How to respond to user inactivity using the Oracle Web SDK messenger: An implementation strategy

Frank Nimphius, April 20

Updated for oda-native-client-sdk-js-20.5.1 and later, May 2020

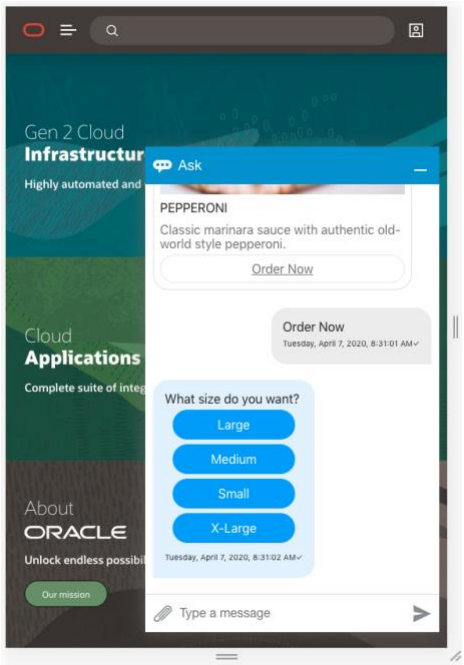
During a bot conversation, users can be distracted for a short time and become inactive. With the Oracle Web SDK, your skills can remind users after a certain idle time or even cancel the conversation. This behavior however does not come out-of-the box but requires you to design your dialog flow conversation for it.

This article explains an implementation strategy that allows bot to remind users of the conversation they started and also to close a conversation that sat idle for too long. Using this implementation you can also avoid users from staying idle beyond the time configured as the channel session expiry time.

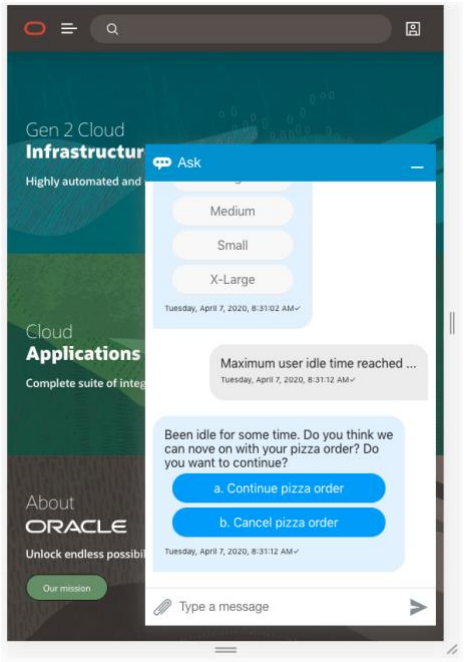
IF YOU ARE NEW TO THE ORACLE WEB SDK, ENSURE YOU TRIED THE FOLLOWING TUTORIAL FIRST
HTTPS://DOCS.ORACLE.COM/EN/CLOUD/PAAS/DIGITAL-ASSISTANT/TUTORIAL-WEB-SDK/INDEX.HTML

The Sample At Runtime

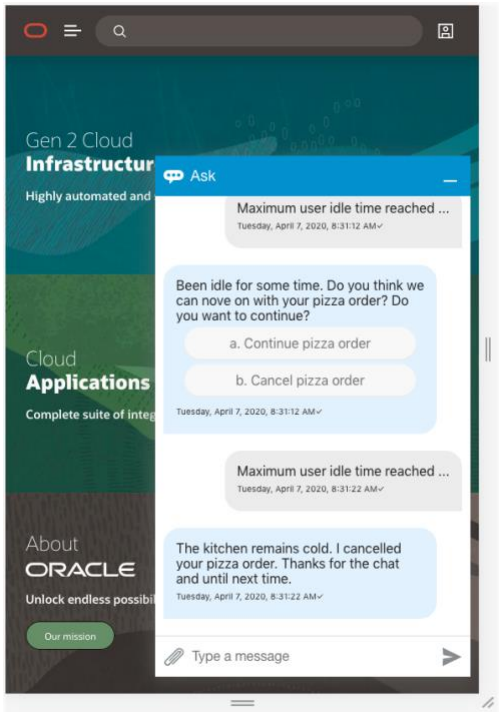
For this article, there exists a sample skill and web messenger for you to download from **here**. After setting up the pizza order sample, you see a user interface like shown below.



When pausing the interaction with the bot, the timer, which for demo purposes is set to 10 seconds, upon expiry displays a message to the user and sends a postback action to the bot.



The image above shows the bot response for when the user remained inactive for too long. As you can see, the user is asked to continue the conversation to where he or she left it or to cancel the interaction. If still then the user remains inactive, then after another 10 seconds (which is configurable) the bot discontinues the conversation (image below).



Skill Setup And Configuration

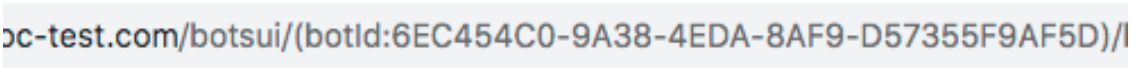
To run this sample, you first need to import the skill from the **downloaded sample**. Unzip the downloaded zip file and import the skill it contains to your Oracle Digital Assistant instance.

IF YOU DON'T HAVE AN ORACLE DIGITAL ASSISTANT, FOLLOW THIS ARTICLE TO START A TRIAL.

TECHEXCHANGE: GETTING STARTED WITH ORACLE DIGITAL ASSISTANT ON ORACLE CLOUD INFRASTRUCTURE (OCI)

Next, open the skill and choose the **cone** icon to access the skill settings. In the skill settings, select the **Configuration** tab and scroll down to the skill parameter section. There are two configuration parameters in this sample. Only the **botId** parameter needs to be edited with the ID the bot has in your Oracle Digital Assistant instance.

To edit the **botId** configuration, select the parameter and press the **Edit** link. The bot ID of the sample skill is displayed as part of the URL. Just copy it from there.



What happens if you forget to update the botId? If you don't update the bot Id, or if you have a typo in the ID you added, then at runtime the web messenger will error with a "Ooops" message.

Understanding The Skill Part Of The Solution

In summary, the `System.CommonResponse` component is required and should become your new best friend. To handle user idle time, the following needs to happen

- You must define the following dialog flow variable: **lastState: "string"** . The variable receives the dialog flow state name to navigate to after the user idle time exceeds in case that the user wants to continue the conversation.
- The bot response needs to contain information about the state name that rendered the response. This information is mandatory if you want to handle the idle time such that users can continue working with the bot after receiving a "wake up" call.
- A "hidden" state (a state that you normally would not navigate to) is needed to display a dialog to the user when the configured tolerated idle time expires. In the sample, the tolerated idle time is set on the messenger (Web SDK). However, with only little code changes, you can pass the idle time as a channel property to the client script. The hidden dialog flow state name in the example is "hiddenPostbackState".
- All bot responses that should reset the timer must use the System.CommonResponse component. On the common response component you use the **channelCustomProperties** property to send information about the current state and the go to state to the client (see line 270 in the image below)

```
257 #SHOW SUMMARY WITH TIMER SETTING
258   showSummary:
259     component: "System.CommonResponse"
260     properties:
261       processUserMessage: true
262       keepTurn: false
263       metadata:
264         responseItems:
265         - type: "text"
266           text: "Your order summary is here. We ship when you submit!"
267         - type: "text"
268
269           #IMPORTANT! TIMER SETTINGS
270           channelCustomProperties:
271           - channel: "websdk"
272             properties:
273               lastState: "showSummary"
274               botId: "${system.config.botId}"
275               goToState: "${system.config.goToState}"
276
```

In the image above, when the order summary is displayed to the user, and if the channel type is *websdk* (Oracle Web SDK), the following custom properties are added: lastState, botId, goToState.

- **lastState** – is the name of the dialog flow state that renders the response. "showSummary" in the image above
- **botId** – is the ID of the bot instance that sends the response message. The bot Id is important because in a digital assistant, the channel needs to know where to route the request to. In the sample, the botId is configured as a skill parameter.
- **goToState** – is the state that handles the user idle expiry.

Note: If you change one of the parameter names without changing the client side script, then you are breaking the solution from working

```
305   hiddenPostbackState:
306     component: "System.CommonResponse"
307     properties:
308       processUserMessage: true
309       keepTurn: false
310       metadata:
311         responseItems:
312         - type: "text"
313           #display a random prompt to be less robotic
314           text: "${messages.value.orderReminderMessage[.now?long % mes
315
316           #IMPORTANT! TIMER SETTINGS. IF USER REMAINS IDLE
317           #CLOSE THE SESSION
318           channelCustomProperties:
319           - channel: "websdk"
320             properties:
321               lastState: "hiddenPostbackState"
322               botId: "${system.config.botId}"
323               goToState: "cancelOrder"
324
325           actions:
326           - label: "a. Continue pizza order"
327             type: "postback"
328             keyword: "a,A,ok"
329             payload:
330               action: "continueWithOrder"
331               variables:
332                 lastState: "orderPizza"
333           - label: "b. Cancel pizza order"
334             type: "postback"
335             keyword: "b,B"
336             payload:
337               action: "cancelOrder"
338     transitions:
339       #force user to make a valid choice. Otherwise
340       #repeat this dialog flow state state
341       next: "hiddenPostbackState"
342       actions:
343         continueWithOrder: "continueWithOrder"
344         cancelOrder: "cancelOrder"
345
```

Finally, there is one more consideration to do, which is to decide how you want it to be handled when the user doesn't respond to the "wake up" dialog and still remains idle. In the sample, the user is given another 10 seconds (again, the time is configurable and in a production environment should be set to several minutes but still less than the expiry time set on the channel.

The image above shows the **hiddenPostbackState** that renders the "wake up" dialog. Notice in line 316 how this dialog flow state too renders a custom channel properties. This time however the **goToState** setting is not to the hiddenPostbackState but to the **cancelOrder** state. This way, if the user remains inactive the bot conversation is cancelled gracefully before the user session expires.

If the user responds positive to the "wake up" dialog in that he or she wants to continue working with the bot, then navigation goes to the **continueWithOrder** state. Notice how the state checks for the **lastState** variable to determine the dialog flow navigation (line 349, and 355).

```
346   continueWithOrder:
347     component: "System.Switch"
348     properties:
349       source: "${lastState.value?has_content?then('navigate','nonavigate')}"
350       values:
351       - "navigate"
352       - "nonavigate"
353     transitions:
354       actions:
355         navigate: "${lastState.value}"
356         nonavigate: "somethingGoneWrongState"
357         NONE: "somethingGoneWrongState"
```

Web SDK Configuration

The Web SDK is provided with the sample. In the extracted sample, navigate to the **WebSDKSample/scripts** folder. Open the **settings.js** file and edit the Oracle Digital Assistant host URI and the channel ID of the Oracle Web channel you created.

SEE THE FOLLOWING DOCUMENT TO LEARN ABOUT HOW TO CONFIGURE THE ORACLE WEB CHANNEL

HTTPS://DOCS.CLOUD.ORACLE.COM/EN-US/IAAS/DIGITAL-ASSISTANT/DOC/ORACLE-WEB.HTML#GUID-9ECA8116-A95B-4850-8D1C-8F9BB845EE4F

```
24  } else {
25      chatWidgetSettings = {
26          URI: '<oda-uri>,
27          channelId: '<channel-id>'
28      };
29  }
```

The **oda-uri** value you need to provide is just the name of your Oracle Digital Assistant instance host. **Don't** provide the leading https:// or a training slash. For example oda-<your host specifics>.com

The **channel-id** is the ID of the Oracle Web channel you need to create and that you need to associate with the imported sample skill. Ensure you enabled the channel and that you **disabled** client authentication.

Running the Sample

After you completed the configuration, on your file system, navigate to the **WebSDKSample** folder in the extracted download. Double click on to the **index.html** file to launch it in the browser. Click on the bot icon on the lower right to start the Web messenger.

If you correctly configured the **settings.js** file and if you set up the Oracle Web channel accordingly, then you can start the bot conversation with "I like to order pizza". This then starts the pizza order dialog. If, anywhere in the conversation, you pause for more than 10 seconds, you will see the "wake up" dialog and, if you don't respond in time, then the cancel confirmation. If you use the dialog to continue the conversation then you should notice that the conversation is brought back to where it was left.

What You Should Know About Index.html

The client logic is in the index.html file. The image below shows the **Bots.connect()** function call. In **line 80** you see the configuration of a delegate object that gets called whenever the bot returns a message to the messenger.

The JavaScript code in **line 83** clears the time if one exists to the time. If the **lastState** message extension could be found in the bot message (just a reminder that this extension is added in the System.CommonResponse component), then a function is called to set a timer. If the user does not respond until the timer expires, in which case the time is cleared, the "wake up" dialog will be requested.

```
79  Bots.connect().then(() => {
80      Bots.setDelegate({
81          beforeDisplay(message) {
82              //ANY RESPONSE RESETS THE TIMER
83              resetTimer();
84              if (message.messagePayload.channelExtensions) {
85                  if (message.messagePayload.channelExtensions.lastState) {
86                      //ONLY RESPONSES WITH A LAST STATE PROPERTY SET THE TIMER
87                      setTimer(message.messagePayload.channelExtensions.lastState,
88                               message.messagePayload.channelExtensions.botId,
89                               message.messagePayload.channelExtensions.goToState);
90                  }
91              }
92
93              return message;
94          }
95      })
96  });
```

The images below show the **resetTimer** function as well as the **setTimer** function.

```
24      <script>
25          var timer = null;
26
27          function resetTimer(){
28              if(timer != null){
29                  clearTimeout(timer);
30              }
31              return;
32          }
33
```

The **setTimer** function defines a time out (line 58) and a function to call (line 42 and following).

```
34      function setTimer(lastState, botId,goToState) {
35
36          var _lastState = lastState;
37          var _botId = botId;
38          var _goToState = goToState;
39          //console.log("...."+_lastState+"....."+_botId+"......"+_goToState);
40          timer = setTimeout(() => {
41                  //send message after max. idle time inactivity
42                  Bots.sendMessage({
43                      "postback": {
44                          //communicate to bot what the state is to
45                          //continue the conversation with. This is
46                          //usually the state that was active when
47                          //the timer expired
48                          "variables": {"lastState": _lastState},
49                          "system.botId": _botId,
50                          "action": "",
51                          "system.state": _goToState
52                      },
53                      "text": "Maximum user idle time reached ...",
54                      "type": "postback"
55                  });
56              },
57              //set to 10 seconds of idle time
58              10 * 1000
59          );
60      }
61  </script>
```

Notice how in 42 the JavaScript function sends a postback call to the bot. The payload of this call contains the value to set the lastState variable in the bot to (line 48) and the name of the dialog state to go to (line 51) and the botId (line 49). The name of an action is not require as you don't need to execute an action for the use case at hand.

Don't Forget

Line 58 in the image above set the timer to expire after 10 seconds. When using this implementation in a productive environment you want to set the idle time out to several minutes but below the time out set to the web channel (which is 60 minutes).

Note that the timeout setting in line 58 is set in seconds.

Running the Sample

To run the sample, start the web messenger with a double click on the index.html file and then open the chat window by clicking on the bot icon. Start your conversation with *I want to order a pizza* and pause for 10 seconds. See what happens.