Oracle Intelligent Bots Training

Lab 2

Define the Chatbot Flow Using BotML

In this lab, you will add BotML to your MasterBot. The code will later on support each of your intents. However, we will be focusing on the **Balances** intent where we will add its variables and then add all the states that are needed to complete its actions. When you're done, you will test the intent to make sure that it work as expected.

Before You Begin

You need the following files for this lab, which are located in the `/labfiles/code` directory:

- `FirstBotYAML-Balances.txt`
- `FirstBotYAML-Complete.txt`

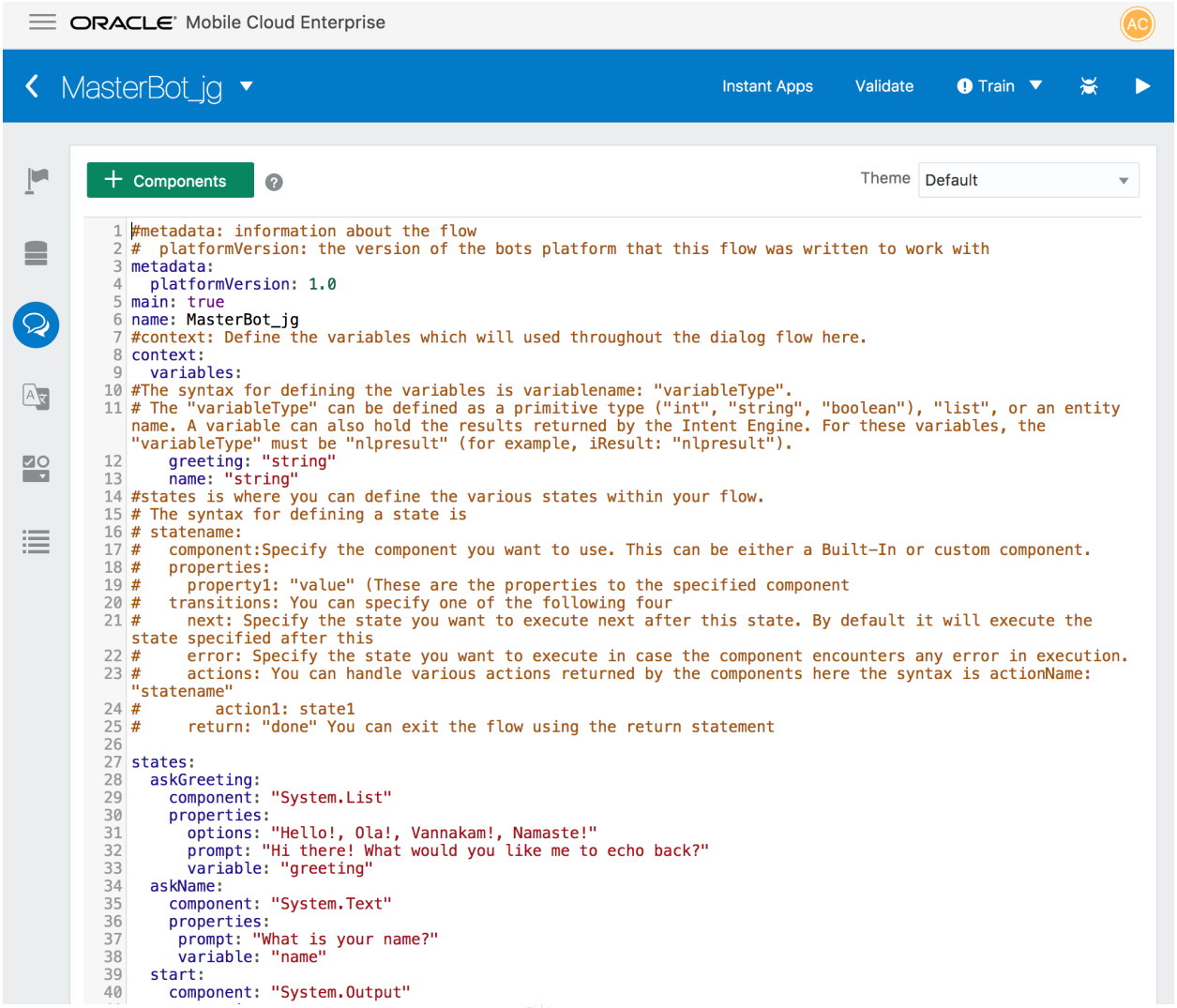
Step 1: Include the Code that Supports the Balances Intent

In this section, you add code to the flow of your chatbot that supports the Balances intent. We'll examine its components and use it as an example of how you should set up other components.

1. Log in to your MasterBot_firstNameLastName chatbot, and click on the Flows icon in the left navbar. You will see the default BotML "Hello" code in the editor. We will not be needing any of it, so delete it (click in the editor -> ctrl+a -> Delete).

Oracle Intelligent Bots Training

Lab 2



2. If you haven't already done so, locate the `FirstBotYAML-Balances.txt` in the `/labfiles/code` directory, open it, copy its contents into the editor and then click **Validate**.

Oracle Intelligent Bots Training

Lab 2

```yaml
metadata:
  platformVersion: "1.0"
main: true
name: "FinancialBotMainFlow"
context:
  variables:
    accountType: "AccountType"
    iResult: "nlpresult"
states:
  intent:
    component: "System.Intent"
    properties:
      variable: "iResult"
      confidenceThreshold: 0.4
    transitions:
      actions:
        Balances: "startBalances"
        unresolvedIntent: "unresolved"
  startBalances:
    component: "System.SetVariable"
    properties:
      variable: "accountType"
      value: "${iResult.value.entityMatches['AccountType'][0]}"
    transitions: {}
  askBalancesAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "For which account do you want your balance?"
      variable: "accountType"
    transitions: {}
  printBalance:
    component: "System.Output"
    properties:
      text: "Balance for ${accountType.value} is $500"
    transitions:
      return: "printBalance"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```

3. Let's look at the code we have and dissect what it represents.
    - Here we see a variety of sections to the code: the header followed by the declaration of content variables and then the intent states.

Oracle Intelligent Bots Training

Lab 2

- The `accountType` variable is used for recording and displaying the account for which the balance is requested. It's used in the `startBalances` state to check if what the user has typed in matches any of the values included in the entity definition. It is also used in the `askBalancesAccountType` state to store what is entered if the account type is not specified in the `startBalances` state. Finally, the account type is used in the `printBalance` state when the balance is displayed to the user.
- There are five states defined. The `intent` state is the result of Intent classification and entity resolution that's provided by the Intent Engine as the `nlpresult` (of which `iResult` is a type). In other words, this variable (`iResult`) holds the result of the Intent Engine (that is, the intent and entity resolution) from the input text provided by the user. The `actions` show a `startBalances` state which implements the Balances intent. There is also another state listed that's used when the intent cannot be resolved: at the end of the flow, notice the `unresolved` state, which is reached if no other state is fulfilled.
- Below the `intent` state are all the other states that the chatbot uses. In our case, the `startBalance` state is the entry point for Balances, regardless if the `accountType` variable is set in `StartBalance`, or if the `nlpresult` (that is, the `iResult` variable) provides this value from parsing user input. In this case, `askBalancesAccountType` does not attempt to set the variable as it's already been set (and as a result, the list of options will not be displayed). If the value isn't set in the `nlpresult`, however, the flow attempts to set it using list of options. If an account type is not mentioned in the user's message, then a `System.List` component displays the valid account types as options. Once an account type is set, the control of the flow drops down to the `printBalance` state, where a `System.Output` component is used to display what the account type is and the balance amount. The final state is entered if the intent cannot be resolved.

Oracle Intelligent Bots Training

Lab 2

Header

Context Variables

Intent States

Start Balance

Ask Balance Account Type

Print Balance

Unresolved

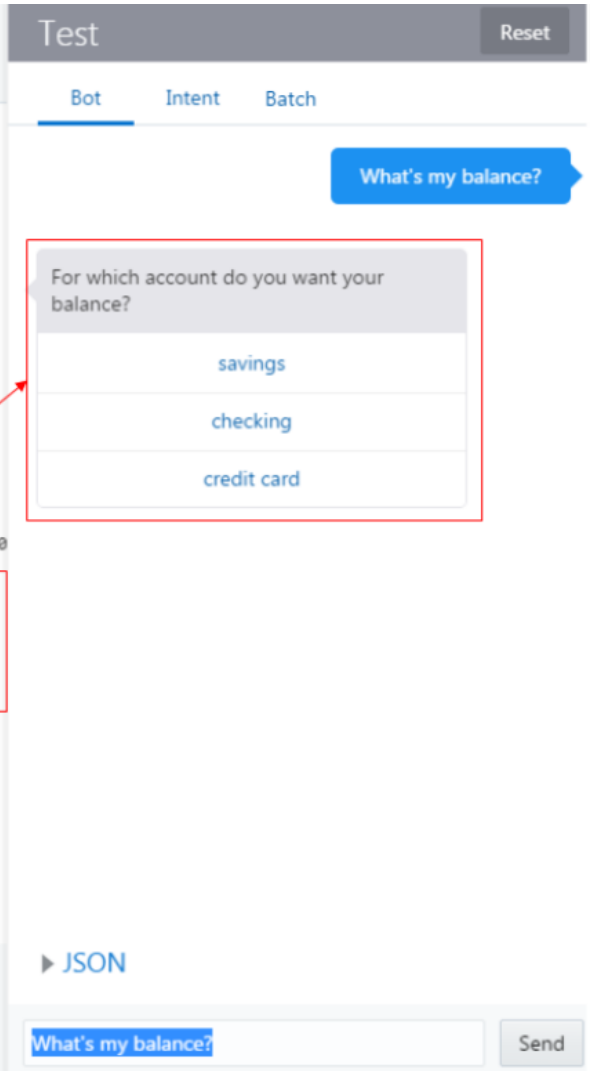
```
1  metadata:
2    platformVersion: "1.0"
3  main: true
4  name: "FinancialBotMainFlow"
5  context:
6    variables:
7      accountType: "AccountType"
8      iResult: "nlpresult"
9  states:
10   intent:
11     component: "System.Intent"
12     properties:
13       variable: "iResult"
14       confidenceThreshold: 0.4
15     transitions:
16       actions:
17         Balances: "startBalances"
18         unresolvedIntent: "unresolved"
19   startBalances:
20     component: "System.SetVariable"
21     properties:
22       variable: "accountType"
23       value: "${iResult.value.entityMatches['AccountType'][0]}"
24     transitions: {}
25   askBalancesAccountType:
26     component: "System.List"
27     properties:
28       options: "${accountType.type.enumValues}"
29       prompt: "For which account do you want your balance?"
30       variable: "accountType"
31     transitions: {}
32   printBalance:
33     component: "System.Output"
34     properties:
35       text: "Balance for ${accountType.value} is $500"
36     transitions:
37       return: "printBalance"
38   unresolved:
39     component: "System.Output"
40     properties:
41       text: "Unable to resolve intent!"
42     transitions:
43       return: "unresolved"
```

4. To test the flow code, click the **Play** button in the upper right, select the Bot tab in the Tester.
5. Enter *What's my balance?* in the Message area, and then click **Send**. You should see a list of all the accounts included in the `System.List` component.
6. Notice the prompt and the list of accounts. They originate from the `askBalanceAccountType` state.

Oracle Intelligent Bots Training

Lab 2

```
1  metadata:
2    platformVersion: "1.0"
3  main: true
4  name: "FinancialBotMainFlow"
5  context:
6    variables:
7      accountType: "AccountType"
8      iResult: "nlpresult"
9  states:
10   intent:
11     component: "System.Intent"
12     properties:
13       variable: "iResult"
14       confidenceThreshold: 0.4
15     transitions:
16       actions:
17         Balances: "startBalances"
18         unresolvedIntent: "unresolved"
19   startBalances:
20     component: "System.SetVariable"
21     properties:
22       variable: "accountType"
23       value: "${iResult.value.entityMatches['AccountType'][0
24     transitions: {}
25   askBalancesAccountType:
26     component: "System.List"
27     properties:
28       options: "${accountType.type.enumValues}"
29       prompt: "For which account do you want your balance?"
30       variable: "accountType"
31     transitions: {}
32   printBalance:
33     component: "System.Output"
34     properties:
35       text: "Balance for ${accountType.value} is $500"
36     transitions:
37       return: "printBalance"
38   unresolved:
39     component: "System.Output"
40     properties:
41       text: "Unable to resolve intent!"
42     transitions:
43       return: "unresolved"
```

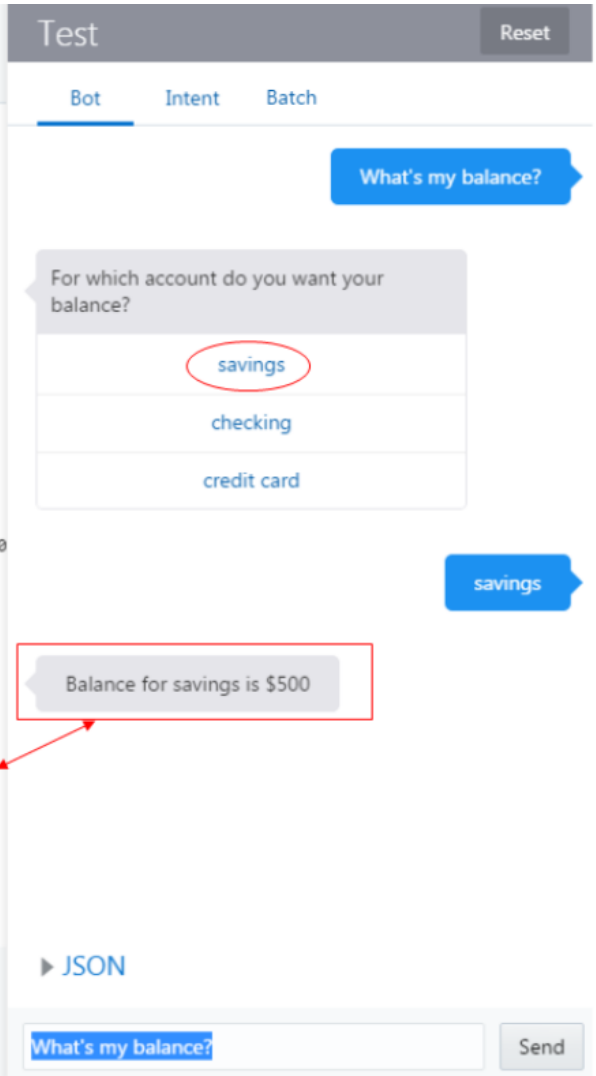


7. Next, select an account. The return should display the `System.Output` component from the `printBalance` state and show you the amount in that account. Notice that the displayed value of $500 is hard-coded in the `System.Output` component (“`Balance for ${accountType.value} is $500`”).

Oracle Intelligent Bots Training

Lab 2

```
metadata:
  platformVersion: "1.0"
main: true
name: "FinancialBotMainFlow"
context:
  variables:
    accountType: "AccountType"
    iResult: "nlpresult"
states:
  intent:
    component: "System.Intent"
    properties:
      variable: "iResult"
      confidenceThreshold: 0.4
    transitions:
      actions:
        Balances: "startBalances"
        unresolvedIntent: "unresolved"
  startBalances:
    component: "System.SetVariable"
    properties:
      variable: "accountType"
      value: "${iResult.value.entityMatches['AccountType'][0
    transitions: {}
  askBalancesAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "For which account do you want your balance?"
      variable: "accountType"
    transitions: {}
  printBalance:
    component: "System.Output"
    properties:
      text: "Balance for ${accountType.value} is $500"
    transitions:
      return: "printBalance"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```



8. Click **Reset** and then try out some other messages, including some with the account in the message text to see how your chatbot responds. (E.g “How much do I have in savings?”)

Oracle Intelligent Bots Training

Lab 2

Step 2: Include the Code that Supports the Send Money and Track Spending Intents

Before we finish, we will also insert he BotML to cover our other intents, but we won't cover them now.

Locate the `FirstBotYAML-Complete.txt` in the `/labfiles/code` directory, open it and copy its contents. In the Flow editor, remove ALL current content and paste the copied BotML and then click **Validate**.

Good for you! You have now completed this lab. In the next lab, you'll learn how to add custom components to your chatbot.