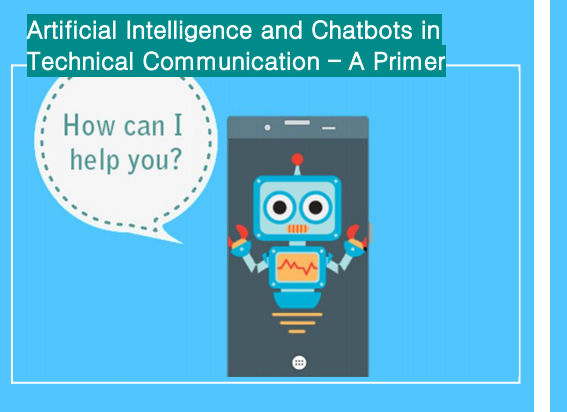
There are two modes of search:

1. Document Search

2. CRM Search

You can change your search mode by type Switch Avatar on input box or

There is a Persistent menu options on messenger You will find option Switch Avatar select it.

****

**Prerequisites**

* Visual Studio Code
* Node.js 8+
* Python 2.7 to 3.0 (need for dependency)
* JavaScript
* ReactJS (Need for Webview frontend)
* Knowledge of restify and asynchronous programming in JavaScript
* Git

**Installation**

* Install Visual studio code into your local system, you can directly download it from Below link.

<https://code.visualstudio.com/download>

* **Node.js** Download 8+ version of Node.js installable archive file

Installation on Windows:

Use the MSI file and follow the prompts to install the Node.js. By default, the installer uses the Node.js distribution in C:\Program Files\nodejs. The installer should set the C:\Program Files\nodejs\bin directory in window's PATH environment variable. Restart any open command prompts for the change to take effect.

* **Python**

Windows Installation

Use below link for installation

<https://www.tutorialspoint.com/python/python_environment.htm>

* You need account on git.

**Parts of UMA Chatbot**

**UMA Chatbot:**

There are two parts

1. Ust\_Doc\_Retrieval Multichannel
2. Ust\_Doc\_Webview

**Ust\_Doc\_Retrieval Multichannel** is responsible for take user input then bot forwards message to NLP for processing user input and response back with intent type. Bot sends message back to user. Only text conversation is done here.

**Ust\_Doc\_Webview**  In our conversation with bot we want bot should response with

Documents or images or we want show slide or popup on messenger then we want Webview.

The Messenger Platform allows you to open a standard WebView, where you can load webpages inside Messenger. This lets you offer experiences and features that might be difficult to offer with message bubbles.

**Displaying the Webview**

**You can open the WebView with any of the following:**

* The [persistent menu](https://developers.facebook.com/docs/messenger-platform/reference/messenger-profile-api/persistent-menu)
* [A URL button](https://developers.facebook.com/docs/messenger-platform/reference/buttons/url) inside a message, including the generic template.

**Handler API**

All the user inputs Either it is normal greetings or query for document search, to get intent type we are using API.

Below are major API we are using:

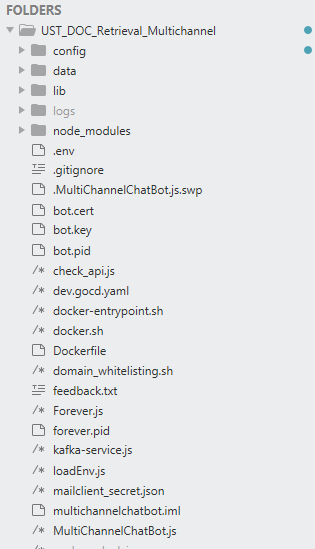
**Emotion Handler** is handle all the greetings parts, for example hi, hello.

**Intent Handler** is handle all document search and retrieve documents for user.

(All type of documents search it can be pdf , doc type file.)

**Summary** API is use or get Summary of searched documents.

**Project Structure**

****

**Components**

**This Project has divided with several components**

**Below are those subfolders:**

**Config**

**Data**

**Services**

**Dialogs**

**NodeModules**

Config:

The **config** subfolder is where we store all our bots whether production or stage in or developer. Be careful while working on files in this folder. *Never use any bot(file) other than yours i.e, dev.json in this case.*Your local bot runs dev.json. Hence any changes you want to implement in your bot you have to change in dev.json only. \* If you add any other bot, make sure it has been also added to .gitignore.

Data:

The **Data** subfolder is where we stored all hard coded data for respective CRM contacts details and images for CRM search.

Services:

**Services** are Objects which get instantiated only once during the lifetime of an application. They contain methods that maintain data throughout the life of an application. The main objective of a service is to organize and share business logic, models, or data and functions with different components.

The **services** subfolder is where we kept all services files. Below are those functionalities is done by Services.

* Check User’s current state and session (i.e. User is already logged in or previously chat with bot or new user).
* Store user’s Previous conversation data into local storage, when logged in user again start with the conversation then bot can make better conversation.
* Clear cache according to cacheCleanScheduler.
* **Textservice.js** is helps response user with text message.
* **UtilService.js** is where all functionality is written for Create Carousel cards, prev and next documents, pagination, get user’s Facebook profile for given page specific Id and send feedback.
* **UserService.js** is mainly containing methods for Add user, send feedback, reset user state and much more.
* **ApiService.js**  is containing methods for get emotion, intent, summary, results by which

We can complete our API requests and appropriate response.

* Spreadsheet

Dialogs:

Dialogs are an Array of function which contain the messages that are sent to the Users. Each function in the Array waits for user input and triggers the next function in the Array, after receiving the user's message. Since we are allowing the user to enter any message anywhere in the conversation flow, our dialogs are usually just 1 element long.

For creating dialogs, one needs to consider:

* how many ways user can trigger this dialog (this is where entity Type is checked)
* how many ways user can exit the dialog (user can always short circuit the dialog, but that’s ok)

For e.g. a dialog that displays a carousel can be triggered by following ways:

* User entered a free form text that had an entity and gender
* User pressed the Show More button
* User is returning from a Brand, Price, Colour or Size filter select dialog

The **dialogs** subfolder is where we kept all dialogs files. Below are those functionality is done by dialogs

* In this UMA Chatbot there, one is greetings conversation another is document retrieval i.e user ask bot with query for retrieve specific document.
* There are several dialogs for bot response with user query, Those are greetings, document retrieval, add filter, persistent menu, api switch, feedback, help, switch avatar,

crm search, summary of the documents etc.

Recognizers:

Translation of message into 'intent' is done by a recognizer. Recognizer is a function that checks the incoming message (or attachments in the message) and determines if the message is of certain type. There are specialized recognizers for each type of message:

* Postback recognizer : checks if the incoming message is a post back button press
* Quick reply recognizer : checks if the message is a quick reply button press
* ExactMatch recognizer : checks if the message is one of the standard keywords
* ImageURL recognizer : checks if the message is an image url
* Intent recognizer : checks if the message returns a non null result from the intent api
* Emoji recognizer : checks if the message is an emoji
* etc.

Each message is passed through each of the recognizers one after the other (in order) untill a recognizer recognizes a message. Once a recognizers recognizes a message, recognizers below it are not triggered.

Each 'intent' is mapped to a Dialog (like a Table of Contents in a book) which is triggered once a match is found by the recognizer. Recognizers sends information to the corresponding dialog using a standard format:

{

intent **:** 'INTENT.NAME', *// Intent name is many-to-one mapped to a dialog*

score **:** 1, *// Score should be either 1 or 0*

entities **:** [ { *// 'entities' contains all the information required by the dialog to display messages to the user*

entity **:** '', *// entity is generally the user input text (e.g. name of the category that user tapped)*

entityType **:** '', *// entityType is a collective noun for the type of user input (e.g. GUIDE.ME.CATEGORY if user pressed one of the many category button displayed)*

data **:** {} *// Any arbitrary data, that may be passed by the preceding dialog or the recognizer to the next dialog*

}]

}

Node Module:

Node modules is require for use node.js packages in our project like restify, botbuilder.

**Basic Workflow**

1. Bot start with MultichannelChatbot.js there you need to import **botbuilder** module.
2. Need to create a http server with **restify** for listen request.
3. Enter into **xpresso-bot.js** there it will load **recognizers** then recognizers do rest of the part like call **NLP API** for for get user input belongs to which intent. After that according to intents match route with the dialog file.
4. Dialog file is responsible for what will be the response back to user.

Best way to visualize the Bot code flow is to think of it as a book. In a book, Table of Contents is used to lookup and go to the corresponding chapter.

Similarly, in the bot, every single incoming message is translated into an 'intent'. This intent is used to lookup and trigger the corresponding dialog that the user is shown.

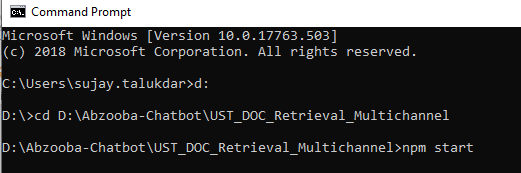
loggly

We use Loggly service for maintaining all our logs. Go to loggly(loggly.xpresso.com) Each & every query made to the bot and the reults/errors replied by the bot are saved to to loggly for our inspection. You can check for any particular error/query by using the keywords of query. And the time is also logged for each and every query. Loggly is very useful in detecting the bugs & working on them efficiently.

**Create a Bot**

**To create your bot and initialize its packages**

1. Open a terminal or elevated command prompt.
2. Go to your existing bot directory

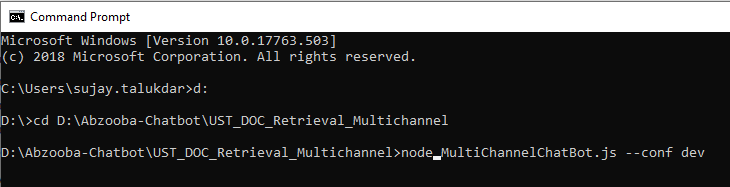


1. Ensure your version of npm is up to date
2. Use npm start command in your project directory.

**Start Bot:**

1. Open a terminal or elevated command prompt.
2. Go to your project directory Type **node MultichannelChatbot.js –conf dev**

**Shown Below:**



Config\dev.json

We keep our all configuration in dev.json file like Microsoft APP ID, Password which is generated on create bot on azure service. Facebook access token as well as All dependent API and text messages for greetings.



Bot Cycle:

1. Take user input on messenger check the user state i.e user is previously logged in or new user.
2. Call recognizer, recognizer is responsible for understand user intent, recognizer file is do the full process like call NLP API and this API is send output back user input is belongs to which intent.
3. According to intent map a dialog file.

Bot Execution flow for introduction:

1. Bot start with MultichannelChatbot.js (Create restify server, initialize bot and create persistent menu.)

2. Xpressobot.js -> (call recognizer according to text, check users state and get intent type then route dialog according to intents match )

3. Prompt2-recognizer.js ->

4. StateService.js ( For checking current state of the user ) ->

5. Emoji-recognizer.js ->

6. ApiService.js ->(This service is responsible for handle API calling.)

7. Emoji-dialog.js ->

8. Introduction-active-user.js ->

9. TextService.js ->

10. BotService.js (End response to user with result)

<https://drive.google.com/drive/folders/1fYtVIJ2AxSwEhe-FnHLPWpY5MfbS9N0W>

**How chatbot recognizes User’s input**

In our days, there is a wide variety of platforms that provide API for natural language processing. This inlcudes Wit.ai from Facebook, Watson from IBM, Api.ai from Google, LUIS from Microsoft, and many others. The capabilities that they provide may vary, but they all are based on several fundamental concepts (the terminology might vary depending on the platform):

**Intent**, which is the goal that the user wants to achieve

Entity, which is a parameter of the goal. Entities differentiate by the type of information they represent. For example, an entity may represent location, date/time, type of requested data, etc.

**Context**, which is a short term memory maintained during the entire conversation. In the example above, when the user answers “First name, last name, and address”, the chatbot understands that the intent is still the same – to retrieve information about customers – so the user’s answer actually means “how to retrieve the first name, last name, and address of the customer”.

**Slot**, which is the information that the chatbot needs to get in order to provide an answer. In our example, all additional information that the chatbot requests – what information exactly about each customer needs to be retrieved, whether the retrieval has to be scheduled, in what programming language the user wants to code, etc. – represent slots. If a slot is empty (that is the user didn’t provide some information), the chatbot asks a question requesting for this information. Slots is a fundamental concept for building conversations.

If we get back to the example above when the user says “How can I retrieve a list of customers who have birthday in this month?”, here’s how the chatbot can deconstruct this phrase to identify the intent and entities:

This is how this question would be translated into the machine-understandable language:

{

“intent”: “retrieval”,

“type”: “customers\_list”,

“criteria”: “date\_time”

}