BOT Platform – Overview

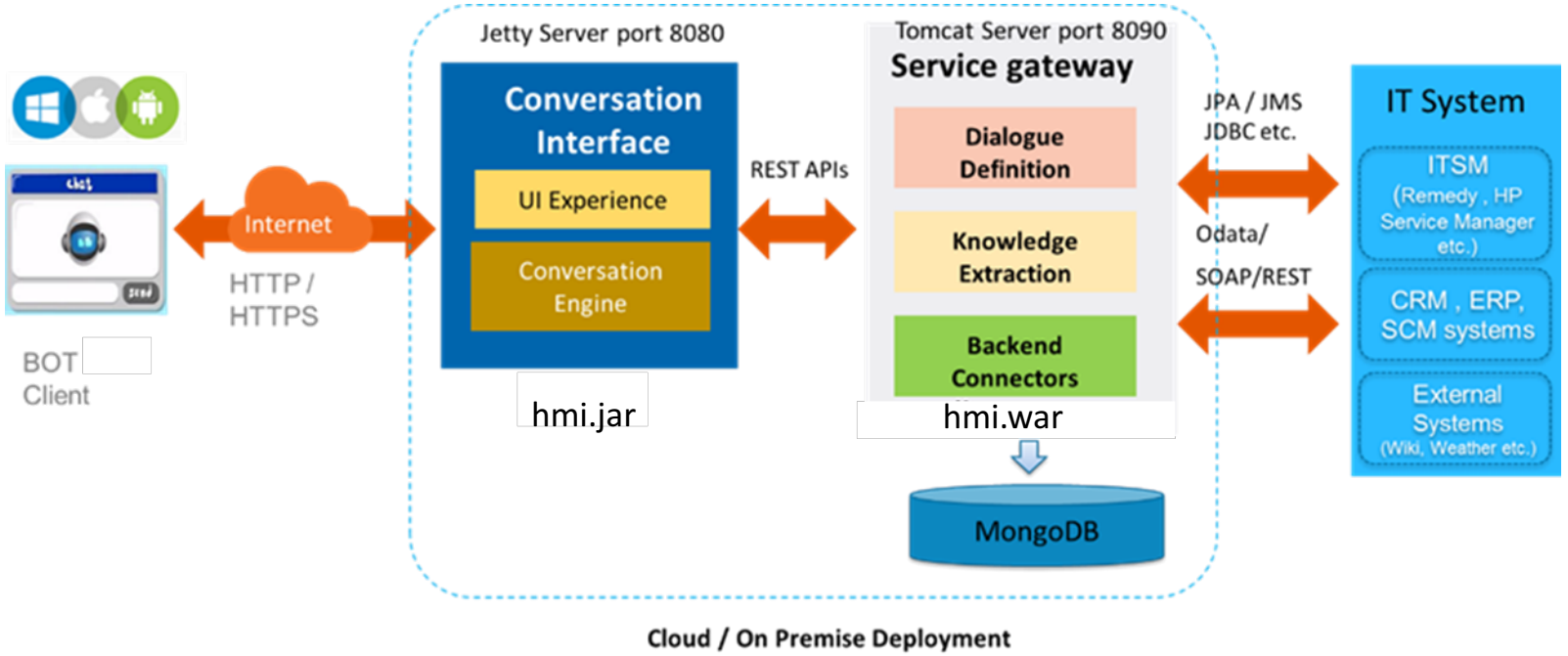
Release V 1.0_6
May-2018

Revision History

CR-101 (7 Apr’18) - Added the IGNORE_PREV_TASK flag in bot.propoerties so that user can still switch to new task but can ignore the previous context.
CR-102(8 Apr’18) – Added the task level CACHE mechanism to transfer info across tasks. (see appendix-7)
CR-103(9 Apr’18) – Added clarifyQuestion XML tag to provide more natural response in case user fail to answer.
CR-104(20 Apr’18) – You can now set the ITO value through groovy action (slide -13)
CR-105(11 May’18) – GDPR compliance mask personal data in user utterance before storing it in Logs
CR-106(15 May’18) – fallbackQuestion tag in DDF now supports use of ITO values. (e.g. Hi %loginUser , How are you ?) (slide-11)
CR-107(22 May’18) – support for Arabic language , minor fixes and updated python scripts.

BOT Platform – Overview

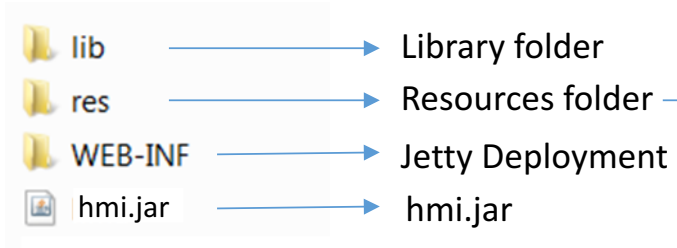
The BOT platform is Client-Server architecture , client being browser , app or messenger like FB, Slack.
The BOT server is completely REST based and run on Jetty Server. This document describes the core bot engine available in form Java library (viz. hmi.jar)



Prerequisite –
Java 1.8+ and all system variables (JAVA_HOME etc.) are set.
Apache-tomcat-9.0 or later on host server.
MongoDB 3.2.8 or later
Anaconda 4.2.0 (in case we need to run knowledge extraction python ML program)

BOT Platform – Getting Started

1. Unzip the hmi.zip to a folder



- lib → Library folder
- res → Resources folder
- WEB-INF → Jetty Deployment
- hmi.jar → hmi.jar

Resources folder contains:

- autocomplete → Folder contains required scripts for autocomplete feature when user types
- bots → Folder contains script to handle generic conversations – Hi, How are you ?etc.
- classifiers → Stanford NLP CRF (conditional random Field) classifier for NER
- config → Config folder that contains bot.properties file and license properties
- dialogues → Contains different dialogue XML files
- dictionary → Contains stopwords, slang words, domain synonyms and soda files
- entities → Stores all data for NER and custom parsers.
- events → Stores all event dialogues that can be triggered externally
- html → Your browser client code can reside here (https://10.77.36.45:8080)
- intents → Contains BOT intent training related data and models (refer Appendix-3)
- keys → Contains your SSL certificate JKS and license key and signature
- logs → Contains log that can be used for BOT monitoring & training
- temp → Used for temporary storages
- upload → The upload documents will be stored in this folder

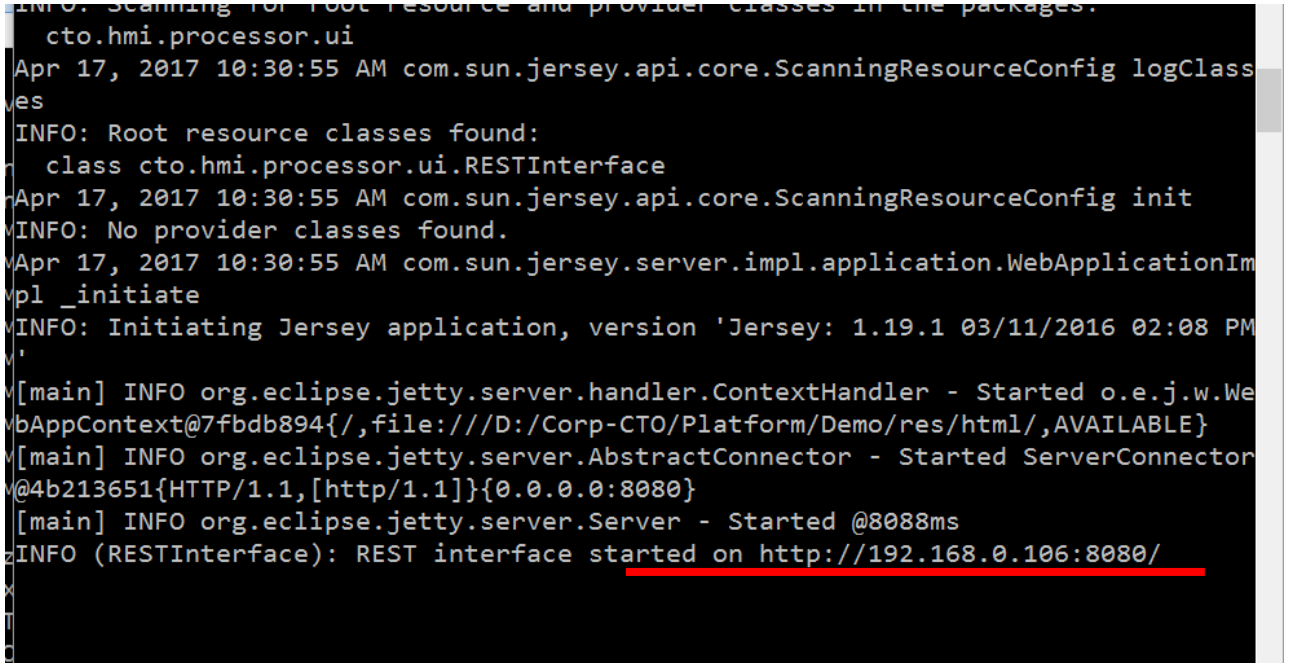
2. Running the BOT server on local console as a standalone (for Testing) or as REST based Server (for Production)

C:\HMI\Demo> java hmi.jar –i rest –r insurance → -i rest => for REST options and –r insurance is for running dialogue insurance.xml

C:\HMI\Demo> java hmi.jar –i console –r insurance → -i console => for CONSOLE options and –r insurance is for running dialogue insurance.xml

BOT Platform – Calling BOT server APIs ..1/4

1. Deploy BOT using the command provided on earlier slide
- Once BOT Server is started it runs Jetty server at default port 8080

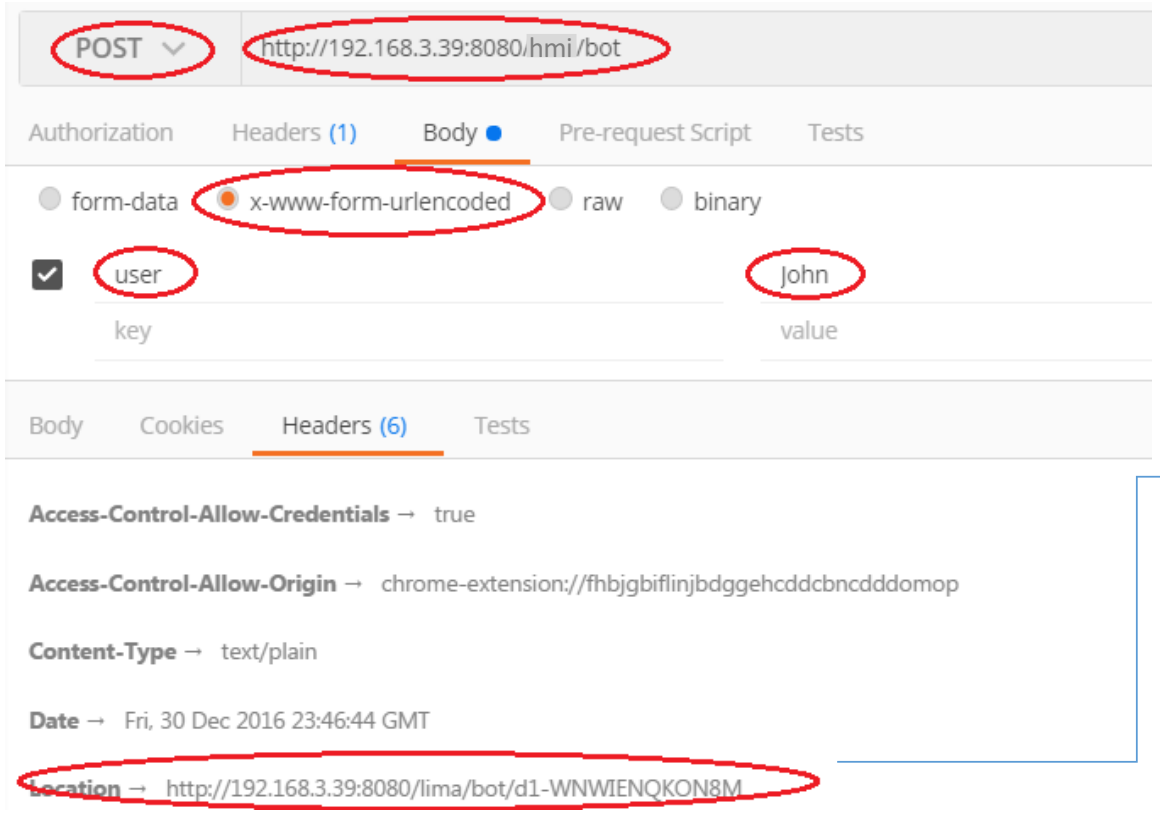


It will show the url and port details of jetty server on console
http://192.168.0.106:8080/ - (in this case)

To start the instance of Bot you need to append /hmi/bot to above url

BOT Platform – Calling BOT server APIs ..2/4

2. Creating BOT instance from REST Client
Enter below details to your RESTClient tool (postman or soapUI etc.)
Method – POST
URL - http://192.168.3.19:8080/hmi/bot
Param - url-encoded-body parameter – key => user value = >John



Once this request is sent , BOT will create a unique URL for every user and returns it in HTTP header response.

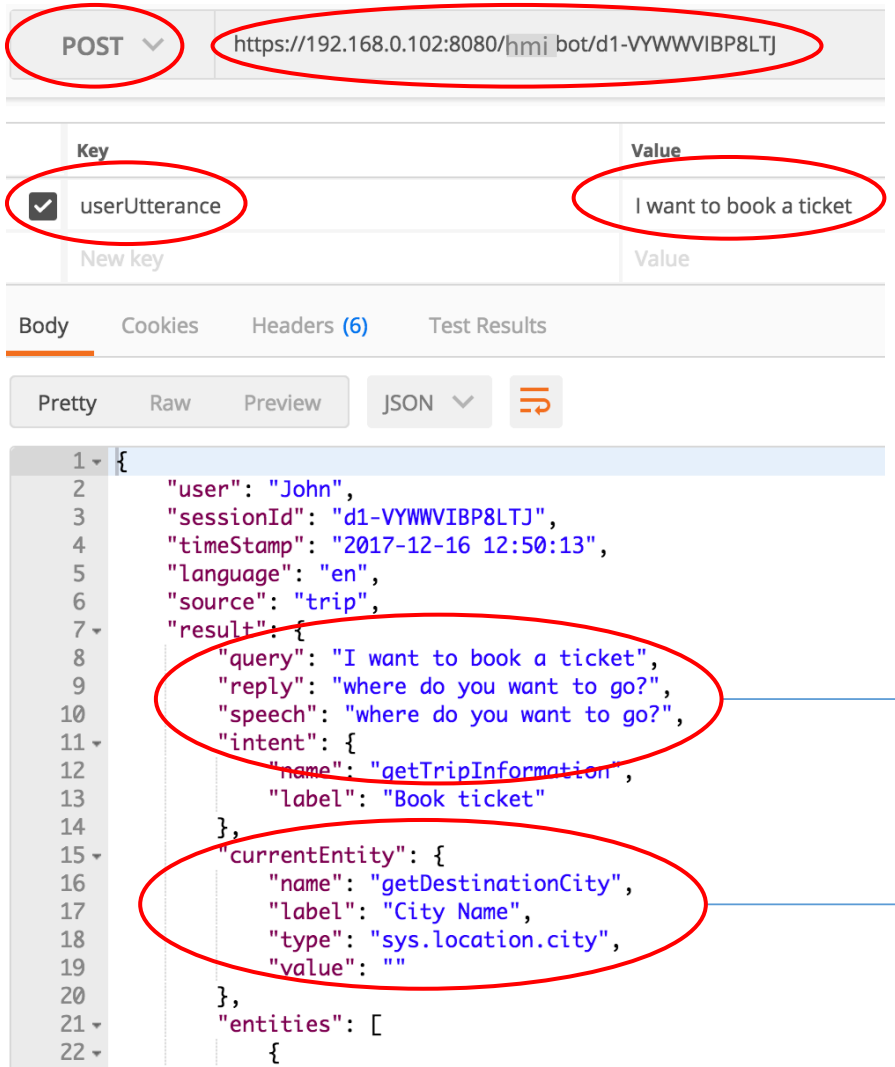
Copy that url
http://192.168.3.39:8080/hmi/bot/d1-WNWIENQKON8M

#curl -k -i -X POST --data-urlencode "user=John" http://192.168.3.39:8080/hmi/bot

BOT Platform – Calling BOT server APIs ..3/4

3. Start conversation

Using the URL that was obtained in step -2 , call following method to initiate the conversation



hmi

Method – POST

URL - http://192.168.3.39:8080/hmi/bot/d1-WNWIENQKON8M

Param - url-encoded-body parameter – key => userUtterance

value = > book a ticket

Once this request is sent , Bot will process the userUtternace and will send JSON response with a reply -> “OK, Where do you want to go?”

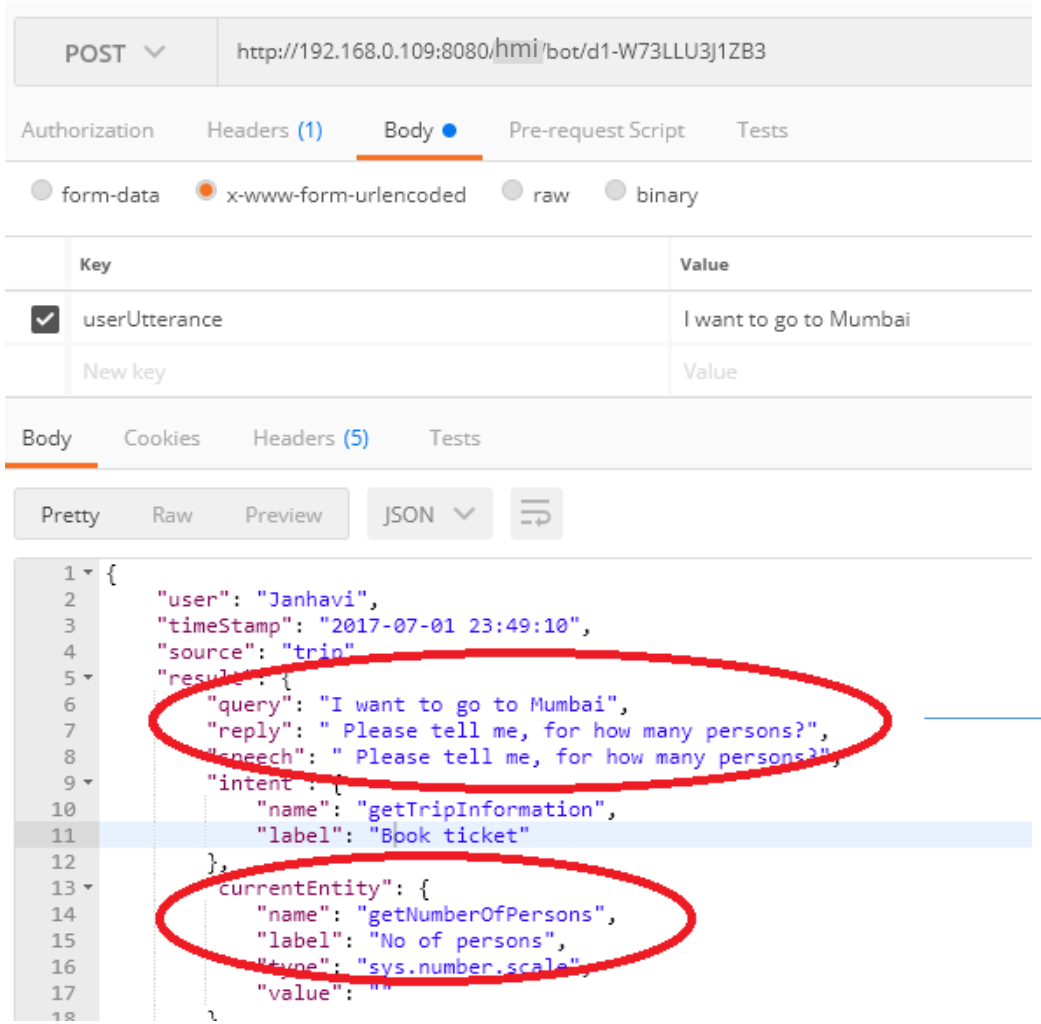
speech –> JSONObject is provisioned for Alexa like devices or message that needs to be sent to TTS engine

It also send status on entities and action

Meta-Information such as sessionId, user, source can be used to log the details.

BOT Platform – Calling BOT server APIs ..4/4

4. Continue your conversation -
You can continue your conversation by sending the user utterance in “userUtterance” parameter



Once this request is sent , Bot will process the userUtternace and will send JSON response with a reply

If it has executed any action it will populate the details (intent name and entities) in action JSON object.

BOT Platform – Dialog Definition file (DDF)

```
<?xml version="1.0" encoding="UTF-8"?>
<n:dialog xsi:schemaLocation="http://cto.net/hmi schema1.xsd" xmlns:n="http://cto.net/hmi/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="trip" company="xyz" version="1.0">
    <startTaskName>start</startTaskName>
    <globalLanguage>en</globalLanguage>
    <useSODA>true</useSODA>
    <allowSwitchTasks>true</allowSwitchTasks>
    <allowOverAnswering>false</allowOverAnswering>
    <allowDifferentQuestion>false</allowDifferentQuestion>
    <allowCorrection>false</allowCorrection>
    <useIntentEngine>true</useIntentEngine>
    <tasks>
        <task name="start" label="Initial Task"> ... </task>
        <task name="getTripInformation" label="Book ticket"> ... </task>
        <task name="getWeatherInformation" label="Weather information"> ... </task>
        <task name="getWikipediaCityInfo" label="City information"> ... </task>
        <!-- ... -->
        <task name="cancelTask" label="Cancel"> ... </task>
        <task name="handoverTask" label="Handover to agent"> ... </task>
    </tasks>
</n:dialog>
```

Domain or Dialog name , Company name , dialog version

Global Flags for defining Dialog behavior

Different Tasks that BOT needs to perform

"cancelTask" and "handoverTask" are reserved tasks used for coming out of current task and to handover the chat from bot to human.

BOT Platform – DDF - Global Flags

```
<startTaskName>start</startTaskName>
<globalLanguage>en</globalLanguage>
<useSODA>true</useSODA>
<allowSwitchTasks>true</allowSwitchTasks>
<allowOverAnswering>true</allowOverAnswering>
<allowDifferentQuestion>true</allowDifferentQuestion>
<allowCorrection>true</allowCorrection>
<useIntentEngine>true</useIntentEngine>
```

startTaskName – This specify the initial task to be executed by BOT. Initial task may be other than ”start” but after execution the intial task BOT will return to start task.

globalLanguage – This specifies the BOT language. This has to be in ISO language code. (e.g. English – en, Hindi – hi, Danish-da, Dutch-nl, Swedish-sv etc.)

useSODA (System Of Dialogue Act) – Each utterance will be classified using Max Entropy Classifier to find whether user is

=> seeking the information

=>Providing information or

=> issuing the defined command (e.g. switch ON, Switch OFF)

If set to true, dialog manager will check if the user is seeking the information if yes, will switch the task for its fulfillment.

```
resulted in: seek[0.0428]  action[0.0101]  prov[0.9471] -> prov
```

allowSwitchTasks – if set to true it will allow sub-dialogues from other tasks , if set to false only sub-dialogues of same task will be executed. (e.g. for trip.xml DDF use case - User can only answer questions specific to getTripInformation and will not allowed to switch to getWeatherInformation task)

allowOverAnswering - if set to true the user is allowed to provide more than the information that has been asked for (but at least the current question) (e.g. for trip.xml – user while in getTripInformation can provide information like “want to go to London for four people” which will fill ITOs for getDestinationCity and getNumberOfPersons)

allowDifferentQuestion - if set to true the user can ignore current question and answer a different unanswered question (e.g. for trip.xml – user while in getTripInformation task can say “I want to leave tomorrow” when Bot is actually asking about “where do you want to go?)

allowCorrection - if set to true user can change a value of an already asked question (e.g. for trip.xml – user while in getTripInformation can say “I want to go to Paris” when Bot is asking “for how many persons”. This will refill the getDestinationCity with new value “Paris”)

useIntentEngine – it set to true , the intent will be identified using internal intent engine trained separately (see appendix -3), else will use BOW specified in dialogue definition file.

BOT Platform – DDF – Task definition

```
<task name="getWeatherInformation" label="Weather information">
    <selector>
        <bagOfWordsTaskSelector>
            <word>weather|forecats|temprature</word>
        </bagOfWordsTaskSelector>
    </selector>
    <itos>
        <ito name="getDestinationCity" label="Weather information">
            <AQD>
                <type>
                    <answerType>sys.location.city</answerType>
                </type>
            </AQD>
            <fallbackQuestion>for which city do you want to know the weather?</fa
            <clarifyQuestion>Would you mind providing correct city name?</clarify
            <required>true</required>
            <useContext>true</useContext>
        </ito>
    </itos>
    <action>
        <httpAction>
            <returnAnswer>true</returnAnswer>
            <utteranceTemplate>The temperature in %getDestinationCity is #result
            <method>get</method>
            <params>q=%getDestinationCity&amp;mode=xml&amp;units=metric&amp;APPID
            <url>http://api.openweathermap.org/data/2.5/weather</url>
            <xpath>/current/temperature/@value</xpath>
            <jpath></jpath>
        </httpAction>
    </action>
</task>
```

Unique task name (intent)

This defines the intent through all the possible utterances. (Uses bag of words approach after removal of stop words. Refer /res/dictionary/ stopwords_en for details (you can use intent engine instead)

These are entities called ITO (Information transfer object) The slots are filled based on the AQD types (Abstract Question Descriptor)

This is a fall back question asked to user if slot is not filled and if “required” flag is true.
You can give multiple options by using “|”separator. You can also use ITOs by preceding the ITO name with %. e.g. for which city do you want to know the weather? | Hey, %loginUser what city would you want temperature? (currently for ENGLISH language) (%loginUser and %sessionID are system default)

(optional) This is clarify question which bot will use in case user fails to provide correct answer. (will use this over regular reply i.e. Sorry, I did not understand that. Please try again.for which city do you want to know the weather?)

(optional) If useContext flag is set to true , it will fill this ITO automatically , if it has been already answered in any of earlier conversation. (refer Appendix 6)

Global ITO

If ITO name ends with “_” then this is treated as global entity and will be passed to every task.
The system also provide loginUser_ and sessionId_ as default global ITOs and any task can use these by referring it as %loginUser_ and %sessionId_

BOT Platform – DDF (Dialogue Definition File) - Action

1. Supports REST Action

```
<action>
   <httpAction>
      <returnAnswer>true</returnAnswer>
      <utteranceTemplate>The temperature in $getDepartureCity is #result degrees.</utteranceTemplate>
      <method>get</method>
      <params>q=$getDepartureCity&amp;mode=xml&amp;units=metric&amp;APPID=706120eb0f11f2ebf8584dadb9bad3d6</params>
      <url>http://api.openweathermap.org/data/2.5/weather</url>
      <xpath>/current/temperature/@value</xpath>
      <jpath></jpath>
   </httpAction>
</action>
```

The api result will be always stored in result variable and accessed using #result

This will be used for Bot reply.

url for backend API

xpath for XML tag extraction

jpath in case the response is JSON (refer http://goessner.net/articles/JsonPath/ or https://github.com/jayway/JsonPath for more details

GET or POST method

url parameters with proper encoding (& for &)

BOT Platform – DDF–Groovy Action with Result Mapping

Supports Groovy Action and also result mapping

```
<action>
    <groovyAction>
        <resultMappings>
            <resultMapping>
                <message/>
                <redirectToTask>homeLoanEligibility</redirectToTask>
                <resultValue>1</resultValue>
                <resultVarName>action</resultVarName>
            </resultMapping>
            <resultMapping>
                <message/>
                <redirectToTask>homeLoanMenu</redirectToTask>
                <resultValue>2</resultValue>
                <resultVarName>action</resultVarName>
            </resultMapping>
        </resultMappings>
        <returnAnswer>true</returnAnswer>
        <utteranceTemplate/>
        <code>
            <![CDATA[
            Integer res = new Integer(2);
            String type=new String(frame.get("getOkInfo"));
            if (type.matches("(?i)^.*?\\b(ok|OK|Okay)\\b.*?"))
            res = 1;
            else if (type.matches("(?i)^.*?\\b(info|Info|INFO)\\b.*?"))
            res = 2;
            executionResults.put("action",res.toString());
            executionResults.put("getOKInfo","processing");

            ]]>
        </code>
    </groovyAction>
</action>
```

Task redirection based on the result of groovy action
Or could be http based action.
The #action value will decide the new task to be created.(API result is always stored in result variable)

NOTE- you can access the variable using # in utteranceTemplate tag. (e.g. #action)
Do not use result as variable as it is internally used in code.

One can use frame.get(“<ITO Name>”) method t oread the ITO value in Grrovy action.
You can modify the ITO value in groovy action by using executionResults.put (“<ITO Name>”,”<value>”)
Note- The changed value will be reflected in subsequent task if referred.

BOT Platform – DDF–Groovy Action ..sending predefined response

Use Groovy whenever you want to send the standard responses with no back end API call.

```
<task name="getCareCenter">
   <selector>
      <bagOfWordsTaskSelector>
         <word>closest care center</word>
      </bagOfWordsTaskSelector>
   </selector>
   <action>
      <groovyAction>
         <returnAnswer>true</returnAnswer>
         <utteranceTemplate>Anthony C Lopez - MD FACC is the closest among all which is at 173 North Morrison Ave, San
         Jose, CA 95126, Phone number (408) 293-1088.</utteranceTemplate>
         <code></code>
      </groovyAction>
   </action>
</task>
```

Pre defined answer with no back end call.

```
            <fallback_question>additional info that you want to provide ?</fallback_question>
            <required>true</required>
         </ito>
      </itos>
      <action>
         <groovyAction>
            <returnAnswer>true</returnAnswer>
            <utteranceTemplate>Hey %loginUser_ , this trip to %getDestinationCity costs #price Dollars.
            info %getInfo</utteranceTemplate>
            <code><![CDATA[executionResults.put("price","255")]]></code>
         </groovyAction>
      </action>
```

BOT Platform – DDF-Action with follow up question for confirmation

Supports Follow up question post executing the action

```
<action>
    <httpAction>
        <returnAnswer>true</returnAnswer>
        <utteranceTemplate>#result</utteranceTemplate>
        <method>get</method>
        <params>format=json&amp;action=query&amp;prop=extracts&amp;explaintext&amp;exsen
        <url>http://en.wikipedia.org/w/api.php</url>
        <xpath></xpath>
        <jpath>$..extract</jpath>
    </httpAction>
</action>
<followup>
    <ito name="anotherOne" label="Another City">
        <AQD>
            <type>
                <answerType>sys.decision</answerType>
            </type>
        </AQD>
        <fallbackQuestion>do you want to know about other cities?</fallbackQuestion>
        <required>true</required>
    </ito>
    <answerMapping>
        <item key="YES">getWikipediaCityInfo</item>
    </answerMapping>
</followup>
```

Follow-up question will be asked immediately after executing the task.

Use Uppercase. This KEY will be matched with ITO value. (sys.decision or custom.item_X

BOT Platform – In built ITOs into platform

Following are the ITOs already available in the platform

AVAILABLE

City => sys.location.city
Time =>sys.temporal.time => 1:00 AM, morning, evening etc.
Date => sys.temporal.date
Person => sys.person
Mail => sys.mail
Contact => sys.contact
First Name => sys.person.firstname
Last Name => sys.person.lastname
Dummy => dummy => captures all the text and passes it to ITO
Organization => sys.organization
QA Parser => sys.corpus.qa (for checking the answer in domain corpus)
Yes/No Parser => sys.decision => results in YES or NO
ON/OFF Parser =>sys.onoff => results in ON or OFF
Custom Buttons=>custom.button_X=>(x from 1 onwards based on corpus collection)
Custom List =>custom.item_X =>(x from 1 onwards based on the corpus collection)
Custom List =>custom.urlList_X =>(x from 1 onwards based on the corpus collection)
(in case of multiple selection items use below ones)
Custom List =>custom.multiItem_X=>=>(x from 1 onwards based on the corpus collection)
Custom List =>custom.multiUrlList_X=>=>(x from 1 onwards based on the corpus collection)
Custom Parser => custom.pattern_X (X from 1 onwards based on REGEX pattern)
Custom Classifier => custom.classifier_Y(Y is domain name) => This is ML based text classifier
Open Text => sys.opentext => capturing free text .. *(used in conjunction with interactive widgets , to be used in conjunction with other ITO. If used without interactive widget keep flag overanswering false)*

MM/DD/YYYY

01/01/2001 , 10-10-10 , 09/09/1998
01012001 (<MMDDYYYY>) , today , tomorrow,
yesterday ,Monday ..# valid
00-00-0000 , 15-15-2000 # invalid

This is used to fill the ITO in case the domain corpus has answer to the user question. This will use bot.properties file located at /res/config
If USE_QACHECK=True it will call QA_URL => which should return True or False based on if it has answer to the question
[Important –
Method => POST
url => QA_URL
Prameters => userUtterance="what is ATM"?
Response =>
{"response":"true"} or {"response":"false"}
]

If answer is "YES" this ITO will be filled and you can use <Action> to call the appropriate URL

BOT Platform – In built ITOs into platform

AVAILABLE

Generic Number => sys.number
Scaling Number =>sys.number.scale
Floating Number => sys.number.float

1.256	=> 1.256
123	=> 123
.234	=> **INVALID**
10 K	=> **INVALID**
1,239	=> **INVALID**
1 hundred	=> **INVALID**
Hundred thousand	=> **INVALID**

One two three	=> 123
Forty One Hundred	=> 4100
1,239	=> 1239
23545	=> 23545
1 hundred	=> **INVALID**
Hundred thousand	=> **INVALID**

1.2 L	=> 120000
1.2 Lac	=> 120000
10 K	=> 10000
1,239	=> 1239
23545	=> 23545
.2L	=> **INVALID**
1 hundred	=> **INVALID**
Hundred thousand	=> **INVALID**

BOT Platform –Creating Custom ITOs - Named Entities

(custom list , url based, and RegEX pattern)

There is often need to fill the slot that are business specific and are not available as part of standard NER. This can be achieved using custom named entities. Platform supports different ways of creating custom ITOs

=> If the single items need to be filled –

- custom_item_1 or custom_urlList_1 or custom_pattern_1

⇒ If multiple Items needs to be filled –

- custom_multiItem_1 or custom_multiUrlList_1

1. Custom List

```
<itos>
    <ito name="getType">
        <AQD>
            <type>
                <answerType>custom.item_4</answerType>
            </type>
        </AQD>
        <fallbackQuestion>whom do you want to contact?</fallbackQuestion>
        <required>true</required>
    </ito>
</itos>
```

item_4.txt file in
/res/entities folder

```
##DO NOT REMOVE THIS LINE - to check
all
broker
brokers
claim
claims
support
individual
```

If the file has name value pair (item=category) as shown, the ITO will be filled with category when item appears in utterance.

```
##DO NOT REMOVE THIS LINE - list of a
truck=auto
car=auto
honda=auto
ford=auto
chevrolet=auto
mercedes=auto
chrysler=auto
genral motors=auto
audi=auto
```

BOT Platform –Creating Custom ITOs - Named Entities

(url Based list entities)

2. Custom Url List – In this we get the list of items fetched from external API. Following steps are taken to support this method
   - Create urlList_X.txt in /res/entities folder. This file specify parameters specific to API
   - The API need to responsd with the list of items in csv format. e.g.

```
{"type":"userIDs",
"list":"johnd, jeevanK,mathew"
}
```

```
<itos>
    <ito name="selectUser" label="User Name">
        <AQD>
            <type>
                <answerType>custom.urlList_1</answerType>
            </type>
        </AQD>
        <fallbackQuestion>what is user name?</fallbackQuestion>
        <required>true</required>
    </ito>
</itos>
```

urlList_1.txt file in /res/entities folder

```
#The item list will be populated from below URL prope
URL_METHOD=GET
URL=http://www.mocky.io/v2/5945fed6130000e81b5b6fde
JPATH=$..list
CACHE_DAYS=0
```

JPATH=$..list → json object that contains list

CACHE_DAYS=0 → No of days for which data can cached. If cache is not required set it to “0”

BOT Platform –Creating Custom ITOs - Named Entities

(button selection entities)

3. Custom Button List – This is similar to list view but has been categorized as button list to show it as a button selection. (as against drop down list selection)

- Create buttont_X.txt in /res/entities folder.

```
<itos>
    <ito name="getClass" label="Class">
        <AQD>
            <type>
                <answerType>custom.button_1</answerType>
            </type>
        </AQD>
        <fallbackQuestion>what class would you like to travel?</fallbackQuestion>
        <required>true</required>
    </ito>
</itos>
```

button_1.txt file in /res/entities folder

```
1  ##DO NOT REMOVE THIS LINE - to check if it is new or list
2  Economy
3  First
4  Business
```

BOT Platform –Creating Custom ITOs - Named Entities
(REGEX based parser)

4. Custom Parser
You can provide list of REGEX pattern that will be used to extract entities from user utterance

```
<ito name="getTime">
    <AQD>
        <type>
            <answerType>custom.pattern_1</answerType>
        </type>
    </AQD>
    <fallbackQuestion>what date was it when you applied for your claim?</fallbackQuestion>
    <required>true</required>
</ito>
```

pattern_1.txt file in /res/entities folder

```
#DO NOT REMOVE THIS LINE - This pattern will find date in DD Month YYYY
\b[0-9]+\s[a-zA-Z0-9-]+\s[0-9-]{4}+\b
```

BOT Platform –Creating Custom ITOs- Multiple Item selection

(item & urlList based)

At times , we need the ITOs that need to be filled with multiple items.
e.g. what devices are owned by user – ITO can fill in one or more items from list (e.g. Mobile, Desktop, VOIP, Laptop etc.)
You can use custom.multipleItem_X or custom.multipleUrlList_X to extract entities from user utterance

```
<ito name="getMenu" label="Menu">
  <AQD>
     <type>
        <answerType>custom.multiItem_1</answerType>
     </type>
  </AQD>
  <fallbackQuestion>food items that you want to order?</fa
  <required>true</required>
</ito>
```

multiItem_1.txt file in /res/entities folder

```
##DO NOT REMOVE THIS LINE
Breakfast = BF
Lunch = LCH
Dinner = DNR
Desert = DST
```

```
<ito name="getDevices" label="Devices">
  <AQD>
     <type>
        <answerType>custom.multiUrlList_1</answerType>
     </type>
  </AQD>
  <fallbackQuestion>the devices registered against your ID?<
  <required>true</required>
</ito>
```

multiUrlItem_1.txt file in /res/entities folder

```
#The item list will be populated from below URL properties
URL_METHOD=GET
URL=http://www.mocky.io/v2/5945fed6130000e81b5b6fde
JPATH=$..list
CACHE_DAYS=2
```

JSON response of this API -
{ "type": ”deviceTypes", "list": ”mobile,desktop,pc,laptop,voip" }

BOT Platform –Creating Custom ITOs

– ML based text classifier (only for ENGLISH)

At times , we need ITO capable of classifying user text to one of the classes.
e.g. if one has to classify nature of ticket when user is asked to describe the problem. As an example “I am facing logging into mail outlook” can be classified as “login” class.
Follow below steps to create ML based classifier ITO.

1. Define your classes in json format as shown.
2. Save your file with proper naming convention in /res/entities/classifier/data/classifier_<domain name>.json (e.g. classifier_ticketType.json)
3. Set your threshold score that would qualify the class in /res/entities/classifier/config/classifier.properties
4. Now you can use this classifier in your DDF file as shown. Whenever the user utterance matches with any of the class (with confidence score >= threshold score) the ITO will get filled.
5. Please note the ITO value will contain “Class:<name> Utterance:<user utterance>” e.g. “Class:**software** Utterance:Assignement of ticket to user not working”.
6. Important: please delete all the models whenever you update the classes in classifier_<domain name>.json file.

Keep “allowDifferentQuestion” and “allowCorrection” flag to false to keep minimal collision with other ITOs

```
#stores configuration
THRESHOLD_SCORE=0.6
```

```
{
    "domain": "ticketType",
    "classes": [{
        "name": "login",
        "utterances": ["I am not able to login into system",
        "user and password is not working on outlook",
        "authentication failed for outlook",
        "can not login into the system",
        "user id and password are not accepted",
        "login issue"]
    },
    {
        "name": "software",
        "utterances": ["Not able to generate report in remedy",
        "The service ticket is not appearing",
        "Ticket assignement is not working",
        "fail to export the file to local folder"]
    },
    {
        "name": "hardware",
        "utterances": ["Printer is not working",
        "Paper jam in printer",
        "keyboard is malfunctioning",
        "Monitor is flickering",
        "Desktop is powering up"]
    }
    ]
}
```

```
<useSODA>true</useSODA>
<allowSwitchTasks>true</allowSwitchTasks>
<allowOverAnswering>true</allowOverAnswering>
<allowDifferentQuestion>false</allowDifferentQuestion>
<allowCorrection>false</allowCorrection>
<useIntentEngine>true</useIntentEngine>
```

```
<ito name="getInfo" label="Additional Info">
   <AQD>
      <type>
         <answerType>custom.classifier_ticketType</answerType>
      </type>
   </AQD>
   <fallback_question>describe the nature of problem?</fallback_question>
   <required>true</required>
</ito>
```

BOT Platform – Training, activity and Dialog Logs

1- The failed conversation with BOT are logged into a log file <mmm>_<dd>_training.log (mmm is current month and dd is current date) located in folder /res/logs/training/<MMM>

```
Sat May 13 23:10:46 IST 2017 domain:trip user:John task:start utterance:How do I book movie tickets
Sat May 13 23:35:26 IST 2017 domain:trip user:John task:start utterance:Can I cancel my reservation?
```

This log will help in training the BOT - for intents that seems to be correct but not defined in DDF file.

2- The logs are captured whenever action is executed in a file <mmm>_<dd>_activity.log (mmm is current month and dd is current date) located in folder /res/logs/activity/<MMM>

```
Sat May 13 23:37:32 IST 2017 domain:trip user:John task:getFAQ TYPE:GroovyAction@4f5de9f8
Sat May 13 23:38:09 IST 2017 domain:trip user:John task:getTripInformation TYPE:GroovyAction@647e46e1
Sat May 13 23:39:42 IST 2017 domain:trip user:John task:getWeatherInformation TYPE:HTTPAction@2bf4b0ef
```

3- The dialog conversations are logged if LOG_DIALOG flag in bot.properties is set to true (LOG_DIALOG=true). They are stores in a file <mmm>_<dd>_dialog.log (mmm is current month and dd is current date) located in folder /res/logs/dialog/<MMM>

```
Sun May 14 16:34:39 IST 2017 domain:trip client_ip:192.168.0.109 user:John id:d1-5SABSB4MOVIF dialog:"S":"How may I help you?","U":"Hi","S":"Hello t
Sun May 14 16:37:52 IST 2017 domain:trip client_ip:192.168.0.109 user:John id:d2-VDQMVNCDSFYB dialog:"S":"How may I help you?","U":"How is weather",
```

Due to GDPR compliance requirement all the important data provided by user is masked (replaced with ??) while storing in LOG file.

BOT Platform – BOT configuration

Bot configuration file is located in /res/config folder and stores some global properties parameter.

```
PROTOCOL=HTTPS
KSPASSWORD=naturaldialog
USE_NLG=true
USE_GREETINGS=true
USE_DOMAIN=false
IGNORE_PREV_TASK = true
DOMAIN_URL_METHOD=get
DOMAIN_URL=http://localhost:5000/faq
USE_QACHECK=false
QA_URL=http://mydomainqa.com/v2
IE_THRESHOLD_SCORE=0.45
IE_SIMILARITY_INDEX=0.5
LOG_DIALOG=true
SHOW_INTERACTIVE_CARDS=true
ALLOWED_FAILURE_ATTEMPTS=3
```

If set to HTTPS the Jetty server will run as secure HTTPS else HTTP (default)

For HTTPS this will be used as keystore password

If set true will fill natural language responses . Tell me now, OK, etc.

If set true, will answer greeting related generic questions . Good Morning etc.

This is for answering domain questions that are build into main conversation itself.

If TRUE (also allowSwitchTask -> TRUE) the prev task will not be retained in context.

URL method for querying domain Qs. (e.g. FAQs that are part of user conversations)

URL => should respond to Qs directly as text (e.g. ATM is cash withdrawal machine)

This is used in conjunction with “sys.corpus.qa” ITO.
If USE_QACHECK=True it will call QA_URL => which should return True or False based on if it has answer to the question
[Important –
Method => POST
url => QA_URL
Prameters => userUtterance=“what is ATM”?
Response =>
{“response”:”true”} or {“response”:”false”}
]

If answer is “YES” this ITO will be filled and you can use <Action> to call the appropriate URL

As an example –
The GET method will be triggered to url http://localhost:5010/faq?userUtterance=“What is ATM”

Response will be -
{“response” : ”ATM is cash with drawl Machine”}
or
{“response” : “NA”}

If true , it will log dialog logs to /res/log/dialog folder

After these many failed attempts Bot will invoke handoverTask to transfer chat from bot to human.

This is used by Intent Engine – If it finds more than one intent with score exceeding threshold level, this parameter will be used to qualify the second intent.(if score difference is < similarity index then this will be set to INTENT_CLARIFICATION where user will be prompted to confirm

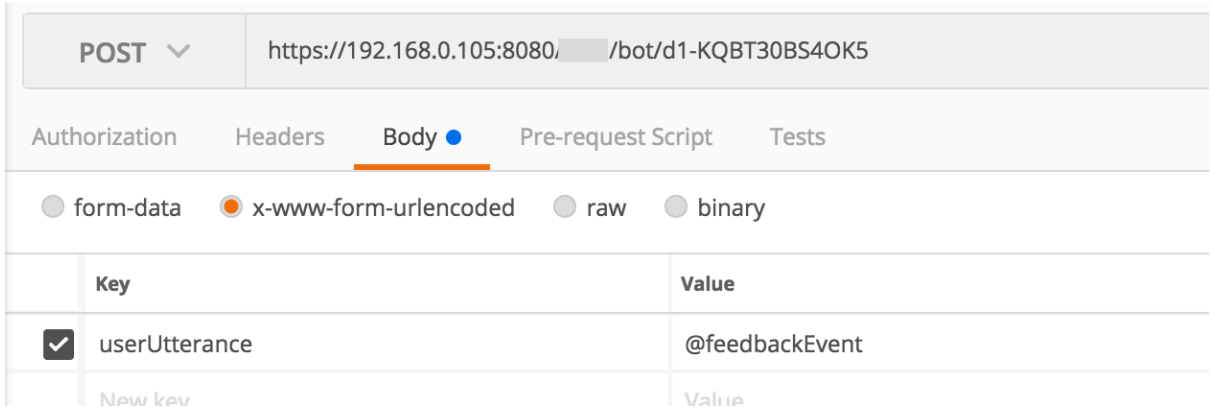
This is used by Intent Engine – this is threshold confidence score , if exceeded intent will be considered as INTENT_FOUND

This will create the interactive card response (see appendix 5 for more details)

BOT Platform – Triggering Event Dialogs …(1/4)

Platform provides mechanism to trigger dialog events that are outside main DDF. The event specific dialogs are not part of main DDF and are written separately in "/res/events" folder. For ease of use , it is written in yaml format with file extension .yml.

- All DDF features are available as listed below
    - use of custom itos
    - support of groovy and http action
    - resultMapping feature to trigger task based on action (http or groovy) result
    - followup task
- The events are triggered by sending "userUtterance" parameter as "@<eventName>" in POST request
  For example "@feedbackEvent" will look for event dialog file "feedbackEvent.yml" file in /res/events folder



- The event dialog contains set of tasks that the BOT engine will execute as if they are part of main dialog. The tasks of event dialog once executed will be removed automatically as they are not part of main dialog defined in DDF.
- All global parameters that are set in main DDF dialog files will automatically apply for event triggered dialogues.
- The other advantage of using yaml , there is no need to restart BOT engine whenever you add new event triggered dialogue in event folder.

BOT Platform – Triggering Event Dialogs …(2/4)

Following are few samples of event triggered dialog file -

Single Task with groovy Action – testSample1.yml

```
# Details of a event
---
tasks :
    - task :
        name : EVT_getFeedback
        label : Feedback
        itos :
            - ito :
                name : getConfirmation
                label : 'Confirmation'
                required : true
                answerType: sys.decision
                fallbackQuestion: 'if BOT was able to answer to your query?'
            - ito :
                name : getRating
                label : Rating
                required : true
                answerType: sys.number
                fallbackQuestion: 'how would you rate this on scale of 0-5?'
        action :
            type : groovyAction
            returnAnswer : true
            utteranceTemplate : '#msg Please visit again.'
            code : 'executionResults.put("msg","Thank you for your feedback.")'
...
```

Please note that all the task name must start with “EVT_”

Use variable other than “result”

Single Task with groovy Action with followup – testSample2.yml

```
# Details of a event
---
tasks :
    - task :
        name : EVT_getFeedback
        label : Feedback
        itos :
            - ito :
                name : getConfirmation
                label : 'Confirmation'
                required : true
                answerType: sys.decision
                fallbackQuestion: 'if BOT was able to answer to your query?'
            - ito :
                name : getRating
                label : 'Rating'
                required : true
                answerType: sys.number
                fallbackQuestion: 'how would you rate this on scale of 0-5?'
        action :
            type : groovyAction
            returnAnswer : true
            utteranceTemplate : '#msg'
            code : 'executionResults.put("msg","Thank you for your feedback.")'
        followup :
            ito :
                name : anotherOne
                label : 'More feedback'
                required : true
                answerType : sys.decision
                fallbackQuestion: 'do you want to give another feedback'
            answerMapping:
                - map :
                    # use quotes for key
                    key : 'YES'
                    value : EVT_getFeedback
                - map :
                    # use quotes for key
                    key : 'NO'
                    value : start
...
```

BOT Platform – Triggering Event Dialogs …(3/4)

Following are few samples of event triggered dialog file -

Single Task with resultMapping Action – testSample3.yml

```
# Details of a event
---
tasks :
    - task :
        name : EVT_getFeedback
        label : Feedback
        itos :
            - ito :
                name : getConfirmation
                label : 'Confirmation'
                required : true
                answerType: sys.decision
                fallbackQuestion: 'if BOT was able to ans
            - ito :
                name : getRating
                label : 'Rating'
                required : true
                answerType: sys.number
                fallbackQuestion: 'how would you rate thi
        action :
            type : groovyAction
            resultMappings :
                - map :
                    message : 'Thank you for your feedbac
                    redirectToTask : start
                    resultVarName : action
                    # use quotes for key
                    resultValue : '1'
            returnAnswer : true
            utteranceTemplate : null
            code : 'executionResults.put("action","1")'
...
```

Single Task with httpAction – testSample4.yml

```
# Details of a event
---
tasks :
    - task :
            name : EVT_getTemprature
            label : Temprature
            itos :
                - ito :
                    name : getCityName
                    label : 'City Name'
                    required : true
                    answerType: sys.location.city
                    fallbackQuestion: 'for which city do you want to know the weather?'
            action :
                type : httpAction
                returnAnswer :  true
                utteranceTemplate : 'The temperature in %getCityName is #result degrees
                method : GET
                params : q=%getCityName&mode=xml&units=metric&APPID=706120eb0f11f2ebf858
                url : http://api.openweathermap.org/data/2.5/weather
                xpath : /current/temperature/@value
                jpath : null
...
```

BOT Platform – Triggering Event Dialogs …(4/4)

Following are few samples of event triggered dialog file -

Multiple Task with resultMapping Action – testSample5.yml

```
# Details of a event
---
tasks :
    - task :
            name : EVT_getBooking
            label : 'Ticket Booking'
            itos :
                - ito :
                    name : getDepartureCity
                    label : 'Departure city'
                    required : true
                    answerType: sys.location.city
                    fallbackQuestion: 'where do you want to go?'
                - ito :
                    name : getDate
                    label : 'Departure Date'
                    required : true
                    answerType: sys.temporal.date
                    fallbackQuestion: 'when do you want to leave?'
                - ito :
                    name : getInfo
                    label : 'Addition Info'
                    required : true
                    answerType: sys.opentext
                    fallbackQuestion: 'any additional information that you may have?'
            action :
                type : groovyAction
                resultMappings :
                    - map :
                        message : 'Your ticket for %getDepartureCity city is booked. Th
                        redirectToTask : EVT_getTemprature
                        resultVarName : action
                        # use quotes for key
                        resultValue : '1'
                returnAnswer : true
                utteranceTemplate : null
                code : 'executionResults.put("action","1")'
    - task :
            name : EVT_getTemprature
            label : 'City Temprature'
            itos :
                - ito :
                    name : getCityName
                    label : 'City Name'
                    required : true
                    answerType: sys.location.city
                    fallbackQuestion: 'For which city do you want to know the weather?
            action :
                type : httpAction
                returnAnswer :  true
                utteranceTemplate : 'The temperature in %getCityName is #result degree
                method : GET
                params : q=%getCityName&mode=xml&units=metric&APPID=706120eb0f11f2ebf8
                url : http://api.openweathermap.org/data/2.5/weather
                xpath : /current/temperature/@value
                jpath : null
```

BOT Platform API for messenger integration

Bot can be used to integrate with Messaging platform like Facebook or slack by calling appropriate API's listed below.

1. To create the BOT instance for specific user ID (e.g. if a unique user ID is 1000345667)
    Method => GET
    URL => http://192.168.3.19:8080/hmi/msgbot?userID= 1000345667
    Return =>
        if successful creates the BOT instance
        if unsuccessful returns
            - if no userID provided returns {\"response\":\"Error: need user ID for BOT creation\"}
            - initialization failed if instance is already running
2. To send the user message to instance that has been created in STEP-1
    Method => GET
    URL => http://192.168.3.19:8080/hmi/msgbot/1000345667?userMessage="Hi"
        -> use URLEncoder.*encode(userUtterance, "UTF-8") for sending the user Utterance*
    Return => JSON response with response in result.reply JSONObject

3. To check if instance is already available
    Method => GET
    URL =>http://192.168.3.19:8080/hmi/msgbot/hasInstance?userID=1000345667
    Returns=> {"response":"false"} or {"response":"true"} based on whether such instance is available.

BOT Platform – Other tools

1. Always have “cancelTask” at the bottom of DDF. This will be triggered when user says “I want to cancel the task”. It will have follow-up confirmation and if user says “YES” , it will come out of current task.
   [Important – If the user says this while in “start” task all the global parameters will be set to NULL]

2. HTML client is available in /html folder
   http://X.X.X.X:8080/ for running it from browser
   http://X.X.X.X:8080/hmi/bot for creating bot instance from REST client
   http://X.X.X.X:8080/hmi/authenticateUser for user authentication
3. For running local MongoDB
   D:\Programs\MongoDb\bin>mongod --dbpath D:\Programs\MongoDb\data
4. For running local tomcat
   D:\Programs\Tomcat-7\bin\startup.bat
5. In order to get the proper Bag Of words that needs to go in DDF follow below steps –
   - create file intent.dat in /res/intents/data folder that contains “|” separated task and intent
   - go to hmi.jar and run below command
   - *>java -cp hmi.jar cto.hmi.bot.util.CreateIntent -f /res/intents/data/intent.dat*
   - It will process file and will generate the output file intent_out.dat with intents that can be fed into DDF.

intent.dat

```
1  createPoilcy|I want to create a policy document
2  listPolicy|Show me my policy document
```

intent_out.dat

```
1  createPoilcy|I want to create a policy document|create policy document
2  listPolicy|Show me my policy document|show policy document
```

Appendix -1 Managing interactive response through HTTP action

One can build the interactive chat by building the API that can return following response.
e.g. To show a issued insurance card in a image window , if you build a API that can return a JSON response like this -
Note – Single quotes that are used to specify the image source. (the use of double quote will make the JSON an invalid JSON)

```
{
  "message": {
    "data": {
      "error": "",
      "info": {
        "image": "<div><img src='http://localhost:8090/lima/images/DupInsuranceCard.jpg'></div>",
        "video": "",
        "audio": ""
      }
    },
    "chat": "Please note down the policy number and group number from the duplicate insurance card in image section."
  }
}
```

Here is task to read the same

- specify japth as JSON object $..message
- **use only** #result in utteranceTemplate – this will put chat object here if present. (this is required irrespective of you have chat object or not)

Here is the BOT response that will have appended JSON object at the bottom

```
,
action": {
 "name": "getWeatherInform ion",
 "entities": {
   "get pa ity": "Pune"
 }
,
message": {
 "data": {
   "error": "",
   "info": {
     "image": "<div><img src='http://localhost:8090/lima/images/DupInsura
     "video": "",
     "audio": ""
   }
 },
 "chat": "Please note down the policy number and group number from the du
```

```
        <fallback_question>for which card you need duplicate?</fallback_question>
        <required>true</required>
        <useContext>true</useContext>
    </ito>
</itos>
<action>
    <httpAction>
        <returnAnswer>true</returnAnswer>
        <utteranceTemplate>#result</utteranceTemplate>
        <method>get</method>
        <params>id=%getCardType</params>
        <url>http://10.44.22.76:8090/lima/serlayer/getDuplicateInsuranceCard</url>
        <xpath></xpath>
        <jpath>$..message</jpath>
    </httpAction>
</action>
</task>
```

Note this is a JSON object and not key

Appendix -1 Managing interactive response through Groovy action

One can build similar experience using Groovy action (refer earlier slide for understanding the interactive action using HTTP)
The BOT JSON response will append the JSON body that you can use on client for creating interactive card.
Build API that has below response. (Needs to have chat object for returning the BOT reply)

```
"message": {
    "data": {
        "error": "",
        "info": {
            "image": "<div><img src='img/ticket.jpg' alt='ticket.jpeg'> </div>",
            "video": "<div class='video-container'><iframe src='https://www.you
            class='video'></iframe></div>",
            "audio": "",
            "text": "Your ticket is booked. Please see the details in attached
        }
    },
    "chat": "<p>Your ticket is booked successfully for 600 Dollars. Wish you a
    href='https://www.irctc.co.in/eticketing/loginHome.jsf'>Click here for logi
}
```

Here is the BOT response that will have appended JSON object at the bottom

```
                "id": "4",
                "name": "cancelTask",
                "label": "Cancel"
            }
        ]
    },
    "message": {
        "data": {
            "error": "",
            "info": {
                "image": "<div><img src='img/ticket.jp
                "video": "<div class='video-container'
                    ='0' allowfullscreen class='video'
                "audio": "",
                "text": "Your ticket is booked. Please
            }
        },
        "chat": "<p>Your ticket is booked successfully
            .irctc.co.in/eticketing/loginHome.jsf'>Cli
```

- Populate the JSON response in “body” variable
- **use only** #result in utteranceTemplate this will put chat object here if present.
- You can as well populate a variable for redirection , if required. (action = 1 in this case)

```
    <groovyAction>
        <resultMappings>
            <resultMapping>
                <message></message>
                <redirectToTask>start</redirectToTask>
                <resultValue>1</resultValue>
                <resultVarName>action</resultVarName>
            </resultMapping>
        </resultMappings>
        <returnAnswer>true</returnAnswer>
        <utteranceTemplate>#result</utteranceTemplate>
        <code>
        <![CDATA[
            String type=new String(frame.get("getImageNumber"));
            Integer action = new Integer(1);
            import org.codehaus.jettison.json.JSONException
            import org.codehaus.jettison.json.JSONObject
            def url = new URL('http://www.mocky.io/v2/58dfb92c1000007d03cc15ca');
            def connection = url.openConnection();
            connection.requestMethod = 'GET';
            if (connection.responseCode == 200) {
            body = connection.content.text;
            executionResults.put("body",body);
            }
            executionResults.put("action",action.toString());
        ]]>
        </code>
    </groovyAction>
</action>
```

Appendix -2 -Managing interactive response through Groovy

One can manage the API integration completely through Groovy.

```
    <resultMapping>
        <message></message>
        <redirectToTask>getWeatherInformation</redirectToTask>
        <resultValue>1</resultValue>
        <resultVarName>action</resultVarName>
    </resultMapping>
</resultMappings>
<returnAnswer>true</returnAnswer>
<utteranceTemplate>#chat</utteranceTemplate>
<code>
    <![CDATA[
    import org.codehaus.jettison.json.JSONException;
    import org.codehaus.jettison.json.JSONObject;
    import java.util.logging.Logger;
    import java.net.URLEncoder;
    Logger logger = Logger.getLogger("");
    String body = new String ("");
    String chat = new String ("");
    def url =new URL('http://10.44.22.76:8090/lima/serlayer/getDuplicateInsuranceCard');
    def connection = url.openConnection();
    if (connection.responseCode == 200) {
    body = connection.content.text;
    JSONObject json = new JSONObject(body);
    if (json.getJSONObject("message").has("chat"))
    chat = json.getJSONObject("message").getString("chat");
    logger.info ("chat output ======"+chat);
    }
    executionResults.put("chat",chat);
    executionResults.put("body",body);
    //for result mapping
    executionResults.put("action","1");
    ]]>
</code>
</groovyAction>
ction>
```

1. Please populate body as a total JSON response. It will look for message (see JSON response on Appendix-1 slide) and if present will process it.
2. Parse which ever object you want in your utterance (#chat in this case)

Option-1 – if you use #chat then it will only give utterance and will not append it in BOT processed response. But you can customize your response.

Option-2 – If you use ONLY #result then it will be automatically processed for chat object for response and additionally append JSON response at the end of BOT processed response.

You can manage #action in case you need to redirect the task based on the result in conjunction with resultMapping

Appendix -3 – Using Intent Engine for training the BOT .. 1/3

BOT provides two ways of defining the Intent or task.

1 – Bag Of Words Approach => In this approach a collection of words if appear in mentioned order , the task gets identified.
As an example –
This will identify “appointment” task for all of the following utterances
Schedule meeting with john
Schedule with John a meeting -> ‘with’ and ‘a’ are stop words and John will be filled in for ‘*’
Fix an appointment with John

```
<task name="appointment">
    <selector>
        <bagOfWordsTaskSelector>
            <word>schedule meeting</word>
            <word>schedule * meeting</word>
            <word>appointment</word>
        </bagOfWordsTaskSelector>
    </selector>
    <itos>
```

Note there is utility to remove the stop words and get the BOW that you can use for configuring the “selector” in DDF.

In order to get the proper Bag Of words that needs to go in DDF follow below steps –

- create file intent.dat in /res/intents/data folder that contains “|” separated task and intent
- go to hmi.jar and run below command
- *>java -cp hmi.jar cto.hmi.bot.util.CreateIntent -f /res/intents/data/intent.dat*
- It will process file and will generate the output file intent_out.dat with intents that can be fed into DDF.

intent.dat

```
1  createPoilcy|I want to create a policy document
2  listPolicy|Show me my policy document
```

intent_out.dat

```
1  createPoilcy|I want to create a policy document|create policy document
2  listPolicy|Show me my policy document|show policy document
```

Appendix -3 – Using Intent Engine for training the BOT .. 2/3

BOT provides two ways of defining the Intent or task.

"en" for ENGLISH
"hi" for HINDI
"mr" for MARATHI

2 – ML based Intent Engine Approach => In this approach we train a BOT on all the intents and their possible utterances. In order to train BOT follow below steps.

- create <domain>_<langugae>.json file (e.g. insurance_en.json) with following JSON structure
- you can add domain specific synonyms to synonyms_en.txt located in res/dictionary folder

```
auto,vehicle,truck
mail,e-mail
text,sms,message
```

```
{
    "domain": "insurance",
    "tasks": [{
        "name": "listCertificate",
        "utterances": ["Certificates",
        "I want to get a certificate of insurance",
        "can you give me a certificate of liability insu
        "I want to see my new certificate",
        "show my certificate",
        "show me the list of certificates"]
    },
    {
        "name": "createNewCertificate",
```

Domain name - Insurance

task name – "listCertificate"

All possible utterances

```
xmlns:n="http://cto.net/hmi/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
name="insurance" company="xyz" version="1.0">
<start_task_name>start</start_task_name>
<global_language>en</global_language>
<useSODA>true</useSODA>
<allowSwitchTasks>true</allowSwitchTasks>
<allowOverAnswering>true</allowOverAnswering>
<allowDifferentQuestion>true</allowDifferentQuestion>
<allowCorrection>true</allowCorrection>
<useIntentEngine>true</useIntentEngine>
<tasks>
  <task name="start" label="Initial Task">
```

- Copy this file into /res/intents/data folder
- Ensure that domain name matches with the one in DDF
- Set the 'useIntentEngine' flag to true => <useItentEngine>true</useIntentEngine>
- Ensure you use intent/task name in your DDF same as that defined in <domain>.json file.
- Keep the <selector> tag empty as intent Engine will identify the task.

```
<task name="listCertificate">
  <selector></selector>
  <itos></itos>
  <action>
    <groovyAction>
      <returnAnswer>true</returnAnswer>
      <utteranceTemplate>Here is list of you
      <code></code>
    </groovyAction>
  </action>
</task>
```

The ML based model will auto train the BOT on intents and its utterances.
In case you add any new utterance to JSON file, you need to restart the BOT instance.

Appendix -3 – Using Intent Engine for training the BOT .. 3/3

2 – ML based Intent Engine Approach (Continued…) =>

This approach uses two parameters specified in bot.properties file.

```
PROTOCOL=HTTPS
KSPASSWORD=naturaldialog
USE_NLG=false
USE_GREETINGS=true
USE_DOMAIN=false
DOMAIN_URL_METHOD=get
DOMAIN_URL=http://localhost:5000/faq
USE_QACHECK=false
QA_URL=http://www.mocky.io/v2/583c63e7290000970b
IE_THRESHOLD_SCORE=0.45
IE_SIMILARITY_INDEX=0.5
LOG_DIALOG=true
SHOW_INTERACTIVE_CARDS=true
ALLOWED_FAILURE_ATTEMPTS=3
```

This is used by Intent Engine – If it finds more than one intent with score exceeding threshold level, this parameter will be used to qualify the second intent.(if score difference is < similarity index then this will be set to INTENT_CLARIFICATION where user will be prompted to confirm

This is used by Intent Engine – this is threshold confidence score , if exceeded intent will be considered as INTENT_FOUND

These parameters should be fine tuned such that BOT is able to identify the intents properly. In case the intents are too similar it will prompt user for clarification.

Important –
There is provision to call the task directly using #<taskname>

e.g. if userUtterance is #getTripInformation and if it is present in DDF , the BOT engine will directly call this task bypassing the intent engine.

Appendix -4 – Managing both Transactional and FAQ conv. using intent engine

In order to manage both Transactional and FAQ conversation in one dialogue one can follow below approach

- Use Intent Engine to handle all your transactional scenarios. (No need to add FAQ specific utterances)
- Create “getFAQ” task to handle all FAQ specific questions ..(Important – task name has to be “getFAQ” and “overanswering” flag to be set to true)
- Set USE_DOMAIN parameter in bot.properties to “false” as we do not want 2 sources for finding FAQ answers
-In DDF add getFAQ task as shown here (use groovy for better control on chat response) note – use answerType as “dummy”

```
#Sat Oct 29 15:04:24 IST 2016
USE_NLG=true
USE_GREETINGS=true
USE_DOMAIN=false
DOMAIN_URL_METHOD=get
DOMAIN_URL=http://localhost:5000/faq
```

```
<task name="getFAQ" label="Find FAQ">
    <selector></selector>
    <itos>
        <ito name="getQuestion">
            <AQD>
                <type>
                    <answerType>dummy</answerType>
                </type>
            </AQD>
            <fallback_question>OK</fallback_question>
            <required>true</required>
        </ito>
    </itos>
    <action>
        <groovyAction>
            <returnAnswer>true</returnAnswer>
            <utteranceTemplate>#chat</utteranceTemplate>
            <code>
            <![CDATA[
            import org.codehaus.jettison.json.JSONException
            import org.codehaus.jettison.json.JSONObject
            String question=new String(frame.get("getQuestion"));
            String body = new String ("");
            String chat = new String ("");
            String api = "http://www.mocky.io/v2/590ae3b82900004b0523d934";
            String urlParam = "?userUtterance="+ URLEncoder.encode(question, "UTF-8");
            def url = new URL(api+urlParam);
            def connection = url.openConnection();
            connection.requestMethod = 'GET';
            if (connection.responseCode == 200) {
            body = connection.content.text;
            JSONObject json = new JSONObject(body);
            if (json.has("response"))
            chat = json.getString("response");
            if (chat.equals("NA"))
            chat="Sorry, I did not understand that.";
            }
            executionResults.put("body",body);
            executionResults.put("chat",chat);
            ]]>
            </code>
        </groovyAction>
    </action>
</task>
```

Appendix -5 – Building Interactive Widgets 1/3

The JSON provides 2 types of iCard object (if enabled in bot.properties file).

1- The iCard object with type “taskList” for user to select the task with a click of button. (use #getTripInformation in userUtterance urlencoded body parameter to call getTripInformation task)

```
"iCard": {
    "type": "taskList",
    "tasks": [
      {
        "id": "1",
        "name": "getTripInformation",
        "label": "Book ticket"
      },
      {
        "id": "2",
        "name": "getWeatherInformation",
        "label": "Weather information"
      },
      {
        "id": "3",
        "name": "getWikipediaCityInfo",
        "label": "City information"
      },
      {
        "id": "4",
        "name": "cancelTask",
        "label": "Cancel"
      }
    ]
  }
```

2- The iCard object with type “entityList” for user to enter various entity data and send

```
"iCard": {
        "type": "entityList",
        "entities": [
            {
                "id": "1",
                "name": "getDestinationCity",
                "label": "City Name",
                "entityType": "sys.location.city",
                "type": "text",
                "elements": "",
                "value": "",
                "isVisible": "true",
                "isActive": "true"
            },
            {
                "id": "2",
                "name": "getNumberOfPersons",
                "label": "No of persons",
                "entityType": "sys.number.scale",
                "type": "number",
                "elements": "",
                "value": "",
                "isVisible": "true",
                "isActive": "false"
            },
            {
                "id": "3",
                "name": "getStartDate",
                "label": "Start Date",
                "entityType": "sys.temporal.date",
                "type": "date",
                "elements": "",
                "value": "",
                "isVisible": "true",
```

Type of UI
e.g. text, Number,button, list, multiSelectionList, radio, date, time etc.

For UI type list, multiItemlist and radio this will show the elements.

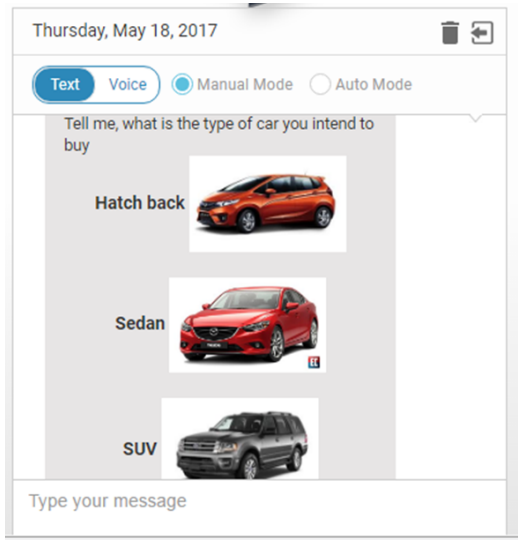
Value of ITO if filled

Current Active entity element

Show the element only if this flag is set to “true”. This is done to handle special ITOs like “opentext”

Appendix -5 – Interactive chat by embedding HTML tags…….2/3

One can use [CDATA] wrapper in utterance tags like <fallback_question> , <utteranceTemplate> or <message> to build more structured response including images.




```
<task name="bookCar" label="Book Car">
        <selector></selector>
        <itos>
           <ito name="getCarType" label="Car Type">
              <AQD>
                 <type>
                    <answerType>custom.item_1</answerType>
                 </type>
              </AQD>
              <fallbackQuestion><![CDATA[<p>what is the type of car you intend to buy?</p>
              <br><center><b>Hatch back</b><img src='img/hatchback.jpg' height='50%' width='%50'></center></br>
              <br><center><b>Sedan</b><img src='img/sedan.jpg' height='50%' width='%50'></center></br>
              <br><center><b>SUV</b><img src='img/suv.jpg' height='50%' width='%50'></center></br>]]></fallbackQuestion>
              <required>true</required>
           </ito>
```

This is custom parser => HatchBack, Sedan, SUV

The images can be kept at /res/html/img folder

Important – The key message that BOT needs to speak through (TTS) needs to be embedded in <p> tag.
The message is automatically parsed and sent to “speech” JSONObject in its REST response

Appendix -5 – Processing utterance filled through i-forms ……..3/3

If interactive cards are created for user to enter the data , in order to process ITOs seamlessly following provision has been made that require special treatment -

City	=> sys.location.city	=> city:<data>;
Person	=> sys.person	=> person:<data>;
First Name	=> sys.person.firstname	=> firstName:<data>;
Last Name	=> sys.person.lastname	=> lastName:<data>;
Organization	=> sys.organization	=> organization:<data>;

e.g. For trip booking the filled in data by user in form for ITO's (getDestinationCity, getNumberOfPerson, getStartDate, getEndDate) could be passed like

city:Mumbai;<space>4;<space>7/3/17;<space>7/14/17

Appendix -6 – Use of context to fill ITOs automatically

When ITO is being defined in DDF there are two flags that can be effectively used to fill them based on the context. The ITO will be filled in automatically if it uses the same name as defined earlier in other task and <useContext> flag is set.

There is another xml TAG that can be used to clear the context in case you are sure that the intended use of ITO is done and now needs to be cleared from history. For example

TASK-1

```
<task name="getTripInformation" label="Book ticket">
    <selector></selector>
    <itos>
        <ito name="getDestinationCity" label="City Name">
            <AQD>
                <type>
                    <answerType>sys.location.city</answerType
                </type>
            </AQD>
            <fallback_question>where do you want to go?|what
            <required>true</required>
```

In this task , we defined ITO “getDestinationCity” which takes city name

TASK-2

```
<task name="getWeatherInformation" label="Weather information">
    <selector></selector>
    <itos>
        <ito name="getDestinationCity" label="Weather information">
            <AQD>
                <type>
                    <answerType>sys.location.city</answerType>
                </type>
            </AQD>
            <fallback_question>for which city do you want to know the weather?</fa
            <required>true</required>
            <useContext>true</useContext>
            <clearContext>true</clearContext>
        </ito>
    </itos>
```

In this task , we again defined ITO “getDestinationCity” but with a two XML tags <useContext> and <clearContext> to true.
With this configuration, if the ITO is used in earlier conversation e.g. London, the user will not be asked again. The ITO will be filled in with London. Also since the “clearContext” has been set to true , it will also cleared from history as task gets completed.. So next time user asks the Weather Information , Bot will prompt user with fallback question “for which city do you want to know the weather?”

Appendix -7 – Storing and passing the information at Task level

For certain scenarios we need a feature where we need to capture the information in one task (captured in dummy ITO) and pass it to the another task to fill different ITOs.

For such scenario there is a provision of storeCache , useCache and clearCache flags – Here are the steps to follow

- set <storeCache> to true for ITO of which you want to store the information (e.g. dummy)
- Use XML attribute useCahce and set it to true while defining the task which wants to use this info
- You can clear the CACHE by setting <clearCache> to true in ITO that you think appropriately.

```
<task name="getWeatherInformation" label="Weather information">
    <selector></selector>
    <itos>
        <ito name="getDestinationCity" label="Weather information">
            <AQD>
                <type>
                    <answerType>sys.location.city</answerType>
                </type>
            </AQD>
            <fallbackQuestion>for which city do you want to know the
            <required>true</required>
            <storeCache>true</storeCache>
        </ito>
    </itos>
    <action>
        <httpAction>
```

```
<task name="getTripInformation" label="Book ticket" useCache="true">
    <selector></selector>
    <itos>
        <ito name="getDestinationCity" label="City Name">
            <AQD>
                <type>
                    <answerType>sys.location.city</answerType>
                </type>
            </AQD>
            <fallbackQuestion>where do you want to go?|what city do you want to trav
            <required>true</required>
            <clearCache>true</clearCache>
        </ito>
        <ito name="getNumberOfPersons" label="No of persons">
            <AQD>
                <type>
                    <answerType>sys.number.scale</answerType>
                </type>
            </AQD>
```

Here is the sequence of events –

User -> I want weather information => will switch to getWeatherInformation
Bot-> for which city you want to know weather
User -> London => (London will be stored as a String in CACHE)
Bot -> Temp is London is 12 degree. How may I help you?
User -> I want to Book a ticket => (here along with this utterance , London will also be passed as useCache attribute is set to TRUE)
Bot -> For How many persons => (The getDestinationCity will automatically filled with London and will also clear it from CACHE)

– note-“allowOverAnswering” flag is set to true)

Appendix -8 – APIs to get DDF and current task stack …..1/4

In order to get the Dialog definition following API is available
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/getDDF
e.g. http://192.168.0.103:8080/hmi/bot/d1-YDEHE06UPRMM/getDDF
Here is a sample response -


```
{
  "allowCorrection": true,
  "name": "trip",
  "company": "xyz",
  "useIntentEngine": true,
  "allowDifferentQuestion": true,
  "useSODA": true,
  "allowSwitchTasks": true,
  "allowOverAnswering": true,
  "tasks": [
    {
      "name": "start",
      "label": "",
      "itos": [
        {
          "name": "welcome",
          "label": "",
          "type": "open_ended"
        }
      ]
    },
    {
      "name": "getTripInformation",
      "label": "Book ticket",
      "itos": [
```


In order to get the active tasks on stack following API is available
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/tasks
e.g. http://192.168.0.103:8080/hmi/bot/d1-YDEHE06UPRMM/tasks
Here is sample response -


```
{
  "tasks": [
    {
      "name": "getTripInformation",
      "label": "Book ticket",
      "entities": [
        {
          "name": "getDestinationCity",
          "label": "To city",
          "type": "sys.location.city",
          "value": null
        },
        {
          "name": "getNumberOfPersons",
          "label": "No of persons",
          "type": "sys.number",
          "value": null
        },
        {
          "name": "getStartDate",
          "label": "Start Date",
          "type": "sys.temporal.date",
          "value": null
        },
```

Appendix -8 – APIs to get dialogs, terminate instance…2/4

In order to get dialogs between user and bot following API is available
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/getDialog
e.g. http://192.168.0.103:8080/hmi/bot/d1-YDEHE06UPRMM/getDialog

Here is sample response-


```
{
  "user": "John",
  "userAgent": "[Mozilla/5.0 (Windows NT 10.0; Win64; x64) A
  "clientIP": "10.88.250.101",
  "instanceID": "d1-FZP1O9PXOKGG",
  "timeStamp": "2017-05-19 18:42:27",
  "dialog": [
    {
      "S": "How may I help you?"
    },
    {
      "U": "I want to book  a ticket"
    },
    {
      "S": "where do you want to go?"
    }
  ]
}
```


In order to terminate the dialog instance following API is available
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/kill
e.g. http://192.168.0.103:8080/hmi/bot/d1-YDEHE06UPRMM/kill

Here is a sample response


```
{
  "response": "INFO:Removed instance d1-3BXY78BBPJ0M"
}
```


Get Dialog Failure Info

In order to get dialog failure count in dialog (bot failed to answer) following API is available
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/failureInfo
e.g. http://192.168.0.103:8080/hmi/bot/d1-NDMMAMPK0XFS/failureInfo

Here is a sample response


```
{
    "sessionId": "d1-PHOZBPO87GBM",
    "failures": 1
}
```

Appendix -8 – Bot engine status and custom entities…3/4

In order to get bot engine status at given time following API is available (it gives id and current failures in responding to user)
Method -> **POST**
URL -> http://<IP>/hmi/status
e.g. http://192.168.0.103:8080/hmi/status

Here is sample response-


```
{
    "status": {
        "domain": "trip",
        "currentSessions": "2",
        "uiType": "RESTInterface",
        "URI": "https://192.168.0.103:8080/",
        "startedOn": "Mon Feb 12 21:19:41 IST 2018",
        "sessionIDs": [
            {
                "failures": 0,
                "id": "d2-ZHRE0LMYXWFC"
            },
            {
                "failures": 1,
                "id": "d1-PHOZBPO87GBM"
            }
        ]
    }
}
```


In order to obtain custom entities from engine to make BOT client more interactive by providing the user option , one can use this API
Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/getEntityData
e.g. http://192.168.0.103:8080/hmi/bot/d1-YDEHE06UPRMM/getEntityData
Here is sample response-


```
{
  "name": "trip",
  "version": 1.1,
  "items": [
    {
      "values": [
        "Hatch back",
        "Sedan"
      ],
      "name": "item_1"
    },
    {
      "values": [
        "all",
        "broker",
        "claim",
        "support",
        "individual"
      ],
      "name": "item_5"
    }
  ]
}
```

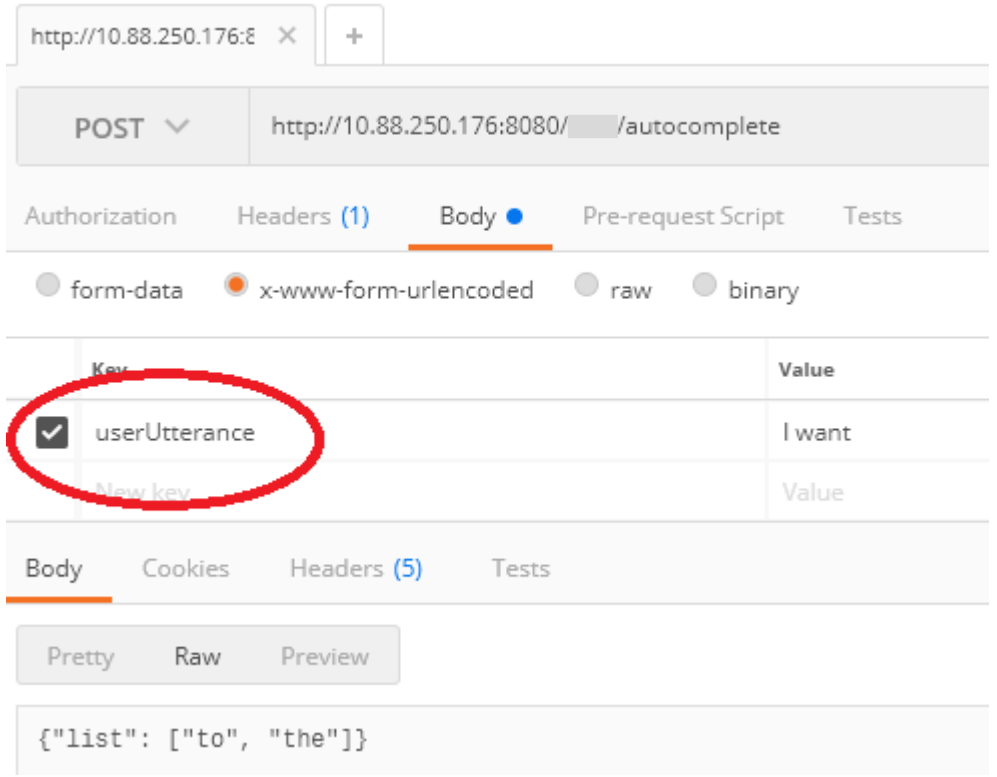
Appendix -8 – APIs to get autocomplete data …4/4

In order to provide the user with recommended words while typing on chat window , bot engine provides list of words based on the users current utterance. Following steps are required to be taken for using this feature

1- Create the corpus (contains all the possible utterances and FAQ Qs that user may type) in /res/autocomplete/data folder with the file name as corpus.txt
2- Call below API from BOT client as user types in (ONCHANGE event of JavaScript in text input field)
3- Get the recommend list of words from bot engine by calling following API

Method -> **POST**
URL -> http://<IP>/hmi/autocomplete
URL encoded body parameter -> userUtterance="I want "
e.g. http://10.88.250.233:8080/hmi/autocomplete

NOTE: if you have uploaded new corpus to BOT , please delete the Pickle file (.pkl) file in model folder.
Here is sample response-

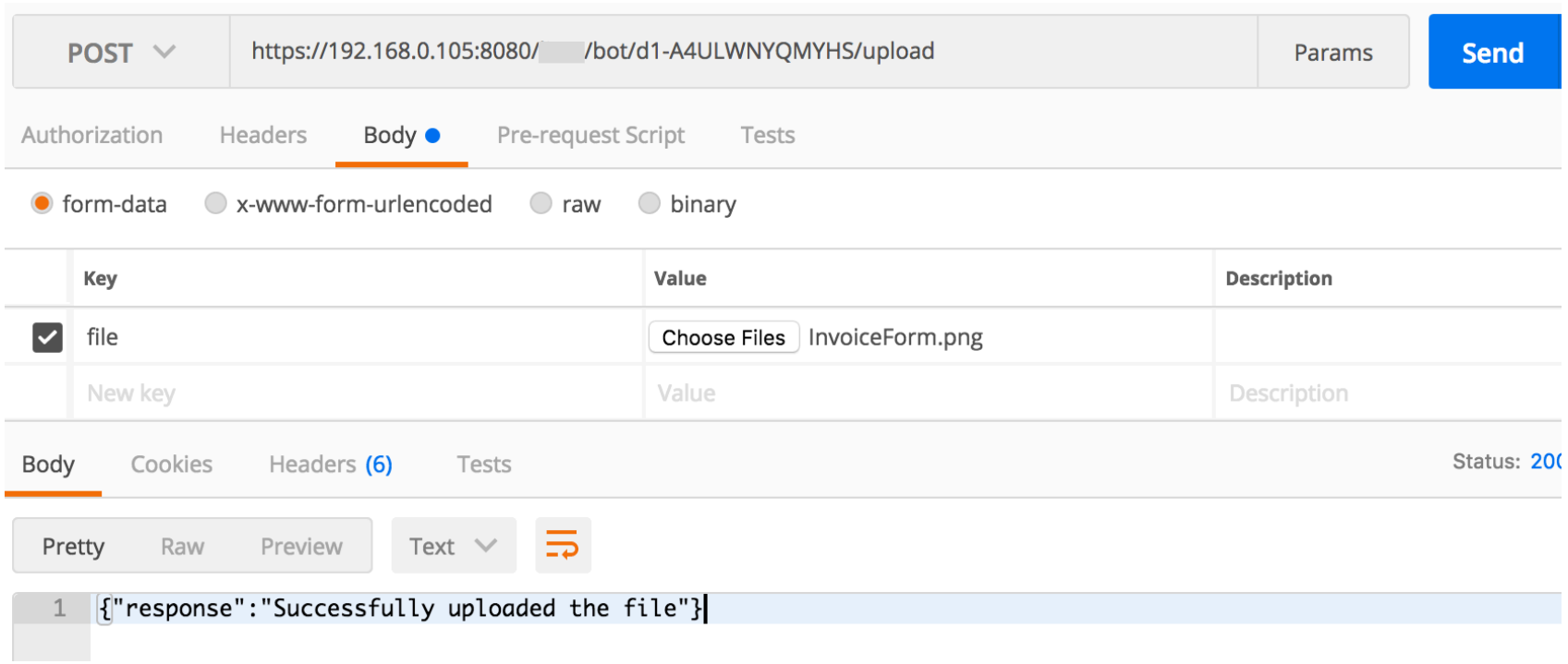


Appendix -9 – APIs to upload the content to server

In order to provide the user to upload the file content to BOT server following API is available

Method -> **POST**
URL -> http://<IP>/bot/{instance_id}/upload
Form Data body parameter -> file=<Selected File>
e.g. https://192.168.0.105:8080/hmi/bot/d1-A4ULWNYQMYHS/upload

1- On client user will select the file to be uploaded.
2- Once uploaded the file will be stored on server in /res/upload folder with following prefix - <User>_<Time>_<FileName>
Example = > John_Sun Aug 20 23/29/08 IST 2017_InvoiceForm.png



Appendix -10 – License

Three types of license are available

1- LIFE_TIME -> does not come with expiry date
2- SINGLT_TIME -> One time license with expiry date
3-TRIAL - Trial for experimentation with expiry date

- All the license keys and signatures are stored in /res/keys folder (viz. license.key and license.sig)
- All the license properties are stored in /res/config/license.properties folder (**CAUTION** - Do not tamper with it)

```
#Please do not update this file
EMAIL=xyz@gmail.com
COMPANY=xyz
LICENSE_TYPE=life_time
EXPIRATION=0000-00-00
VERSION=1.2
```

Appendix -11 – Creating Hindi Bot

One can create Hindi Bot by following below mentioned steps -

1- Create the <domain>_hi.json file that contains all the possible user utterances in Hindi (e.g. in this case “trip_en.json” file to be stored in /res/intents/data folder)

2 – Add domain specific synonyms , if any to “synonyms_hi.txt” file located in /res/dictionary folder

3 – Create the DDF file <domain>.xml that contains all hindi specific messages. Ensure <globalLangauge> set to “hi”. (e.g. in this case trip.xml to be stored in /res/dialogues folder)

4 – The utterances in Hindi are processed in English so the HINDI is transliterated into ENGLISH during utterance processing by BOT engine

e.g. मै इकॉनमी क्लास से जाऊंगा => mai ikonami klasa se ja'unga

So all the ITOs need to be accordingly created. Example below shows Custom.item_1 ITO for getting class of travel -

```
{
    "domain": "trip",
    "tasks": [{
        "name": "getTripInformation",
        "utterances": ["मैं टिकट बुक करना चाहता हूँ",
        "मुझे टिकट बुक करना है",
        "मुझे रिज़र्वेशन कराना है",
        "मुझे रिजर्वेशन करना है",
        "मैं यात्रा करना चाहता हूँ",
        "मैं टिकट आरक्षित करना चाहता हूं",
        "क्या आप कृपया मेरे लिए टिकट बुक कर सकते हैं"]
    },
    {
        "name": "getWeatherInformation",
        "utterances": ["मौसम कैसा है?",
```

```
<startTaskName>getWeatherInformation</startTaskName>
<globalLanguage>hi</globalLanguage>
<useSODA>true</useSODA>
<allowSwitchTasks>true</allowSwitchTasks>
<allowOverAnswering>true</allowOverAnswering>
<allowDifferentQuestion>true</allowDifferentQuestion>
<allowCorrection>true</allowCorrection>
<useIntentEngine>true</useIntentEngine>
<tasks>
    <task name="start"> ... </task>
    <task name="getTripInformation" label="टिकट बुकिंग"> ...
```

```
##DO NOT REMOVE THIS LINE
Economy=Economy
ikonami=Economy
First=First
Business=Business
Busines=Business
```

To accommodate the transliteration issues

In order to know the transliterated words that needs to go in ITO follow below steps –

- create file utternce.dat in /res folder that contains all possible utterances
- go to hmi.jar and run below command
- *>java -cp hmi.jar cto.hmi.bot.util.Transliteration -f /res/utterance.dat –t DEV_TO_ENG ..(for European use EU_TO_ENG)*
- It will process file and will generate the output file utterance_out.dat with transliterated words that can be used in ITO.

```
मै  business क्लास से जाऊंगा |mai  business klasa se ja'unga
मै  इकॉनमी  क्लास से जाऊंगा |mai  ikonami  klasa se ja'unga
```

Appendix -12 – Creating AR based dialogue

In order to create the augmented reality application one need to take following approach.

- The user enters into either virtual room or AR android application where bot will greet the user. (this is the default mode of interaction)
- When user gets into conversation with intent that needs the AR feature the event will be triggered and user will be taken into a AR mode (ViewID = ARMode). The app will move from BOT mode to AR mode (cameras will be turned ON). The user will continue to interact till the “ViewID” is “ARMode” – (this is mainly DIY mode where user will follow steps suggested by BOT using “ARMode”)
- User can make use of “navigationBar” menu to navigate through different steps.
- When user wants to take help of external human expert then the mode is switched from “ARMode” to “RTCMode” (real time communication) <TBD>


```
"message": {
    "chat": "OK. Identifying type..",
    "userUtterance":"",
    "ARdata": {
       "viewID":"ARMode",
       "object":"Router",
       "overlayItems":[
           {
               "name":"Text1",
               "lable":"STEP-1",
               "type":"text",
               "size":"10",
               "position":"X,Y,Z",
               "rotation":"0",
               "color":"R,G,B",
               "isClickable":"false"
           },{
               "name":"Image1",
               "lable":"",
               "type":"image",
               "size":"1",
               "position":"X,Y,Z",
               "rotation":"0",
               "color":"R,G,B",
               "isClickable":"false"
           }
       ],
        "navigationBar":{
            "left_2":"cancel",
            "left_1":"previous",
            "center_0":"play",
            "right_1":"next",
            "right_2":"mic"
        },
        "menuBar":{
            "menu_1":"remoteExpert",
            "menu_2":"logout"
        }
    },

}
```

Appendix -13 – Supporting new language to BOT

In order to add new language to BOT following steps are required

- For generic conversation add required utterances in AIML format in /res/bots/aiml/Generic_xx.aiml file. E.g.
  <category><pattern>jeg har det godt</pattern> <template>godt at høre, at</template> </category>
- Add necessary utterances in /res/bots/aiml/Nlgfiller.aiml to cover the NLU generation.
- Create the dialog file that is specific to given language. Ensure that XML tag <globalLanguage> specifiec the language code e.g. da for Danish
- Add all the stopwords in /res/dictionary folder and name it as stopwords_xx.txt (where xx is language code)..DO=dog etc.
- Add synonyms ,if any to /res/dictionary folder and name it as synonyms_xx.txt (where xx is language code) .. bestil,reservere
- All ITOs in /res/entities needs to be created in transliterated English for engine to identify.
- Create the intents and its utterance variations in /res/intents/data/<domain>_xx.json file.
- Create the dialogue and events in /res/dialogues and /res/events folder respectively

(This section is not for bot builder , only for bot developer)

- Add necessary utterances to SODA classifier (/res/dictionary)to identify whether utterance is of information, command or seek type.
- Add the locale and region combination to DailogManagerContext , and transliteration details in DialogManagerHelper class
- Add the standard messages that are part of core conversation i.e. SORRY_MSG(Sorry I did not understand that), GOT_MSG(I go that) etc. to /res/config/appMessages_xx.properties file. (where xx is language code. For example, da for danish)