Action Bot

Owner:

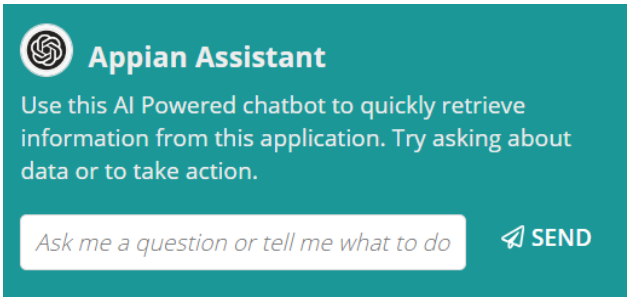
Drew Hamrock (drew.hamrock@appian.com) *Last updated - 4/12/24*

Purpose:

Assist the user in quick actions or questions around their current work. Instead of navigating to a specific record to find information or perform a related action, do this instead from this bot on the dashboard. This component uses GPT3.5 Turbo (default) and GPT4 through Azure. Click the Bot title to switch configurations. To configure the parameters of this bot, continue to configurations.

Action Prompts:

There are two types of actions: Record actions and related actions. Record and related actions play nice out of the box, but you can customize further to have additional actions based on the way I set triggers for the output response. They do not have to be record/related actions specifically, but those are easier to set up and allow for the dialogue box.



Your prompt does not need to be exactly what your prompt data reflects – it should understand what you want. That being said I wouldn't try to confuse it too much.

Questions:

If you would like to ask the bot a question, you can do that as well. The bot has been trained on data using the same method as in the ChatGPT Data Bot utility. Adding a "?" at the end of your question will ensure that your bot knows how to respond, but this isn't required – just good to know. This bot does not have memory of your conversation, so don't build prompts off one another.

Configurations:

To begin setup navigate to CBB_configurations. You will see details on how to set configurations for the bot. Use these as presets for the users depending on their roles. Here are examples of configuring inputs.

CBB_configurations

```
56    value: ri!configurationId,
57    default: {
58      turnOffActions: false,
59      shape: "SEMI_ROUNDED",
60      chatColor: "#041E42",
61      promptData: rule!CBB_flexQuery(recordType: recordType!CBB Prompt),
62      recordType: recordType!WN Event,
63      recordActionList: {
64        recordType!WN Event.actions.newEvent,
65        recordType!WN alert.actions.newAlert,
66        recordType!WN absenceReport.actions.newAbsenceReport,
67        recordType!WN Employee.actions.newEmployee
68      },
69      relatedActionList: {
70        recordType!WN Employee.actions.updateEmployee,
71        recordType!WN Event.actions.updateEvent,
72        recordType!WN alert.actions.updateAlert,
73        recordType!WN Operation.actions.updateOperation,
74        recordType!WN Location.actions.updateLocation
75      },
76      relatedActionData: {
77        rule!CBB_flexQuery(
78          recordType: recordType!WN Employee,
79          fields: {
80            WN Employee.id,
81            WN Employee.name
82          },
83          batchSize: 200,
84        ),
85        rule!CBB_flexQuery(↔),
93        rule!CBB_flexQuery(↔),
101       rule!CBB_flexQuery(↔),
109       rule!CBB_flexQuery(↔)
117     },
118   },
```

CBB_configurations

```
195 ▸   then: {↔},
277     equals: 3,
278 ▾   then: {
279       turnOffActions: false,
280 ▾     promptData: {
281 ▾       tostring(
282 ▾         {
283 ▾           a!map(id: 1, prompt: "Create new event", response: 1),
284 ▾           a!map(id: 2, prompt: "Update event", response: 2)
285           }
286         )
287       },
288       recordType: recordType!WN Employee,
289 ▾     fields: {
290         WN Employee.wnMessage
291       },
292 ▾     filters: {
293 ▾       a!queryFilter(
294           field: WN Employee.id,
295           operator: ">",
296           value: 50
297         )
298       },
299 ▾     recordActionList: {
300         recordType!WN Event.actions.newEvent
301       },
302 ▾     relatedActionList: {
303         recordType!WN Event.actions.updateEvent,
304       },
305
306 ▾     relatedActionData: {
307
308 ▸       rule!CBB_flexQuery(↔),
```

Configuration Parameters:

rule!CBB_ActionBot(configurationId)

The configurationId will point to your CBB_configurations expression rule with the following parameters as presets.

displayTitle (Text): The title to be displayed above the chatbot. Default "Appian Assistant".

promptData (Any Type): The record type that contains your prompt training data. Usually a dataset with three columns: id, prompt, response. Prompt containing the keywords associated to trigger the action. The response field should match the ID field. You can use a CBB_FlexQuery wrapper around a recordType, or simply just use an tostring() over an a!map() of prompt data.

recordType (Any Type): The record type that will be used to answer data related questions.

filters (Any Type): List of a!queryFilters to apply to the query. This is where you could apply a filter to a record summary using a rule input.

fields (Any Type): The fields you want to use in the data query. Default grabs all the fields in the record type. To add an additional relationship, key the relationship from the base record type. It will then grab all the related data fields from that relationship.

batchSize (Any Type): Batch size. Default 100.

recordActionList (Any Type): List of record actions. Make sure this aligns EXACTLY to the order listed in the promptData.

relatedActionList (Any Type): List of related actions. Make sure this aligns EXACTLY to the order listed in the promptData.

showAILogo (Boolean): Boolean to show AI logo. Default true.

shape (Text): Shape parameter for the card layout of the bot.

decorativeBarPosition (Text): DecorativeBarPosition parameter for card layout.

decorativeBarColor (Text): DecorativeBarColor parameter for card layout.

chatColor (Text): Hex color option for chat bot color. Default "#1C9797"

relatedActionData (Any Type): Data queries for the related action identifier you are searching for. These should be formatted using the CBB_flexQuery expression rule.

These should be in EXACT order as listed in relatedActionList. Ask yourself the question "What data would the bot need to look through to find this object?" Narrow down the fields in CBB_flexyQuery to just the records Id field and the field you will likely be searching with.

chatIconSize (Text): The size of the icon next to the chat bubbles. Default “MEDIUM”

turnOffActions (Boolean): Rule input to turn off the actions feature. Default false.

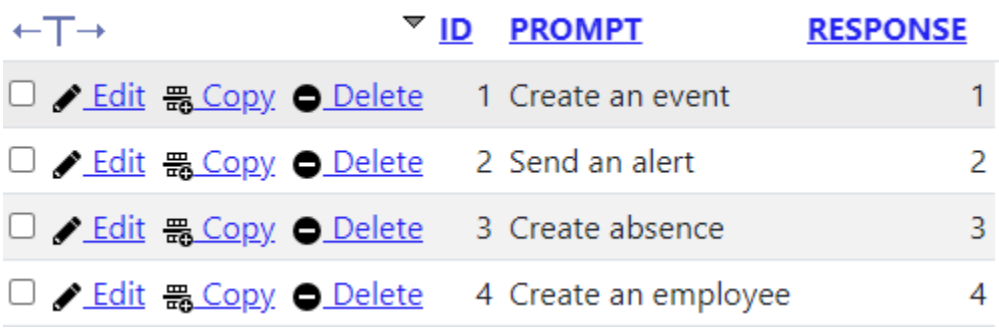
Example Prompt Data:

↑ Id	Prompt	Response
1	Create an event	1
2	Send an alert	2
3	Create absence	3
4	Create an employee	4
5	update employee	5
6	Update event	6
7	Update alert	7
8	Update operation	8
9	Update location	9

9 items

Customizations

If you want the bot to do basically anything (ex. display a form input, provide a completion, run a process model) you will need to follow a few somewhat simple steps.

1. Add a new prompt
    a. I typically use CBB Prompt Record Type and insert a new row from the Cloud DB. If you want to use a different record type the fields should be: id, prompt, and processId. In CBB_Configurations you will use CBB_FlexQuery over your prompt record and use that as your value for 'promptData'.
    b. 

	ID	PROMPT	RESPONSE
Edit Copy Delete	1	Create an event	1
Edit Copy Delete	2	Send an alert	2
Edit Copy Delete	3	Create absence	3
Edit Copy Delete	4	Create an employee	4

    c. If you want to use an a!map() instead, just make sure it is casted as a text value and add your new prompt. These configurations for the bot are found in CBB_Configurations.
    d.
```
promptData: tostring(
  {
    a!map(id: 1, prompt: "create new task", processId: 1),
    a!map(id: 2, prompt: "send an email", processId: 2)
  }
),
```
2. If you added the prompt to the DB, sync the associated record type
3. Navigate to CBB_AppianActionBot interface
4. Create a new local variable under /* Variables for Interface */. You should name this something like local!*myaction*Trigger. Set its value to **false**.
5. Ctrl + F, and find /* Logging Triggers */
    a. This is where your trigger will be hit after the user submits their prompt. You can see how I associate the triggers to their related/record action.
    b. Add a new if statement that looks like this
    c.
```
/* Logging triggers */
if(local!response = myProcessId, a!save(local!myactionTrigger, true), a!save(local!myactionTrigger, false)),
```
    d. Go to the related action trigger and data trigger below statement and add a **not = to your processId value**
```
/* If related action */
if(and(local!response >= local!relatedActionsIndex, local!response < 911, not(local!response = myprocessId)), a!save(
```
    e.

f. If you want to run a process or something like that once it hits the trigger, add that above /* If asking for a related action run CBB search */. The format of this code would look something like this but for your trigger.

```
/* If asking data question run CBB Integration to answer question */
if(local!dataTrigger,
{
  a!save(local!completion, rule!CBB_GPT35TurboSummarizeWithContext(
    data: rule!CBB_flexQuery(
      recordType: local!recordType,
      fields: local!fields,
      filters: local!filters,
      batchSize: local!batchSize
    ),
    prompt: local!prompt
  ).result.Response.choices[1].message.content)
},
{
  /* Do NOTHING */
}),
```

g.

6. Ctrl + F, and find /* Conversation Saving */
   a. In the a!map() that is being appended add a new parameter for your *myaction*Trigger

```
/* Conversation Saving */
a!save(local!conversation,
append(local!conversation,
a!map(
  id: local!counter,
  prompt: local!prompt,
  response: local!response,
  completion: local!completion,
  myactionTrigger: local!myactionTrigger
  recordAction: local!recordActionTrigger,
  relatedAction: local!relatedActionTrigger,
  dataTrigger: local!dataTrigger,
```

   b.
   c. If you ran a process for that trigger and saved the value to a local variable, add that to the conversation saving as well as another parameter so you can display it later.
   d. Scroll down to /* Reset Triggers */ and save your action trigger back to false.

```
/* Reset Triggers */
a!save(local!cantFind, false),
a!save(local!recordActionTrigger, false),
a!save(local!relatedActionTrigger, false),
a!save(local!dataTrigger, false),
a!save(local!counter, local!counter + 1),
a!save(local!prompt, null()),
a!save(local!relatedActionIdentifier, null),
a!save(local!myactionTrigger, false)
```

e.

7. Last step. Ctrl + F, /* Display Message for Record Action */. You will add a line of code just above this.
    a. I would copy my format of the side by side layout to give it consistency.
    b. Make this display/do whatever you want. You will key into this value by local!conversation[fv!index].*whatYouSavedInConversationSaving*
    c. Add a showWhen fv!item.*myactionTrigger*
8. Test it out. Reach out if you have any questions or run into trouble. If your customizations are very strenuous just reach out.