## Integrate Custom Services Hosted on OMCe

In this lab, you connect your chatbot to the Mobile Core of OMCe (Oracle Mobile Cloud, Enterprise) and consume some custom components. At this point, you have the skeleton of a working chatbot – it recognizes the 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 isn't any business logic being executed.

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 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

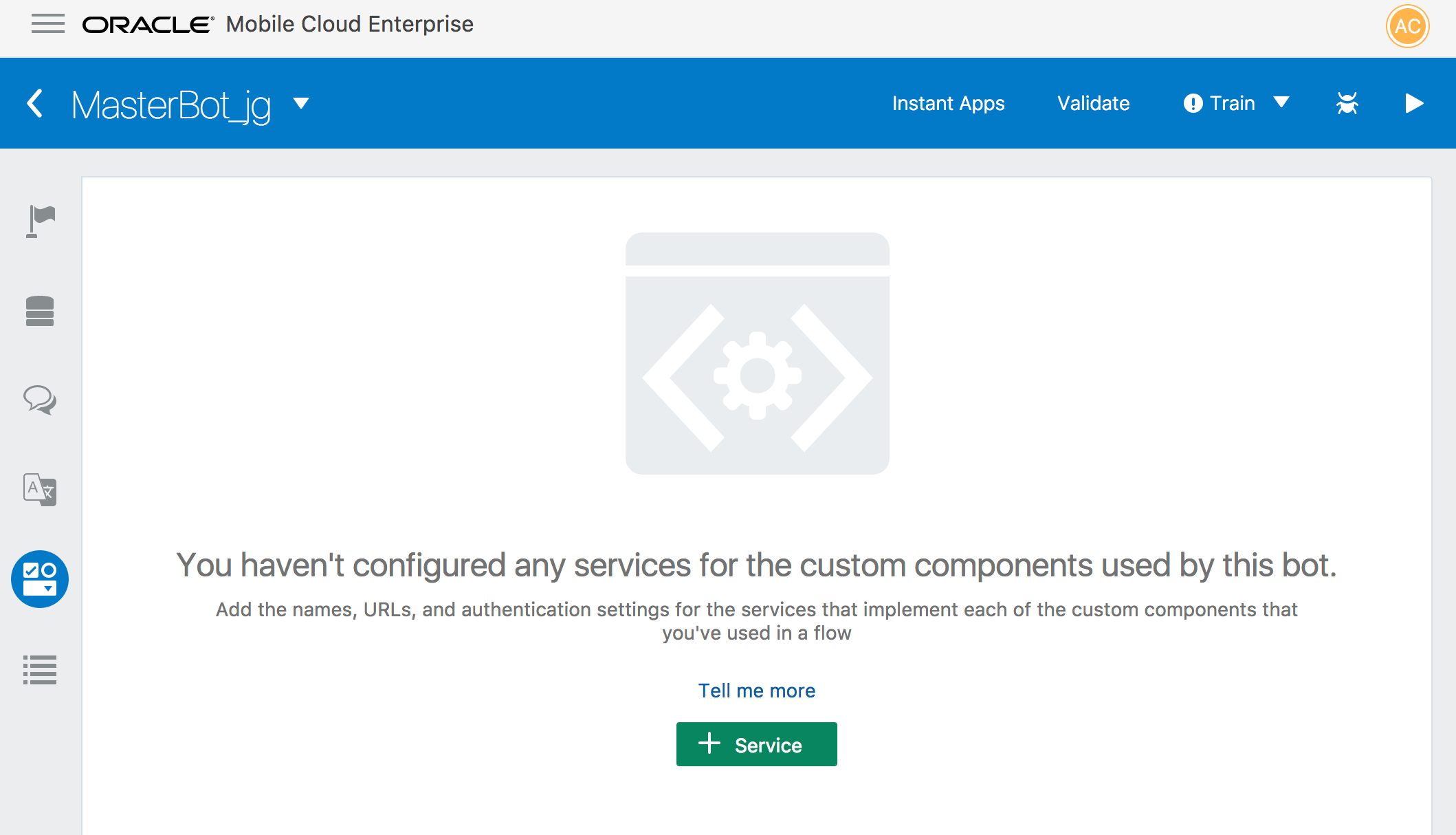
Have the following files from the labfiles.zip close at hand. You can find all of them in the labfiles/code directory:

* MCSServices.txt
* CustomPrintBalance.txt
* CustomStartPayments.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 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\_firstNameLastName and select the Components icon from the left navbar.
2. Your page should show there are currently no component services configured for your chatbot.



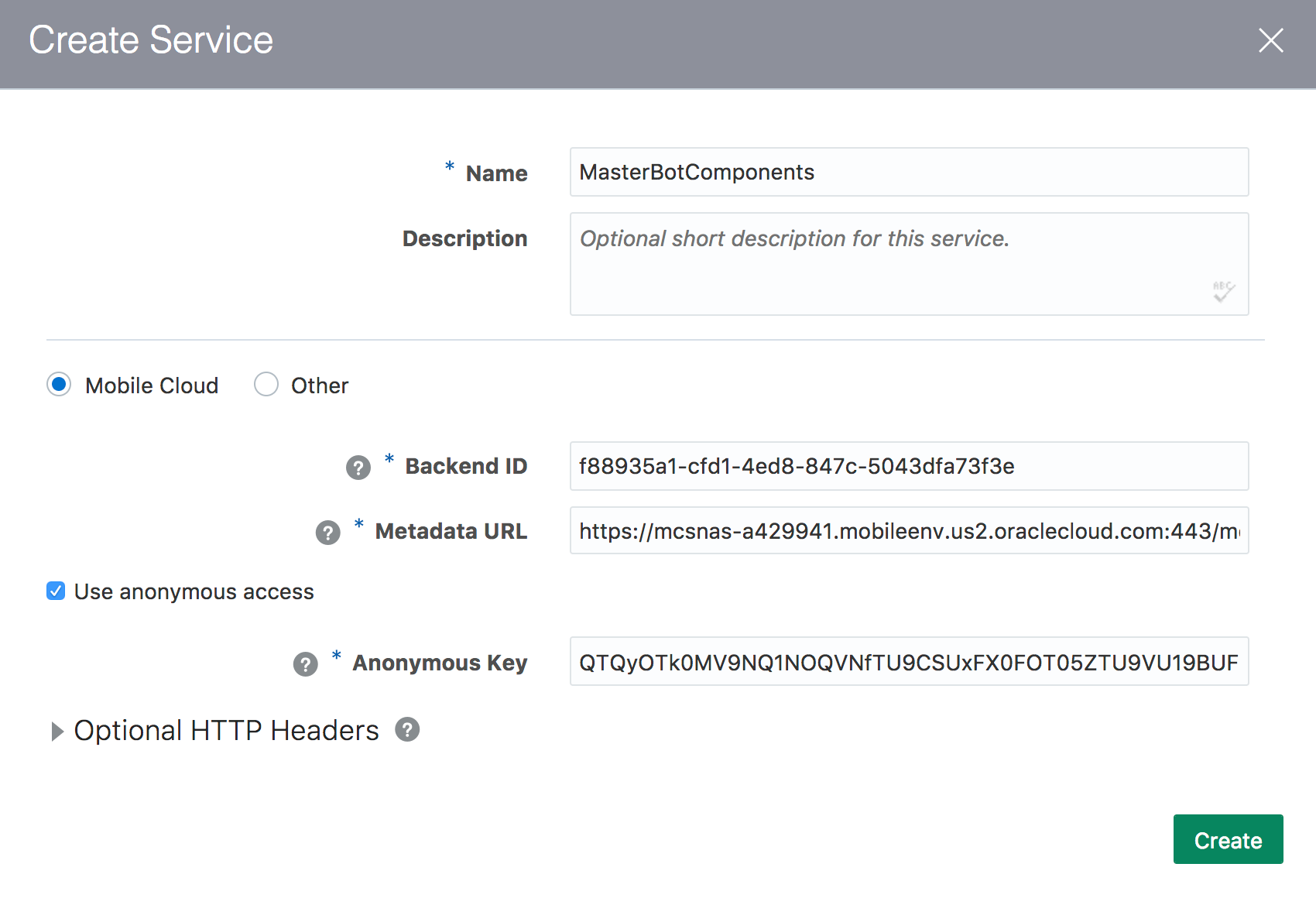
1. To connect to the OMCe backend, we need some information from both the backend and the APIs that it accesses.

.

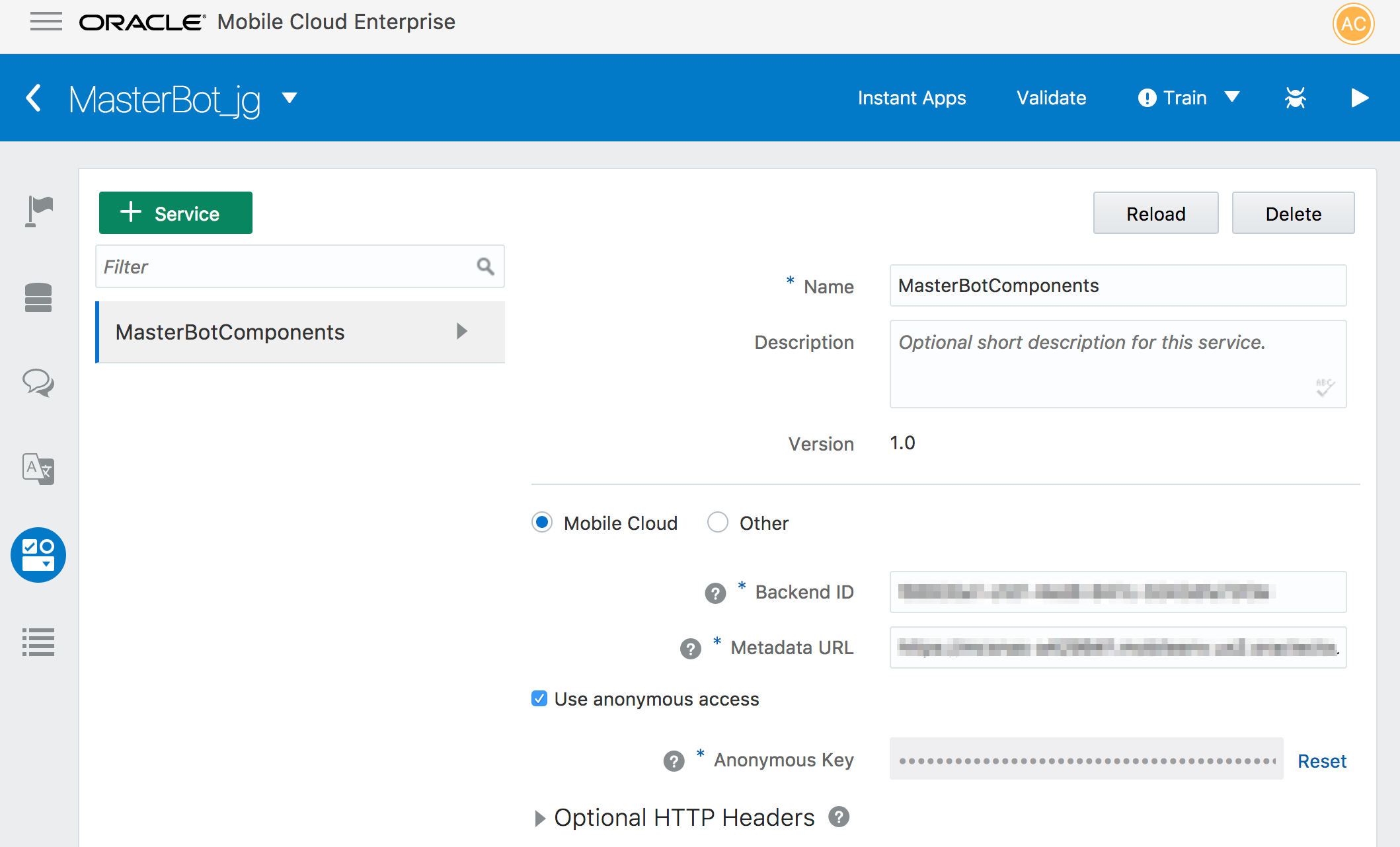
To complete this exercise, you don’t have to navigate to Mobile Core to look at the API and the backend. All of the information that you need is in the MCSServices.txt file, which is located in the labfiles/code directory.

To configure a component service using these values:

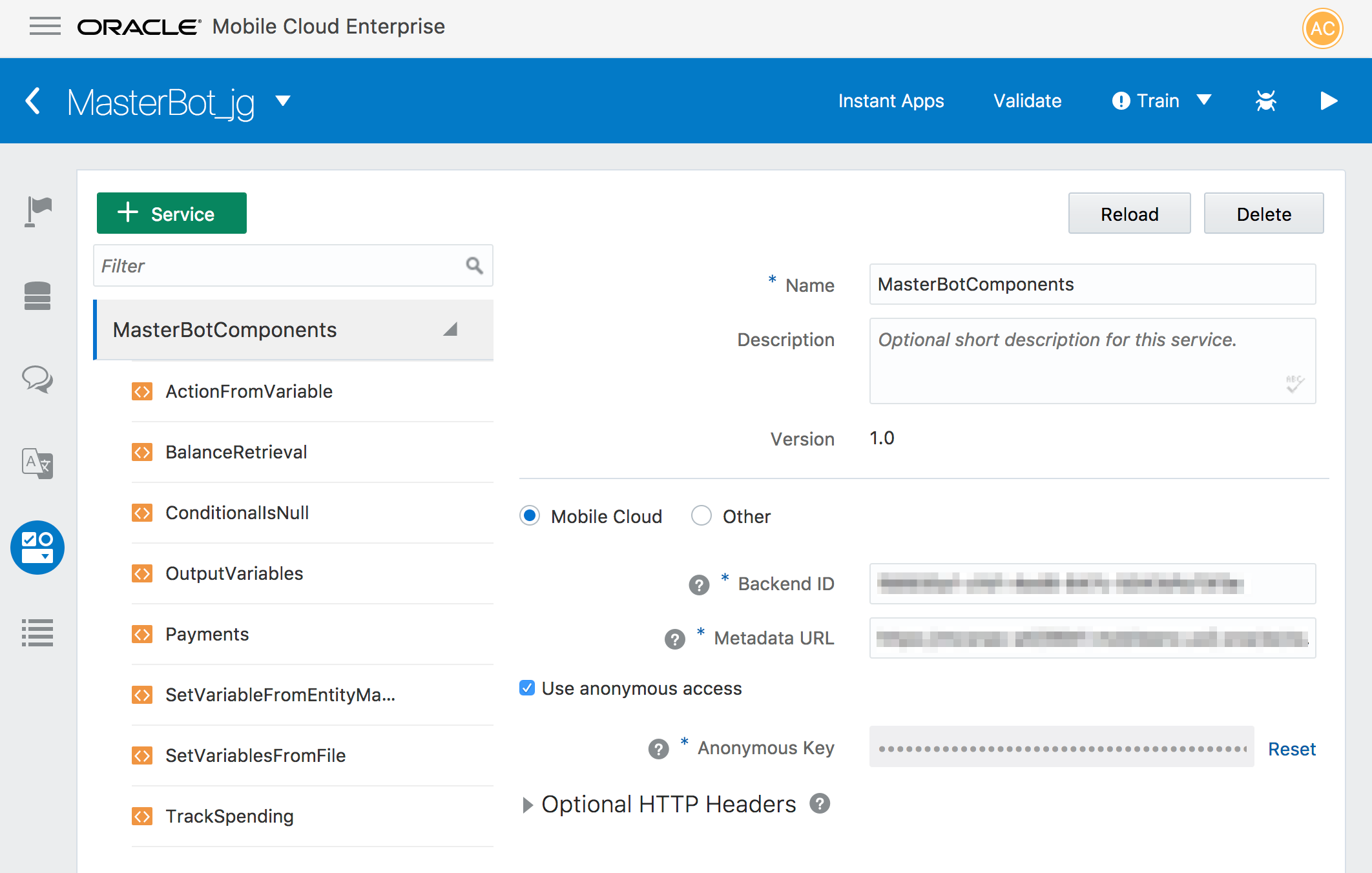
1. Returning to Oracle Intelligent Bots and the MasterBot, click the green **Service** button to configure a new component service to access the backend.
2. Enter MasterBotComponents as the service name.
3. Copy the for Backend ID and Metadata URL from the MCSServices.txt file and paste them into the Create Service window.
4. Click **Use anonymous access**.
5. Copy and paste the Anonymous Key value from the MCSServices.txt file.



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



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



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

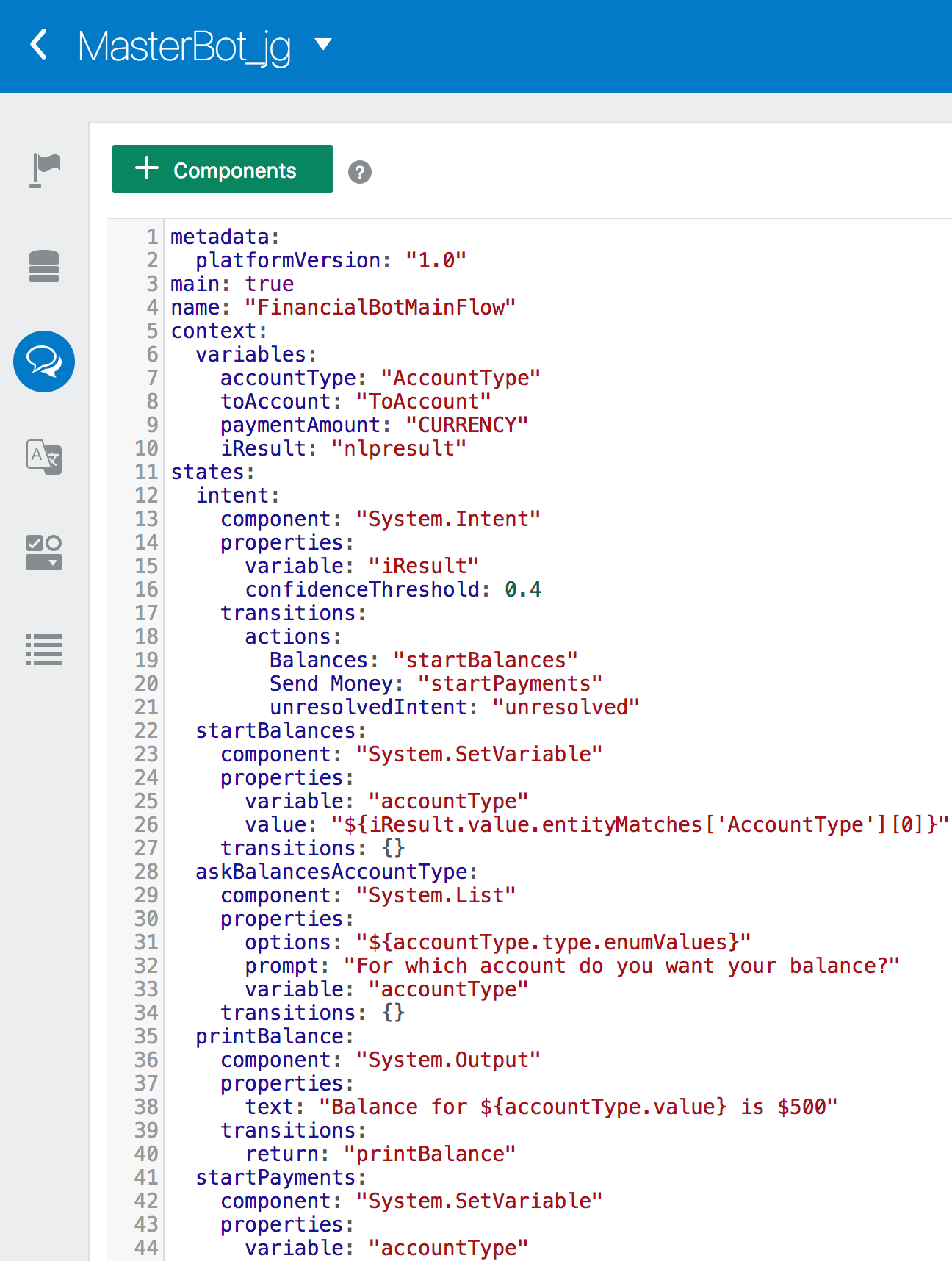
Now that you have consumed some pre-built components, let's use them in your BotML code.

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

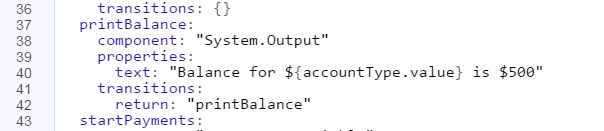
In this section, you will replace some of the BotML code to now access the custom components. These components will then use the APIs associated with the backend to return data from OMCe. For the first steps, you just examine the custom component code and see how it differs from the exiting BotML definition.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

1. Click the Flow 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.

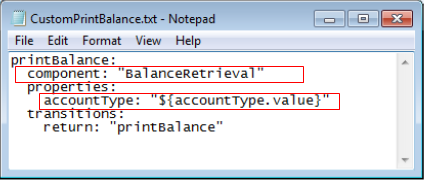


1. First, let's work with the Balances intent. Here is what you should see in the printBalance state.



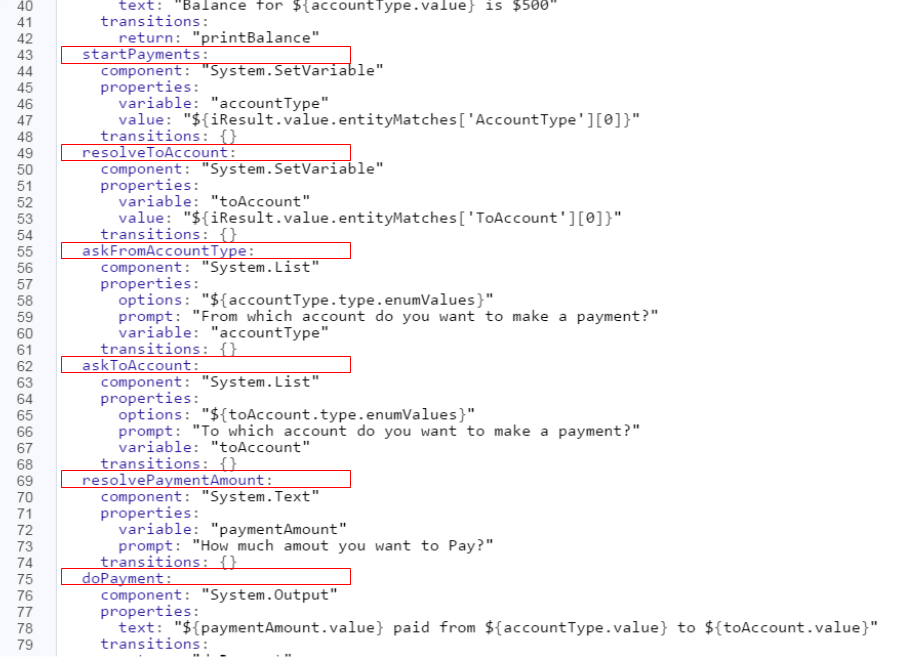
1. Using the CustomPrintBalance.txt file that’s located in the labfiles/code directory, make the following two changes to the printBalance state to update it from using a system component to the custom component, BalanceRetrieval. There are two changes that you need to make:
   1. First, change the component 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.



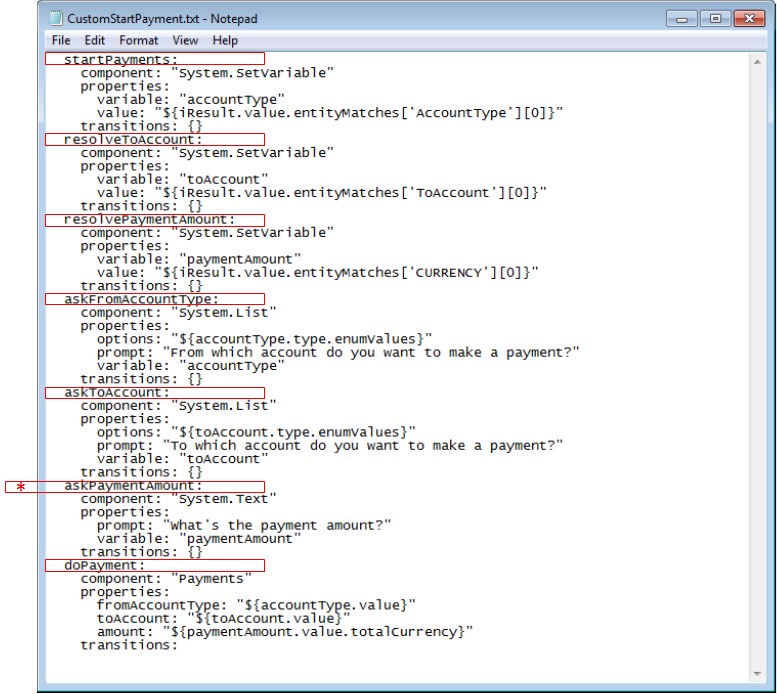
1. Next, let's work with the Send Money intent. This one is a bit more complex because it requires adding some new states. We'll start off by explaining what is happening and then you'll replace all the states for that intent with code from a file.

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



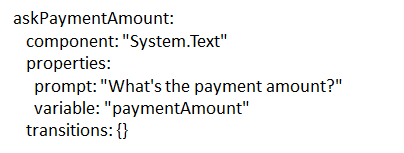
1. The CustomStartPayment.txt file contains all the new code. Open the file and take a look at it.

It contains all of the six states that currently exist in your chatbot, and adds a new one: askPaymentAmount.



1. Now let's look at how the new state operate. It does not use any custom components, so there's really nothing new or different from what we've used before.

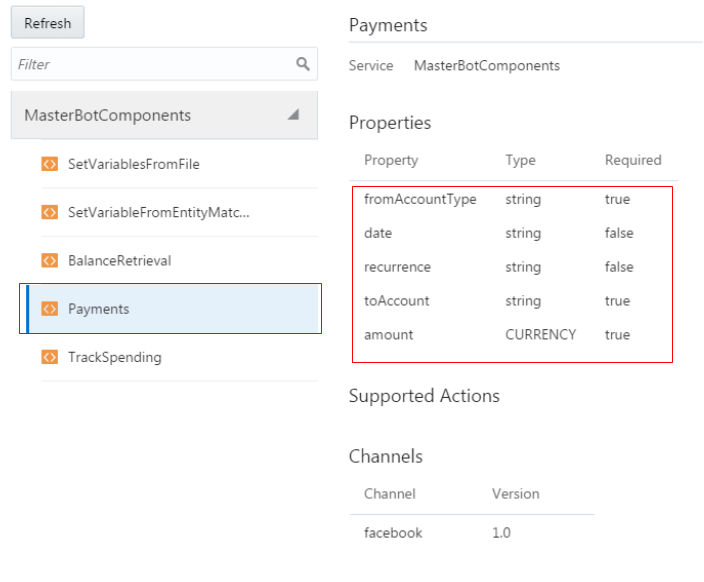
The askPaymentAmount, uses a System.Text component to prompt the user for an amount and a variable to store the payment amount.

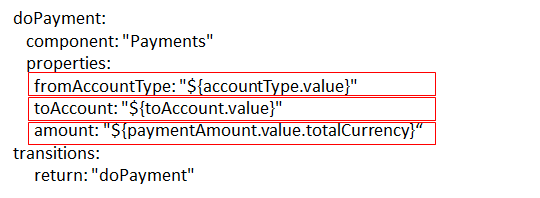


Likewise, the startPayments, resolveToAccount, askFromAccountType and askAccountType states are all the same, so there's no need to examine them.

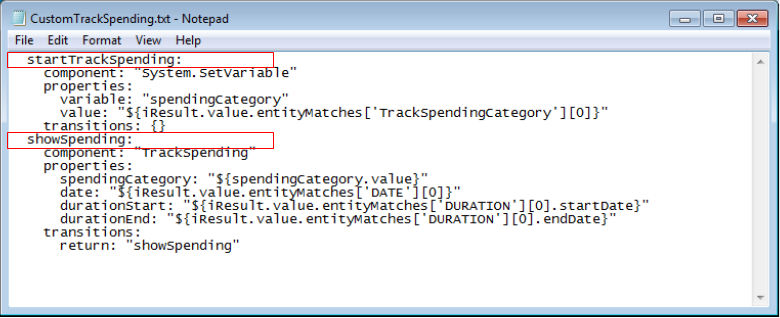
However, the doPayment state is different. It uses the Payments custom component.

1. 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.



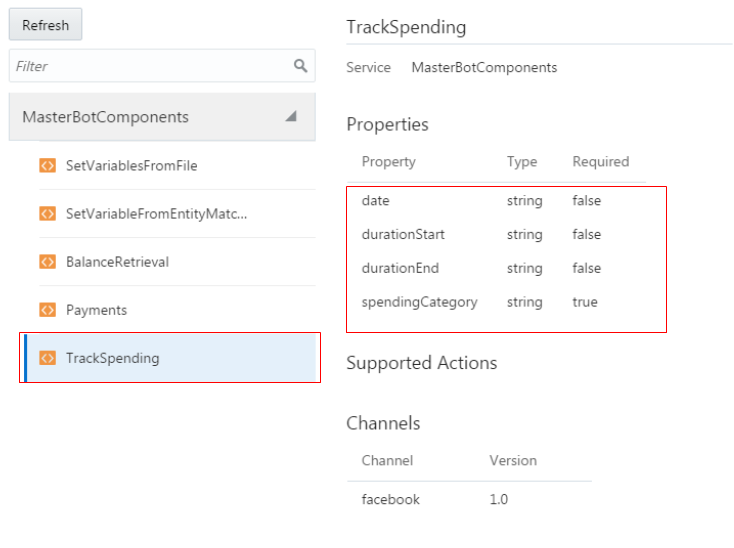
1. Looking at the doPayment state definition, you can see that the fromAccountType, toAccount, and paymentAmount properties are all set.
2. There’s one additional thing to note: In the variables node at the top of the BotML definition, paymentAmount is set to a system entity, CURRENCY. 
3. Finally, let's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

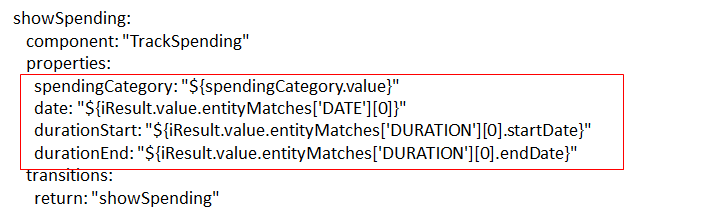


1. Let's look at how these two states operate.

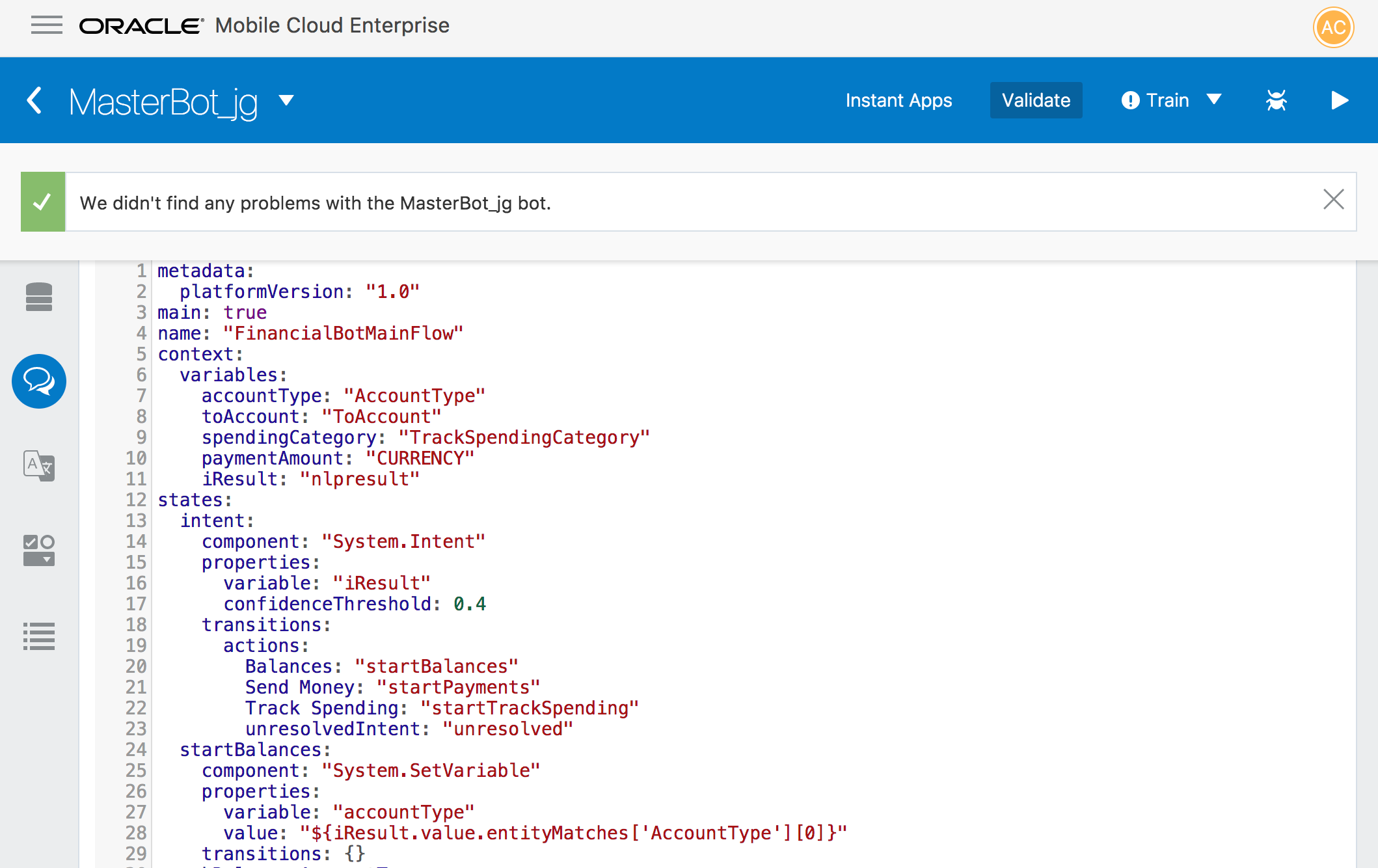
The startTrackSpending uses a System.SetVariable component and a spendingCategory variable. The showSpending state uses the TrackSpending custom component (shown in the Components page, below).



This component sets all for variables.



1. 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:
2. Open the CustomMasterBotYAML.txt file and copy all of its contents into the editor, replacing any existing code.
3. Click the **Validate** button.

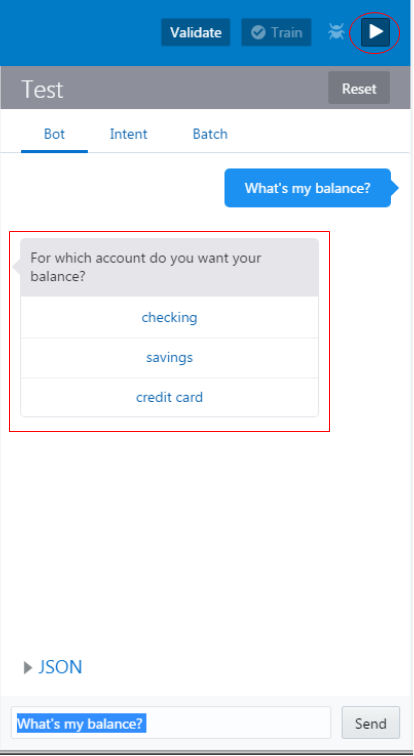


In the next step, you’re going to test your chatbot to see how the custom components pull data from OMCe APIs.

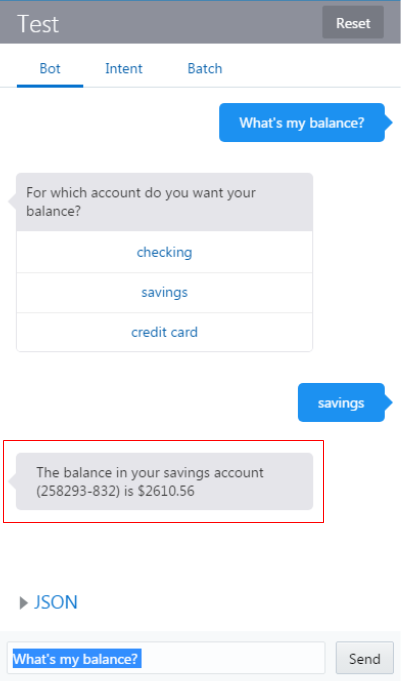
### Step 3: Test the Chatbot with the Custom Components

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.

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.



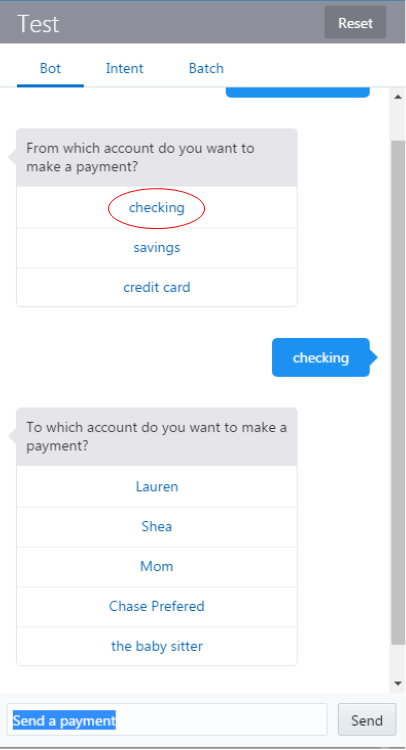
1. Next, select an account. The balance appears, which is a value returned by the BalanceRetrieval custom component.



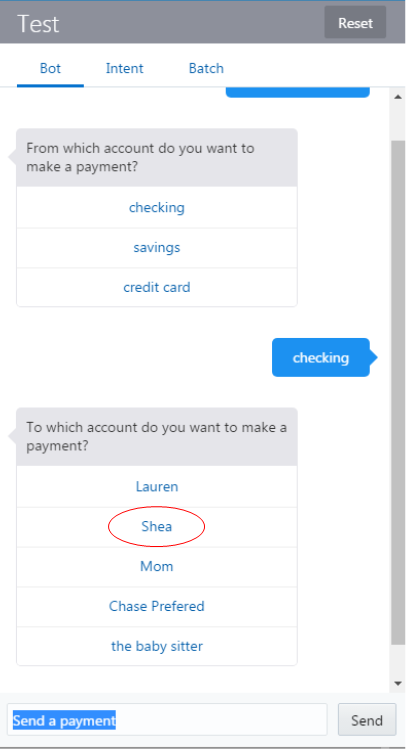
1. Try out some other messages, including some with the account in the message text to see how the chatbot responds.
2. Now let's test the flow of the Send Money intent.
3. First, click the **Reset** button.
4. Next, enter *Send a payment* and then click **Send**.



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

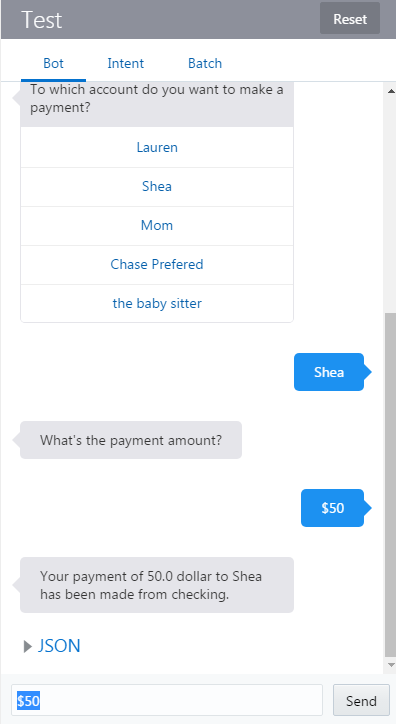


1. Next, you're prompted for a person to send the money to. Select a person to receive the money.

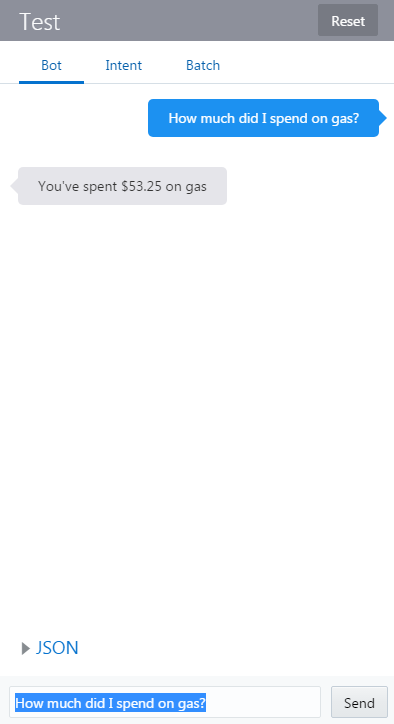


1. Finally, you are prompted for an amount to send. Enter $50 and click **Send**.

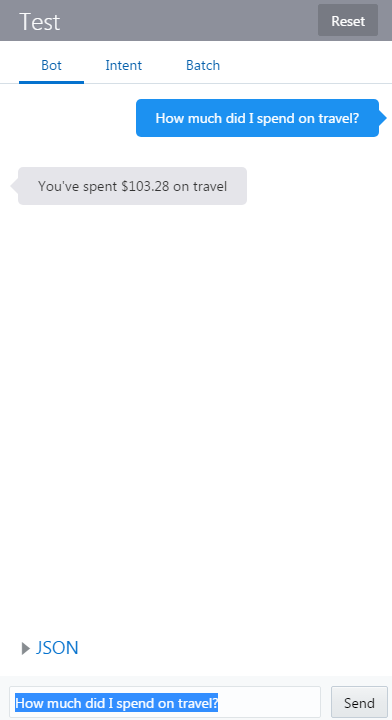
The chatbot confirms the recipient, amount, and the account that the funds were drawn from.



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



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



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