

# Oracle MOOC: Oracle Intelligent Bots

Session 2

## Integrate Custom Services Hosted on OMCe

In this lab, you connect your chatbot to OMCe (Oracle Mobile Cloud, Enterprise) and consume some custom components. At this point, you have the skeleton of a working chatbot – it recognizes several intents and prompts you when you don't provide the needed information. However, the only actual work it does during the handling of an intent is to output a statement – there's no business logic.

Recall that **each state in a chatbot's dialog flow has a component associated with it that is invoked upon entering that state**. So far, you've been using the built-in system components (the ones that begin with `System.`). In Oracle Intelligent Bots, you provide the business logic through custom components. **Custom components are called by REST services that chatbot developers create and deploy onto any infrastructure that can expose the components on the Internet**. Once the components are available, chatbot developers can then configure their chatbots to call them.

We've provided a set of custom components that implements the business logic for the MasterBot. This lab will walk you through the steps of configuring your MasterBot so it can access them. You will then modify the MasterBot's dialog flow to invoke these custom components.

---

## Before You Begin

Have the following files from the `labfiles.zip` close at hand. You can find all of them in the `labfiles/code` directory:

- `MCSSTextServices.txt`
- `CustomPrintBalance.txt`
- `CustomStartPayment.txt`
- `CustomTrackSpending.txt`
- `CustomMasterBotYAML.txt`

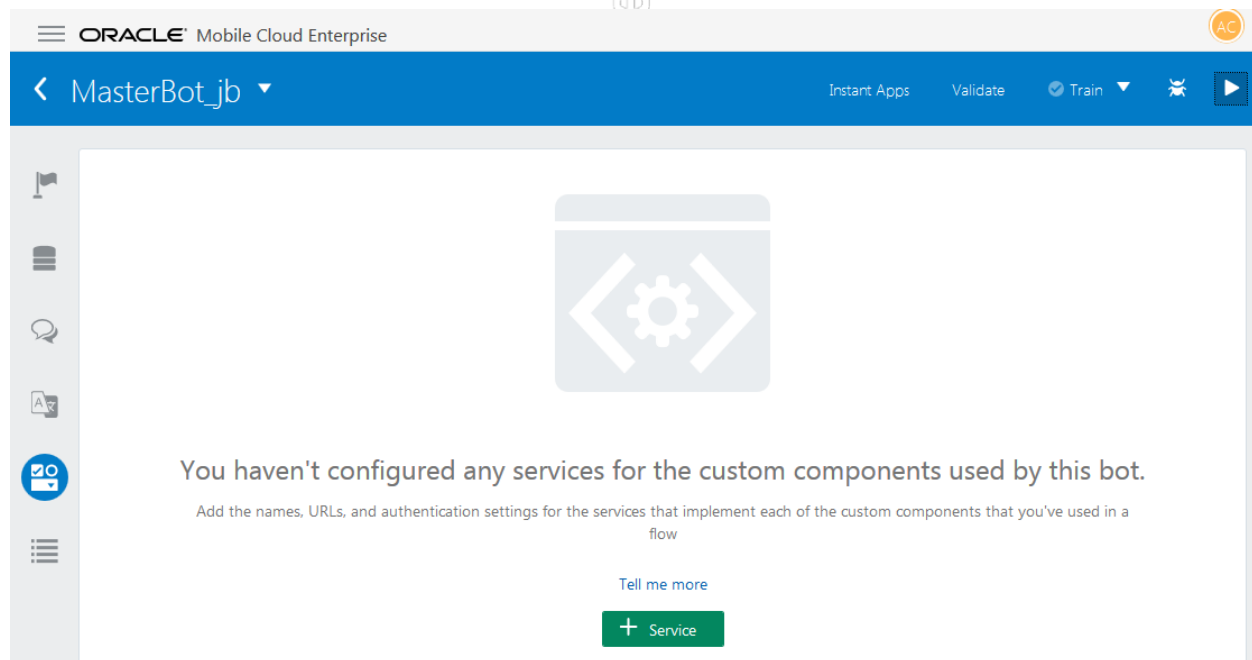
## Step 1: Consume the Custom Components from OMCe

In this step, you'll create some custom components that use Oracle Mobile Cloud, Enterprise (OMCe) APIs. Here, you'll connect your chatbot tenant to a backend that accesses APIs. When you're done, your chatbot will then include the definitions for a

# Oracle MOOC: Oracle Intelligent Bots

custom components container. The container will hold five components that retrieve balances, enable payments, and track spending. The container also returns information about two system-related variables.

1. Go back into your MasterBot\_xx and select the Components icon from the left navbar.
2. Next, click the **Add Service** button. Your page should show there are currently no component services configured for your chatbot.



3. To connect to the OMCe backend, we need some information from both the backend and the APIs that it accesses, so let's look at what we need.

We'll provide some background: if you were to navigate to OMCe, you would find the mobile backend called Banking\_00. By clicking the **Settings** icon in the left navbar, you would see information that looks something like the following image.

# Oracle MOOC: Oracle Intelligent Bots

APPLICATIONS > MOBILE BACKENDS > Banking\_00 1.0

Diagnostics

Settings

Clients

APIs

Storage

Users

Notifications

App Policies

**Banking\_00 1.0**  
has been created and is ready to use in this environment.

## Suggested next steps:

Download a quickstart project and SDK for your platform.

Learn more about the built-in services for push notifications, object storage, and mobile user management.

Banking\_00

MBE for Banking Bot APIs

## Access Keys ?

### HTTP Basic



Refresh

Revoke

Mobile Backend ID

c8738af1-6d38-4a77-b689-33306d128d02

Anonymous Key

WU9EQV9MRUIBX01PQkIMRV9BTk9OWU1PVVNfQVBQSQUQ6cDBnZGtsUmUucm0zbmk=

### OAuth Consumer



Refresh

Revoke

Client ID

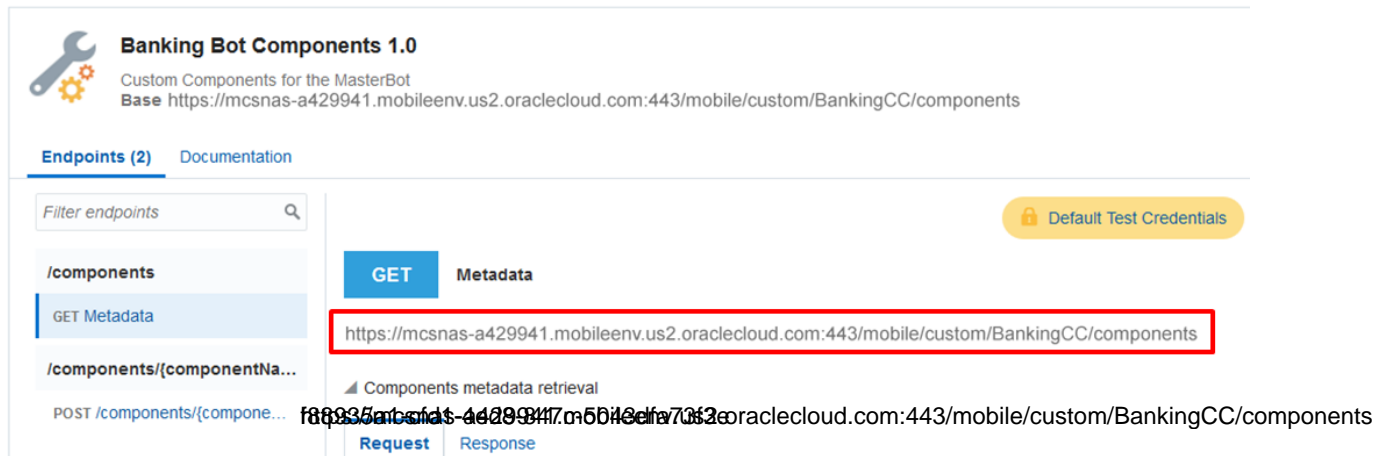
36e1913e-cea4-48ec-9d45-45c10edd2e0d

The two pieces of information you need to connect to the backend and access its APIs are the **Backend ID** and the **Anonymous Key** that display in this page. The Anonymous Key isn't shown by default. You need to click the **Show** button to reveal it.

You also need the URL of the API that you want to access. You can find this URL by:

1. Clicking **APIs** in the left navbar.
2. Selecting the Banking Components
3. Copying the GET URL that's displayed under the /components endpoint.

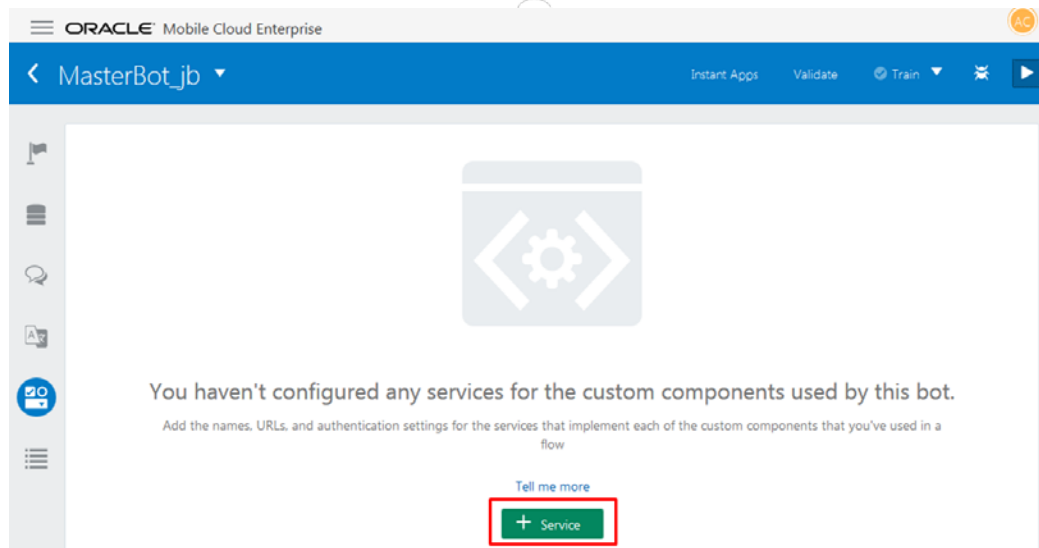
# Oracle MOOC: Oracle Intelligent Bots



To complete this exercise, you don't have to navigate to the mobile backend. All of the information that you need is in the `MCSServices.txt` file, which is located in the `labfiles/code` directory.

To configure a component service using these values:

1. In your MasterBot, click the green **Add Service** button to configure a new component service to access the backend.



2. Enter `MasterBotComponents` as the service name.
  3. Copy the following values for the backend ID and the Metadata URL from the `MCSServices.txt` file.
- Backend ID: `f88935a1-cfd1-4ed8-847c-5043dfa73f3e`

# Oracle MOOC: Oracle Intelligent Bots

- Metadata URL: `https://mcsnas-a429941.mobileenv.us2.oraclecloud.com:443/mobile/custom/BankingCC/components`
- 4. Click **Use anonymous access**.
- 5. Enter the following value for the Anonymous Key from the `MCSServices.txt` file:  
`QTQyOTk0MV9NQ1NOQVNfTU9CSUxFX0FOT05ZTU9VU19BUFBjRDpFb3owcWVfYz  
d0ZnJ3Zw==`

**Important:** Copy all of these values from the `MCSServices.txt` file, not from this PDF.

The screenshot shows the 'Create Service' dialog in the Oracle Mobile Suite. The dialog has a title bar with a close button. Inside, there are several input fields and a checkbox. The 'Name' field is labeled with an asterisk and contains 'MasterBotComponents'. The 'Description' field is labeled 'Description' and contains 'Optional short description for this service.'. Below these, there are two radio buttons: 'Mobile Cloud' (selected) and 'Other'. Then, there are three fields with question mark icons: 'Backend ID' (f88935a1-cfd1-4ed8-847c-5043dfa73f3e), 'Metadata URL' (https://mcsnas-a429941.mobileenv.us2.oraclecloud.com:443/mot), and 'Anonymous Key' (CSUxFX0FOT05ZTU9VU19BUFBjRDpFb3owcWVfYz d0ZnJ3Zw==). There is a checkbox labeled 'Use anonymous access' which is checked. At the bottom, there is a section for 'Optional HTTP Headers' with a question mark icon. A green 'Create' button is located at the bottom right of the dialog.

6. Click **Create**. Now that you've created the MasterBotsComponent service, its details now display in the Components page.

# Oracle MOOC: Oracle Intelligent Bots

The screenshot shows the Oracle Intelligent Bots console interface. At the top, there's a blue header with a back arrow, the text "MasterBot\_jb", and buttons for "Instant Apps", "Validate", "Train", and a bug icon. On the left, a sidebar contains icons for a flag, a list, a chat bubble, a document, and a bot icon. The main area has a "Service" tab with a "+ Service" button and a "Filter" search bar. Below the filter, a list shows "MasterBotComponents" with a right-pointing arrow. To the right of this list, the configuration for "MasterBotComponents" is displayed. It includes fields for "Name" (MasterBotComponents), "Description" (Optional short description for this service.), and "Version" (1.0). There are radio buttons for "Mobile Cloud" (selected) and "Other". Below these are fields for "Backend ID" (f88935a1-cfd1-4ed8-847c-5043dfa73f3e) and "Metadata URL" (https://mcsnas-a429941.mobileenv.us2.oraclecloud.com:443). A checkbox "Use anonymous access" is checked. Below that is an "Anonymous Key" field with a masked value and a "Reset" button. At the bottom, there's an "Optional HTTP Headers" section with a question mark icon. Buttons for "Reload" and "Delete" are at the top right of the configuration area.

- To see the various components that this service provides for your chatbot, click the arrow next to the service name.

This screenshot is similar to the previous one, but the "MasterBotComponents" service is expanded. In the left sidebar, the "MasterBotComponents" entry now has a dropdown arrow. Below it, a list of components is shown, each with an orange expand icon: "ActionFromVariable", "BalanceRetrieval", "ConditionalIsNull", "OutputVariables", "Payments", "SetVariableFromEntityMatc...", "SetVariablesFromFile", and "TrackSpending". The configuration details on the right remain the same as in the previous screenshot.

Among the custom components is the BalanceRetrieval component. Later on, you'll update the BotML definition with this component.

# Oracle MOOC: Oracle Intelligent Bots

Now that you have consumed some pre-built components, let's use them in your BotML code.

## *Step 2: Add your Custom Components to the BotML Flow*

In this section, you will replace some of the BotML code to now access the custom components. These components will then use the APIs associated with the backend to return data from OMCE. For the first steps, you just examine the custom component code and see how it differs from the existing BotML definition. Then, at the end of this section, you will replace the existing BotML definition with code that employs the custom components.

1. Click the **Flows** icon in the left navbar.

Each intent already has a start state in the flow where the business logic is executed. Let's change those states into ones that reference the components provided by the MasterBotComponents service.

# Oracle MOOC: Oracle Intelligent Bots

< MasterBot\_jb ▾

+ Components ?

```

1 metadata:
2   platformVersion: "1.0"
3 main: true
4 name: "FinancialBotMainFlow"
5 context:
6   variables:
7     accountType: "AccountType"
8     toAccount: "ToAccount"
9     spendingCategory: "TrackSpendingCategory"
10    paymentAmount: "CURRENCY"
11    iResult: "nlpresult"
12 states:
13   intent:
14     component: "System.Intent"
15     properties:
16       variable: "iResult"
17       confidenceThreshold: 0.4
18     transitions:
19       actions:
20         Balances: "startBalances"
21         Send Money: "startPayments"
22         Track Spending: "startTrackSpending"
23         unresolvedIntent: "unresolved"
24   startBalances:
25     component: "System.SetVariable"
26     properties:
27       variable: "accountType"
28       value: "${iResult.value.entityMatches['AccountType']}[0]}"
29     transitions: {}
30   askBalancesAccountType:
31     component: "System.List"
32     properties:
33       options: "${accountType.type.enumValues}"
34       prompt: "For which account do you want your balance?"
35       variable: "accountType"
36     transitions: {}

```

2. First, let's work with the Balances intent. Here is what you should see in the printBalance state.

```

36   transitions: {}
37 printBalance:
38   component: "System.Output"
39   properties:
40     text: "Balance for ${accountType.value} is $500"
41   transitions:
42     return: "printBalance"
43 startPayments:

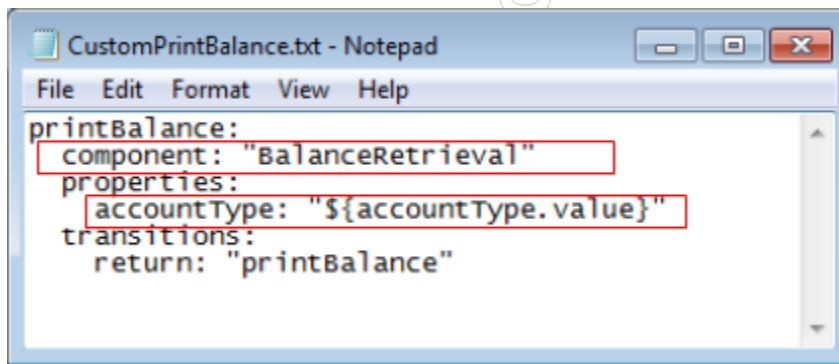
```



# Oracle MOOC: Oracle Intelligent Bots

3. Using the `CustomPrintBalance.txt` file that's located in the `labfiles/code` directory, make the following changes to the `printBalance` state to update it from using a system component to the custom component, `BalanceRetrieval`. There are two changes that you need to make:
  1. First, change the component type from `System.Output` to `BalanceRetrieval`.
  2. Now that you are using the custom component to call the API, you need to pass along the `accountType`. Remove the text property and replace it with `accountType: "${accountType.value}"`.

**Tip:** To avoid indentation errors, copy this code from the `CustomPrintBalance.txt` file. The `printBalance` state should look the following image.



4. Next, let's work with the `Send Money` intent. This one is a bit more complex because it requires adding some new states. We'll start off by explaining what is happening and then you'll replace all the states for that intent with code from a file.

The following image shows all of the code for the `Send Money` states you currently use: `startPayments`, `resolveToAccount`, `askFromAccountType`, `askToAccount`, `resolvePaymentAmount`, `askPaymentAmount`, and `doPayment`.

# Oracle MOOC: Oracle Intelligent Bots

```

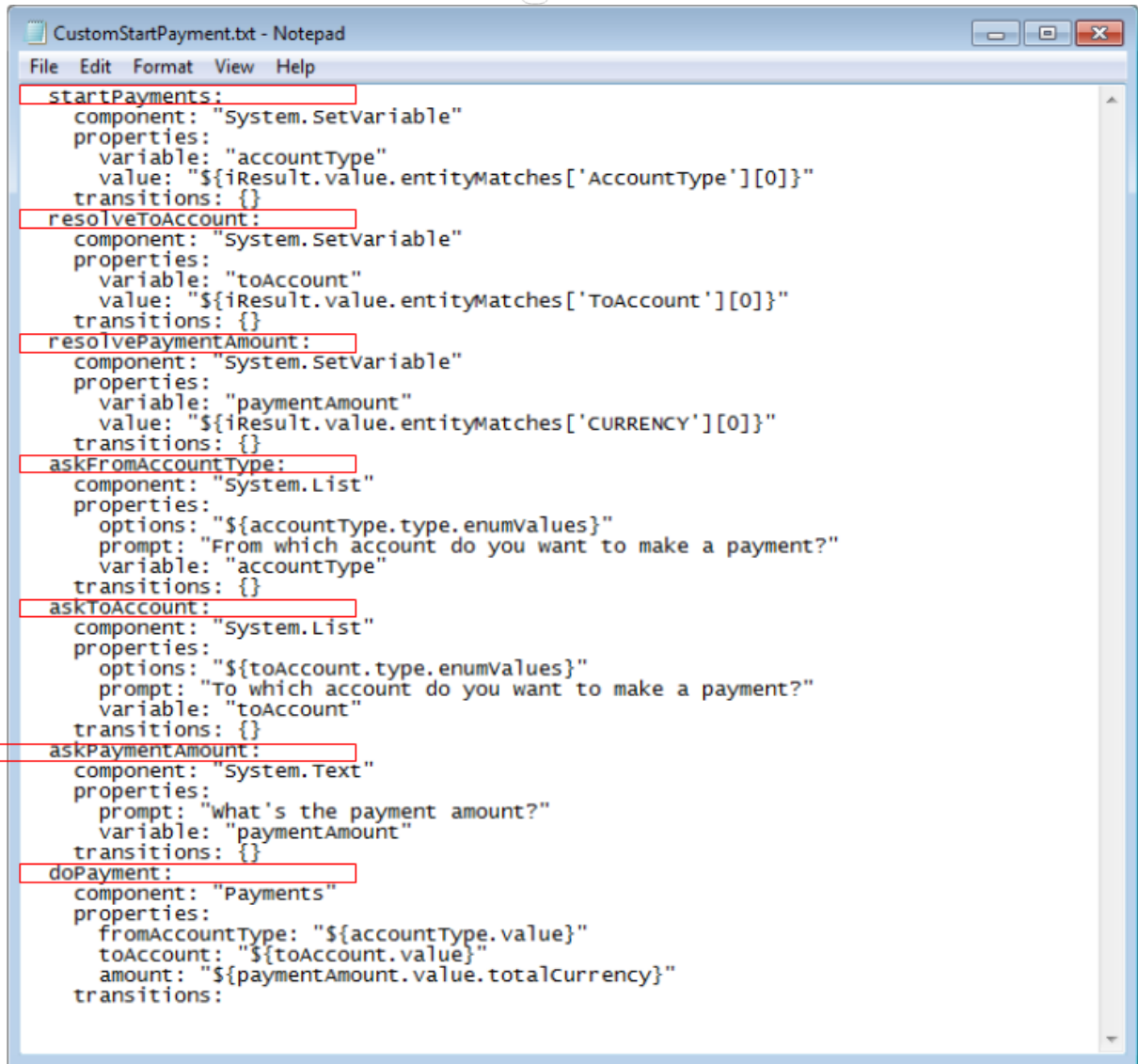
43 startPayments:
44   component: "System.SetVariable"
45   properties:
46     variable: "accountType"
47     value: "${iResult.value.entityMatches['AccountType']}[0]}"
48   transitions: {}
49 resolveToAccount:
50   component: "System.SetVariable"
51   properties:
52     variable: "toAccount"
53     value: "${iResult.value.entityMatches['ToAccount']}[0]}"
54   transitions: {}
55 askFromAccountType
56   component: "System.List"
57   properties:
58     options: "${accountType.type.enumValues}"
59     prompt: "From which account do you want to make a payment?"
60     variable: "accountType"
61   transitions: {}
62 askToAccount:
63   component: "System.List"
64   properties:
65     options: "${toAccount.type.enumValues}"
66     prompt: "To which account do you want to make a payment?"
67     variable: "toAccount"
68   transitions: {}
69 resolvePaymentAmount:
70   component: "System.SetVariable"
71   properties:
72     variable: "paymentAmount"
73     value: "${iResult.value.entityMatches['CURRENCY']}[0]}"
74   transitions: {}
75 askPaymentAmount:
76   component: "System.Text"
77   properties:
78     prompt: "What's the payment amount?"
79     variable: "paymentAmount"
80   transitions: {}
81 doPayment:
82   component: "System.Output"
83   properties:
84     text: "${paymentAmount.value.totalCurrency} paid from ${accountType.value} to ${toAccount.value}"
85   transitions:
86     return: "doPayment"

```

5. The CustomStartPayment.txt file contains all the new code. Open the file and take a look at it.

It contains all of the seven states that currently exist in your chatbot.

# Oracle MOOC: Oracle Intelligent Bots



```

CustomStartPayment.txt - Notepad
File Edit Format View Help

startPayments:
  component: "System.SetVariable"
  properties:
    variable: "accountType"
    value: "${iResult.value.entityMatches['AccountType']}[0]}"
  transitions: {}
resolveToAccount:
  component: "System.SetVariable"
  properties:
    variable: "toAccount"
    value: "${iResult.value.entityMatches['ToAccount']}[0]}"
  transitions: {}
resolvePaymentAmount:
  component: "System.SetVariable"
  properties:
    variable: "paymentAmount"
    value: "${iResult.value.entityMatches['CURRENCY']}[0]}"
  transitions: {}
askFromAccountType:
  component: "System.List"
  properties:
    options: "${accountType.type.enumvalues}"
    prompt: "From which account do you want to make a payment?"
    variable: "accountType"
  transitions: {}
askToAccount:
  component: "System.List"
  properties:
    options: "${toAccount.type.enumvalues}"
    prompt: "To which account do you want to make a payment?"
    variable: "toAccount"
  transitions: {}
* askPaymentAmount:
  component: "System.Text"
  properties:
    prompt: "what's the payment amount?"
    variable: "paymentAmount"
  transitions: {}
doPayment:
  component: "Payments"
  properties:
    fromAccountType: "${accountType.value}"
    toAccount: "${toAccount.value}"
    amount: "${paymentAmount.value.totalCurrency}"
  transitions:

```

- Now let's look at how the new states operate. They do not use any custom components, so there's really nothing new or different from what we've used before. The askPaymentAmount, uses a System.Text component to prompt the user for an amount and a variable to store the payment amount.

# Oracle MOOC: Oracle Intelligent Bots



```
askPaymentAmount:
  component: "System.Text"
  properties:
    prompt: "What's the payment amount?"
    variable: "paymentAmount"
  transitions: {}
```

Likewise, the `startPayments`, `resolveToAccount`, `askFromAccountType` and `askToAccountType` states are all the same, so there's no need to examine them.

However, the `doPayment` state is different. It uses the `Payments` custom component.

7. Click the **Components** tab in the left navbar and then expand the `MasterBotsComponent` service (if it isn't already opened). Looking at the `Payments` custom component, you'll see that it uses five properties: `fromAccountType`, `date`, `recurrence`, `toAccount` and `amount`.

# Oracle MOOC: Oracle Intelligent Bots

The screenshot shows the Oracle Intelligent Bots console. On the left, a sidebar contains navigation icons. The main area is divided into three sections: a top section with a '+ Service' button and a 'Filter' search bar; a middle section titled 'MasterBotComponents' with a list of components including 'ActionFromVariable', 'BalanceRetrieval', 'ConditionalIsNull', 'OutputVariables', 'Payments' (highlighted with a red box), 'SetVariableFromEntityMat...', 'SetVariablesFromFile', and 'TrackSpending'; and a right section titled 'Payments' which contains a 'Properties' table and a 'Supported Actions' section.

Property	Type	Required
fromAccountType	string	true
date	string	false
recurrence	string	false
toAccount	string	true
amount	CURRENCY	true

- Looking at the `doPayment` state definition, you can see that the `fromAccountType`, `toAccount`, and `paymentAmount` properties are all set.
- There's one additional thing to note: In the `variables` node at the top of the BotML definition, `paymentAmount` is set to a system entity, `CURRENCY`.

**doPayment:**

**component:** "Payments"

**properties:**

`fromAccountType: "${accountType.value}"`

`toAccount: "${toAccount.value}"`

`amount: "${paymentAmount.value.totalCurrency}"`

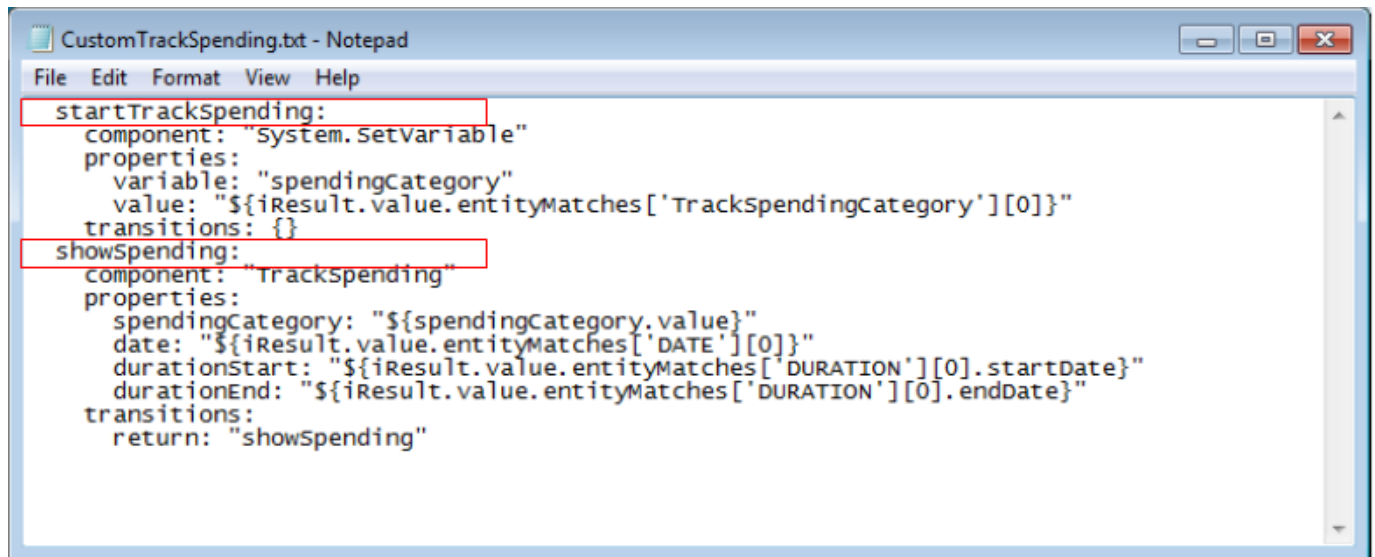
**transitions:**

`return: "doPayment"`

# Oracle MOOC: Oracle Intelligent Bots

10. Finally, let's look at how the last intent, Track Spending. It uses a custom component. You can find all of the code for `startTrackSpending` in the `CustomTrackSpending.txt` file located in the `labfiles/code` directory. Open this file.

The `startTrackSpending` state exists in the current flow, but note that `showSpending` is a new state.



```
startTrackSpending:
  component: "System.SetVariable"
  properties:
    variable: "spendingCategory"
    value: "${iResult.value.entityMatches['Trackspendingcategory'][0]}"
  transitions: {}
showSpending:
  component: "TrackSpending"
  properties:
    spendingCategory: "${spendingCategory.value}"
    date: "${iResult.value.entityMatches['DATE'][0]}"
    durationStart: "${iResult.value.entityMatches['DURATION'][0].startDate}"
    durationEnd: "${iResult.value.entityMatches['DURATION'][0].endDate}"
  transitions:
    return: "showSpending"
```

11. Let's look at how these two states operate.

The `startTrackSpending` uses a `System.SetVariable` component and a `spendingCategory` variable. The `showSpending` state uses the `TrackSpending` custom component (shown in the Components page, below).

# Oracle MOOC: Oracle Intelligent Bots

The screenshot shows the Oracle Intelligent Bots console. On the left, a sidebar contains navigation icons. The main area displays a list of 'MasterBotComponents'. The 'TrackSpending' component is highlighted with a red box. To the right, the 'TrackSpending' component's properties are shown in a table, also highlighted with a red box.

Property	Type	Required
date	string	false
spendingCategory	string	true
durationStart	string	false
durationEnd	string	false

This component sets all four variables.

showSpending:

component: "TrackSpending"

properties:

```
spendingCategory: "${spendingCategory.value}"
date: "${iResult.value.entityMatches['DATE'][0]}"
durationStart: "${iResult.value.entityMatches['DURATION'][0].startDate}"
durationEnd: "${iResult.value.entityMatches['DURATION'][0].endDate}"
```

transitions:

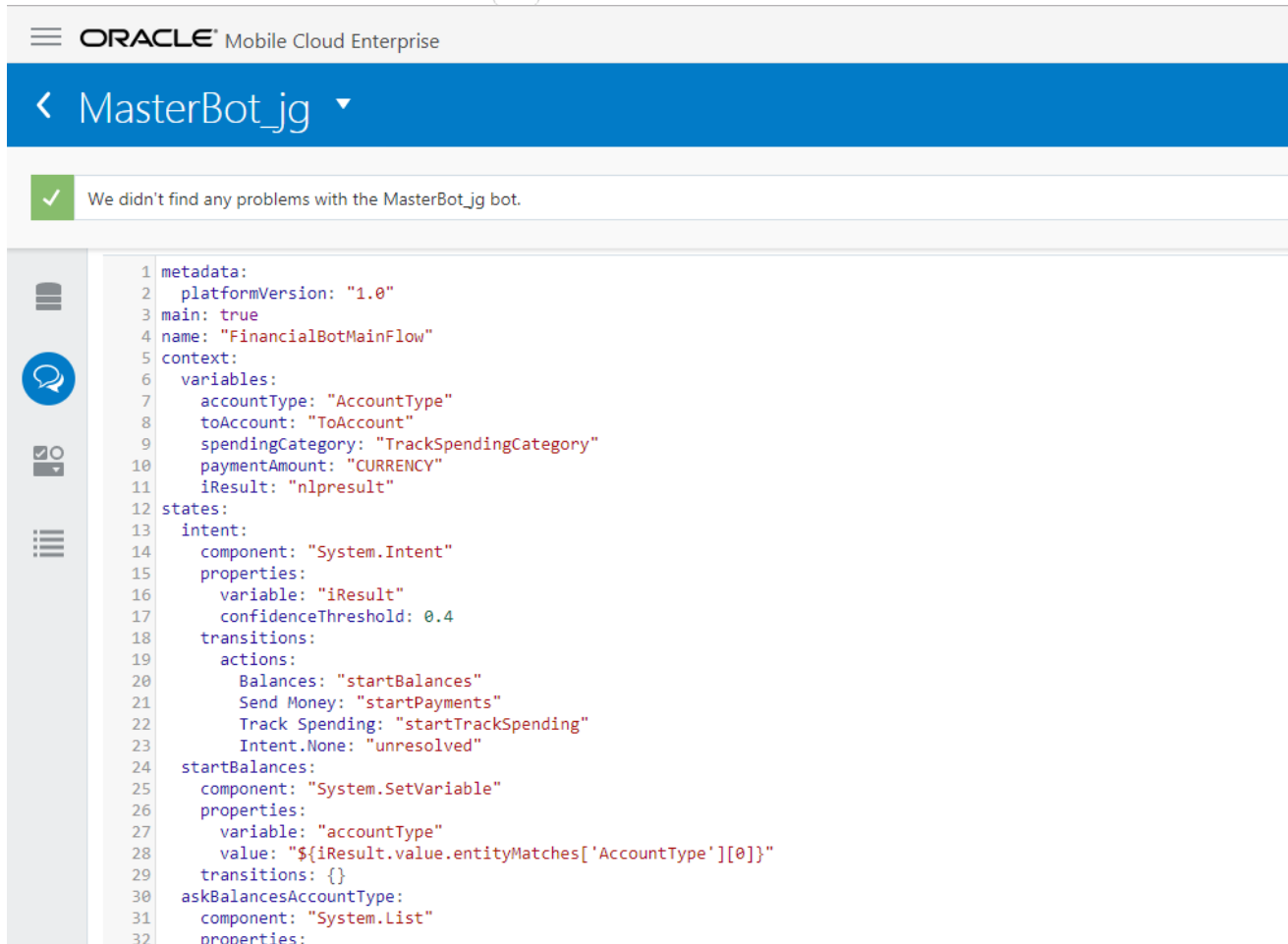
return: "showSpending"



# Oracle MOOC: Oracle Intelligent Bots

12. Now that we've examined all of the flow code and how the custom components fit in, it's your turn to update the BotML definition in your chatbot:

- Open the CustomMasterBotYAML.txt file and copy all of its contents into the editor, replacing any existing code.
- Click the **Validate** button.



```

1 metadata:
2   platformVersion: "1.0"
3 main: true
4 name: "FinancialBotMainFlow"
5 context:
6   variables:
7     accountType: "AccountType"
8     toAccount: "ToAccount"
9     spendingCategory: "TrackSpendingCategory"
10    paymentAmount: "CURRENCY"
11    iResult: "nlpresult"
12 states:
13   intent:
14     component: "System.Intent"
15     properties:
16       variable: "iResult"
17       confidenceThreshold: 0.4
18     transitions:
19       actions:
20         Balances: "startBalances"
21         Send Money: "startPayments"
22         Track Spending: "startTrackSpending"
23         Intent.None: "unresolved"
24   startBalances:
25     component: "System.SetVariable"
26     properties:
27       variable: "accountType"
28       value: "${iResult.value.entityMatches['AccountType']}[0]}"
29     transitions: {}
30   askBalancesAccountType:
31     component: "System.List"
32     properties:
  
```

In the next step, you're going to test your chatbot to see how the custom components pull data from OMCE APIs.

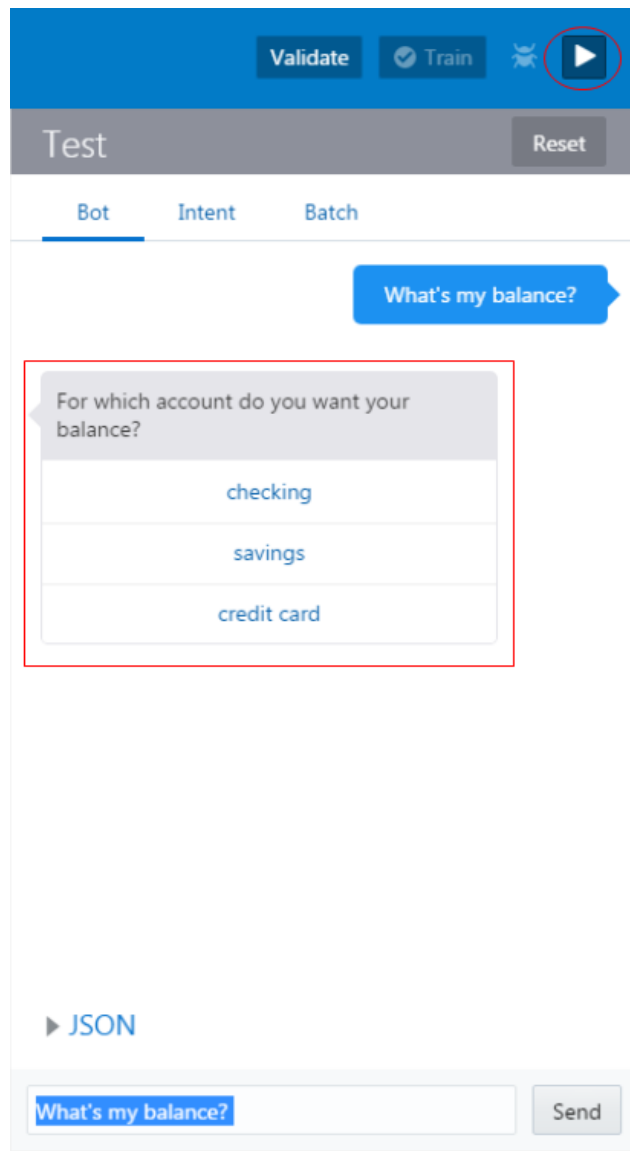
## Step 3: Test the Chatbot with the Custom Components

In this section, you will run your chatbot to see how custom components make a difference in what's returned. So let's start testing. We'll begin with the Balances intent.



# Oracle MOOC: Oracle Intelligent Bots

1. To test the flow code that uses custom components, click the **Play** button in the upper right to open the Tester. If it's already open, click **Reset** to start a new session.
2. Next, enter *What's my balance?* in the Message area and then click **Send**. You should see a list of all the accounts you've included in the `System.List` component.



3. Next, select an account. The balance appears, which is a value returned by the `BalanceRetrieval` custom component.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

What's my balance?

For which account do you want your balance?

checking
savings
credit card

savings

The balance in your savings account (258293-832) is \$2610.56

► JSON

What's my balance? Send

4. Try out some other messages, including some with the account in the message text (like *What's my credit card balance?*) to see how the chatbot responds.
5. Now let's test the flow of the Send Money intent.
  - a. First, click the **Reset** button.
  - b. Next, enter *Send a payment* and then click **Send**.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

Send a payment Send

- c. When prompted, either type the account to send the money from, or select it from the list.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

From which account do you want to make a payment?

checking
savings
credit card

checking

To which account do you want to make a payment?

Lauren
Shea
Mom
Chase Preferred
the baby sitter

Send a payment Send

- d. Next, you're prompted for a person to send the money to. Select a person to receive the money.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

From which account do you want to make a payment?

checking
savings
credit card

checking

To which account do you want to make a payment?

Lauren
Shea
Mom
Chase Preferred
the baby sitter

Send a payment Send

- e. Finally, you are prompted for an amount to send. Enter \$50 and click **Send**.

The chatbot confirms the recipient, amount, and the account that the funds were drawn from.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

To which account do you want to make a payment?

Lauren
Shea
Mom
Chase Preferred
the baby sitter

Shea

What's the payment amount?

\$50

Your payment of 50.0 dollar to Shea has been made from checking.

► JSON

\$50 Send

- Now let's test the flow of the Track Spending intent, so click **Reset**.
- You need to specify a spending category for this intent, so start off by entering *How much did I spend on gas?* and then click **Send**.

# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot

Intent

Batch

How much did I spend on gas?

You've spent \$53.25 on gas

► JSON

How much did I spend on gas?

Send

- Now click **Reset** and try another: enter *How much did I spend on travel?* and then click **Send**.



# Oracle MOOC: Oracle Intelligent Bots

Test Reset

Bot Intent Batch

How much did I spend on travel?

You've spent \$103.28 on travel

► JSON

How much did I spend on travel? Send

Congratulations! You just completed this lab. Next up, you'll integrate your chatbot into Facebook Messenger.