# 5.3 All Scenario Exercises

## Bright Pattern Documentation

Generated: 7/06/2022 10:03 pm

# Table of Contents

1. REDIRECT 5.3:Scenario-builder-reference-guide/Exercises/HowtoCreateChatScenarioThatPopsCaseorContactInformation

1. REDIRECT 5.3:Scenario-builder-reference-guide/Exercises/HowtoCreateChatScenarioThatUsesBots

# How to Configure a Chat Scenario That Uses a Microsoft Azure Web App Bot

In our tutorial, How to Configure a Microsoft Azure Web App Bot for Chats, you learned how to set up a basic Web App Bot from a template.

In this scenario exercise, you will learn how to edit a Bright Pattern Contact Center scenario to use your Web App Bot in a chat conversation.

## Procedure

Because we are using Bright Pattern scenario blocks for authentication to the Direct Line Bot Framework and for having a chat conversation, no integration account setup is needed. Simply download our example scenario, and follow the instructions to edit it.

### Step 1: Import and edit example scenario

1. In the Bright Pattern Contact Center Administrator application, open the scenario that will be using your bot.

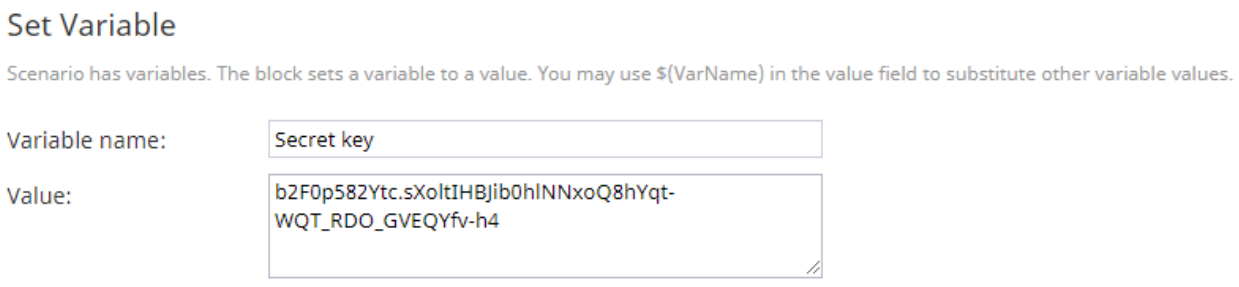
2. Import our example scenario File:App Azure Web App Bot Example.zip to get started quickly. This basic scenario includes comments to describe how the scenario blocks are being used.

### Step 2: Add secret key in Set Variable

In the first Set Variable block, set the following properties:

1. **Variable name** - Name the block (e.g., “Secret”)

2. **Value** - The secret key generated by the Azure Bot Framework when you created the Direct Line channel. Your Bright Pattern scenario will use this secret key to authenticate the Direct Line API requests that it issues to communicate with your bot.



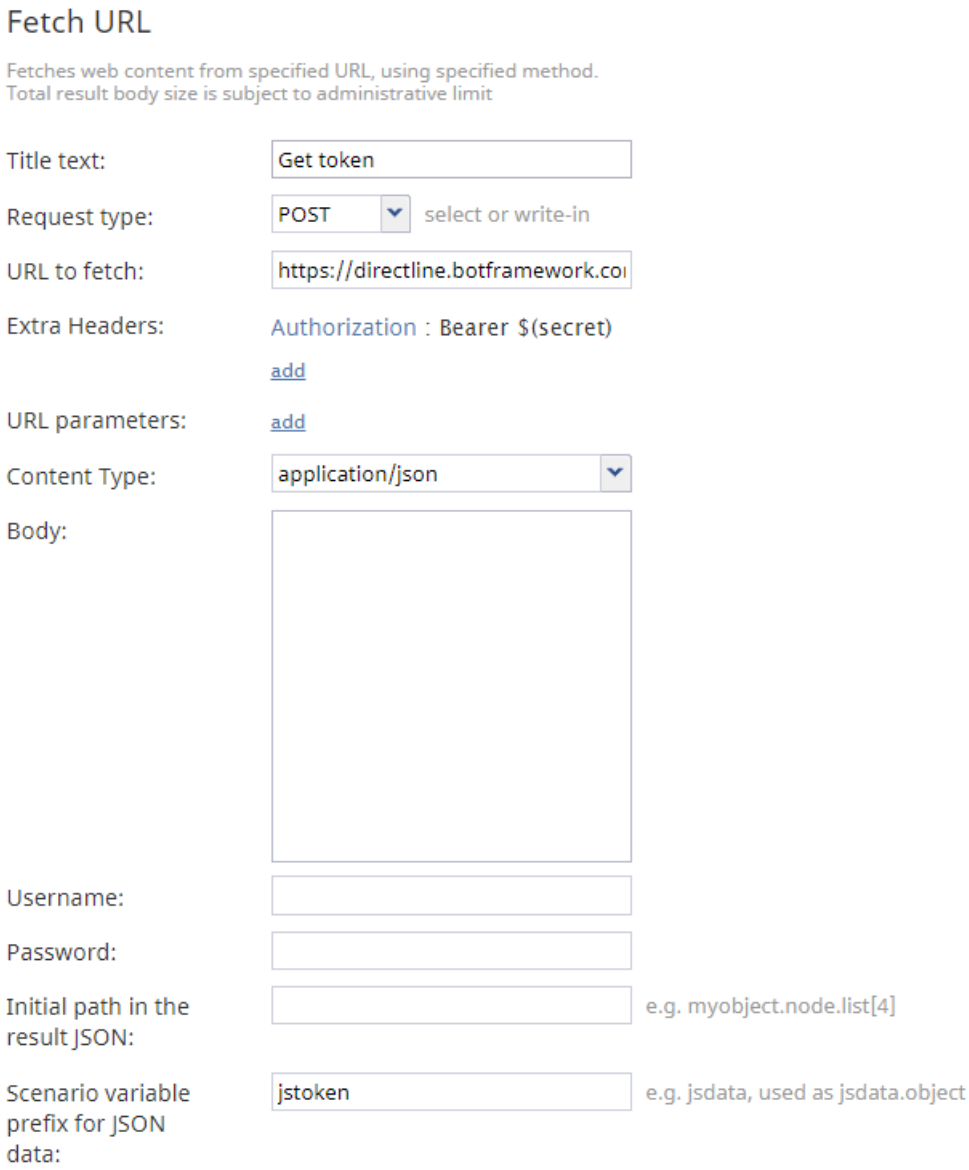

Variable scenario block with the secret key in the Value field

### Step 3: Authenticate to Direct Line API 3.0

In the first Fetch URL block we will obtain a session token to authenticate requests to Direct Line API 3.0 by using the secret that you copied from the Direct Line channel configuration page.

Set the following properties:

- **Title text** - Name the block (e.g., “Get token”)
- **Request type** - POST
- **URL to fetch** - https://directline.botframework.com/v3/directline/tokens/generate
- **Extra Headers** - Authorization: Bearer $(secret)
- **URL parameters** - None
- **Content Type** - application/JSON
- **Body** - Empty
- **Username** - Leave blank
- **Password** - Leave blank
- **Initial path in the result JSON** - Leave blank
- **Scenario variable prefix for JSON data** - jstoken

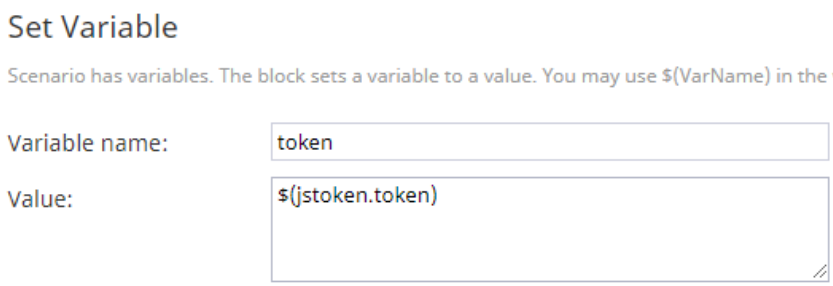



Fetch URL scenario block being used for authentication

### Step 4: Add token to Set Variable

In the second Set Variable block, set the following properties:

- **Variable name** - Name the block (e.g., “token”)
- **Value** - $(jstoken.token)




Variable scenario block setting the token variable the Value field

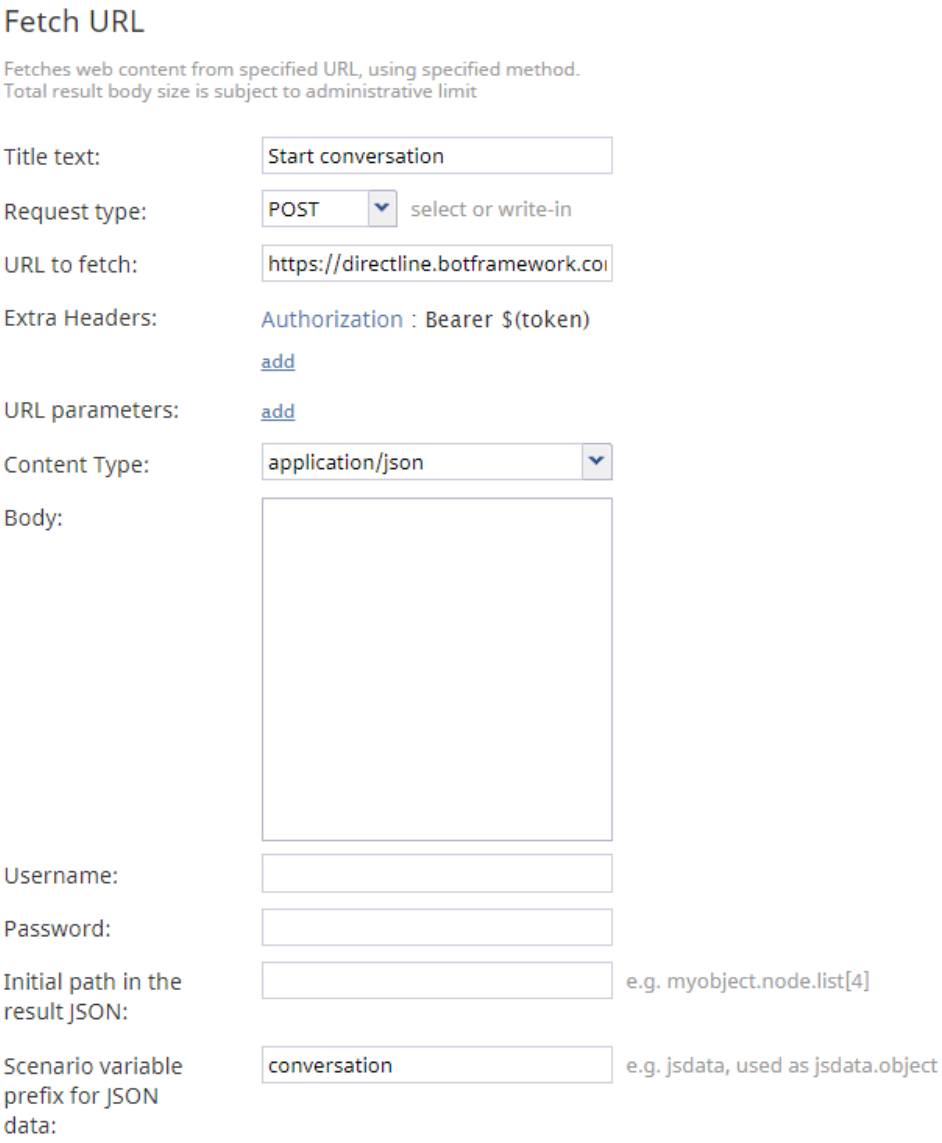
## Step 5: Start a conversation

In the second Send Message+ block, we will open a new Direct Line conversation by issuing a POST to the */v3/directline/conversations* endpoint. While the conversation is open, both the bot and client may send messages.

If the request is successful, the response will contain an ID for the conversation (i.e., "conversationId" which is needed in later steps), a token, a value that indicates the number of seconds until the token expires, and a stream URL that the client may use to receive activities via WebSocket stream.

Set the following properties:

- **Title text** - Name the block (e.g., "Start conversation")
- **Request type** - POST
- **URL to fetch** - https://directline.botframework.com/v3/directline/conversations/
- **Extra Headers** - Authorization: Bearer $(token)
- **URL parameters** - None
- **Content Type** - application/JSON
- **Body** - Empty
- **Username** - Leave blank
- **Password** - Leave blank
- **Initial path in the result JSON** - Leave blank
- **Scenario variable prefix for JSON data** - conversation



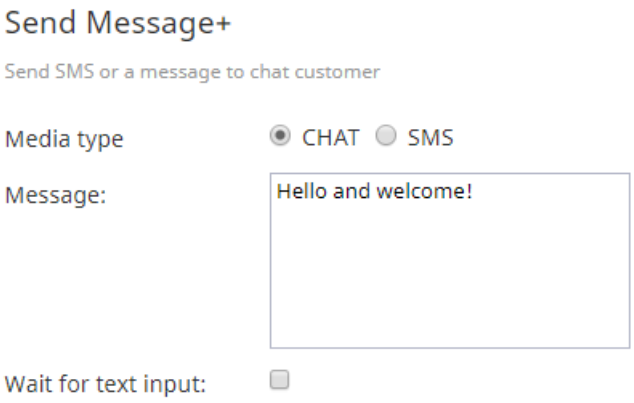

Fetch URL block being used to open a new Direct Line conversation

### Step 6: Send a message to the customer

In the Send Message+ block, specify the message you want to send in the conversation.

Set the following properties:

- **Media type** - Chat
- **Message** - Any text message (e.g., “Hello and welcome!”)



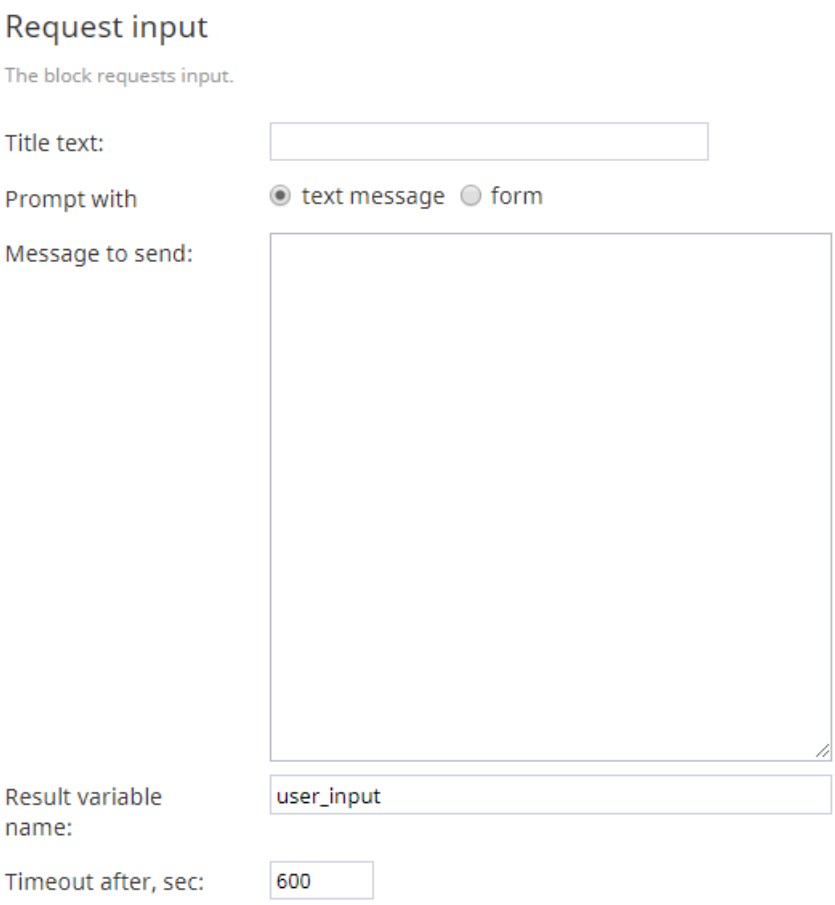

Send Message+ block being used to send a message to the customer

### Step 7: Prompt for a customer response or wait

In the Request Input block, we can optionally prompt for a response from the customer and save the response to the "user_input" variable name. Alternatively, we can simply wait for text input from the previous scenario block.

Set the following properties:

- **Prompt with** - text message
- **Message to send** - Leave empty
- **Result variable name** - user_input




Request Input block could be used to prompt for a customer response

### Step 8: POST activities

In the third Fetch URL block, we will use the Direct Line 3.0 protocol to exchange activities supported by your bot, like message activities and typing activities. We will issue a POST to the */v3/directline/conversations/$(conversation.conversationId)/activities* endpoint, where "$(conversation.conversationId)" is the ID that was obtained from the previous Fetch URL block's POST request to the the */v3/directline/conversations* endpoint.

Set the following properties:

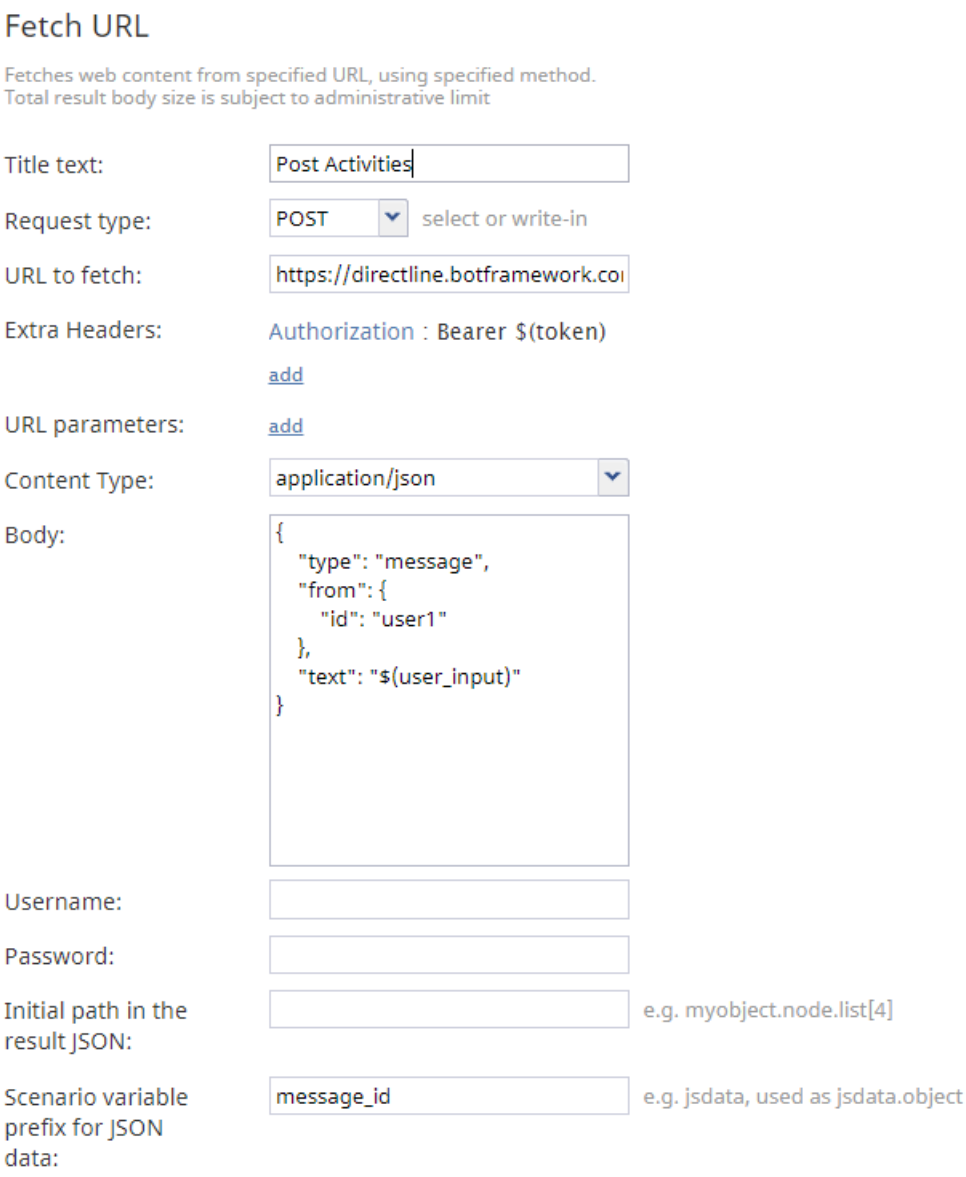
- **Title text** - Name the block (e.g., "Post Activities")
- **Request type** - POST
- **URL to fetch** - https://directline.botframework.com/v3/directline/conversations/$(conversation.conversationId)/activities
- **Extra Headers** - Authorization: Bearer $(token)
- **URL parameters** - None
- **Content Type** - application/JSON

- **Body** - Specify the Activity object in the body of the request.

  For example:

```
{
   "type": "message",
   "from": {
      "id": "user1"
   },
   "text": "$(user_input)"
}
```

- **Username** - Leave blank
- **Password** - Leave blank
- **Initial path in the result JSON** - Leave blank
- **Scenario variable prefix for JSON data** - message_id




Fetch URL block being used to POST activities

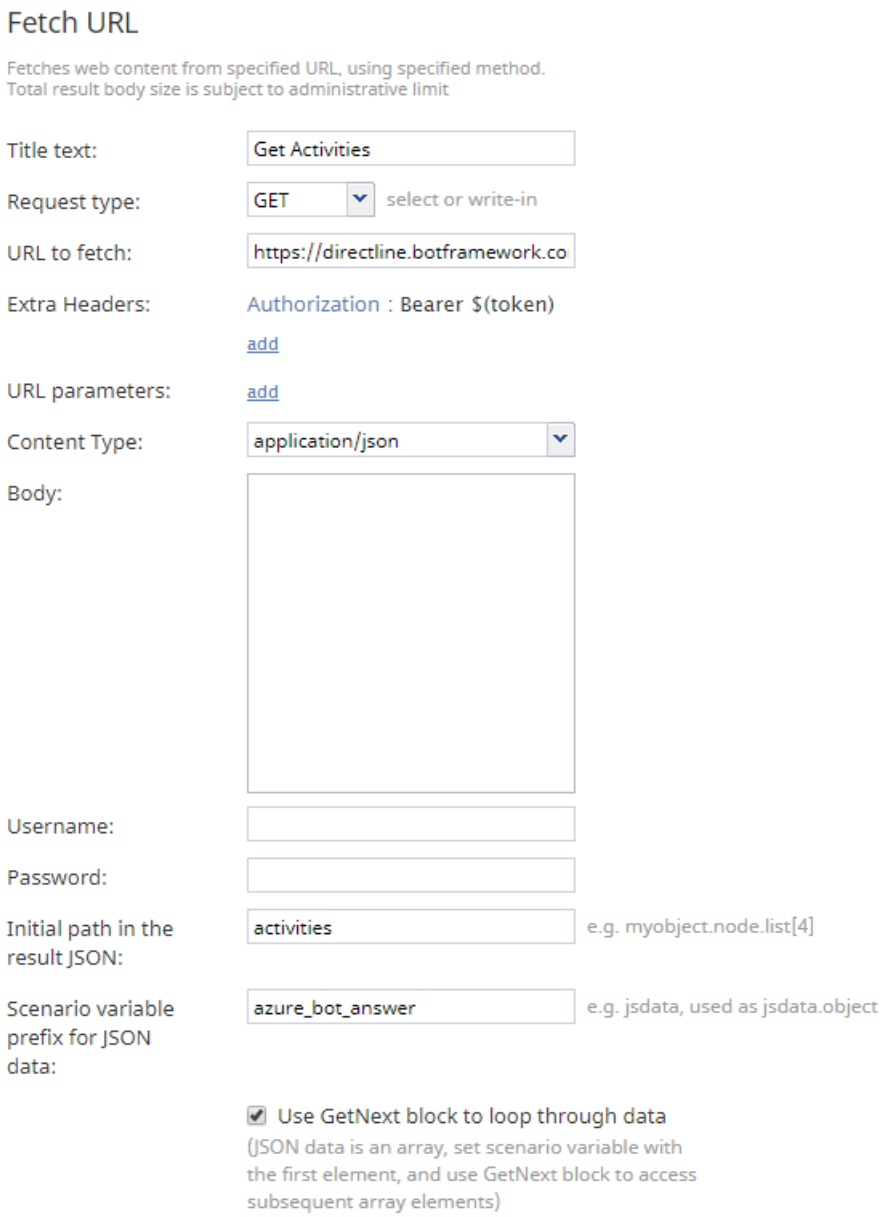
### Step 9: Retrieve activities

In the fourth Fetch URL block, we will retrieve activities (e.g., your bot's answer) by using HTTP GET.

Set the following properties:

- **Title text** - Name the block (e.g., "Get Bot Answer")
- **Request type** - GET
- **URL to fetch** - https://directline.botframework.com/v3/directline/conversations/$(conversation.conversationId)/activities
- **Extra Headers** - Authorization: Bearer $(token)
- **URL parameters** - None

- **Content Type** - application/JSON
- **Body** - Empty
- **Username** - Leave blank
- **Password** - Leave blank
- **Initial path in the result JSON** - activities
- **Scenario variable prefix for JSON data** - azure_bot_answer
- **Use GetNext block to loop through data** - Select checkbox



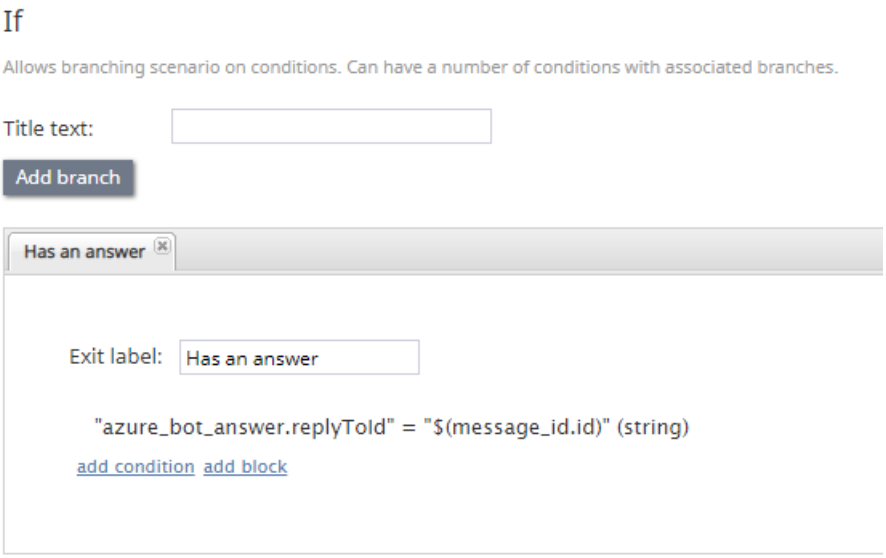

Fetch URL block being used to retrieve the bot's answer

### Step 10: Set conditions for routing bot answers

In the If block and the Send Message+ block, we will specify that if the bot has an answer, then we will send that answer to the customer in a chat message using *$(azure_bot_answer.text)*.

Set the following properties:

- **Exit label** - Has an answer
- **Conditions**
    - Scenario variable (string)
    - azure_bot_answer.replyToId
    - is
    - =
    - $(message_id.id)

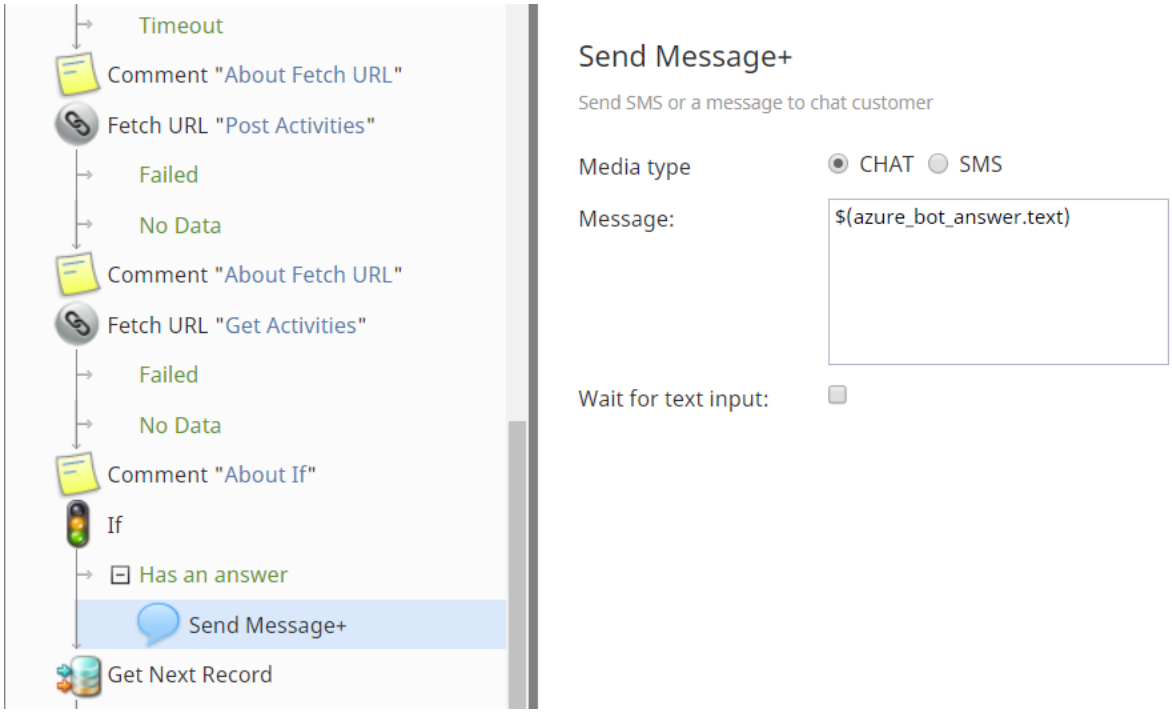



If block being used to set conditions to send the bot's answer to the customer, if there is one

### Step 11: Pass along the bot's answer to the customer

Under the “Has an answer” exit, add a Send Message+ block and set the following properties:

- **Media type** - Chat
- **Message** - $(azure_bot_answer.text)



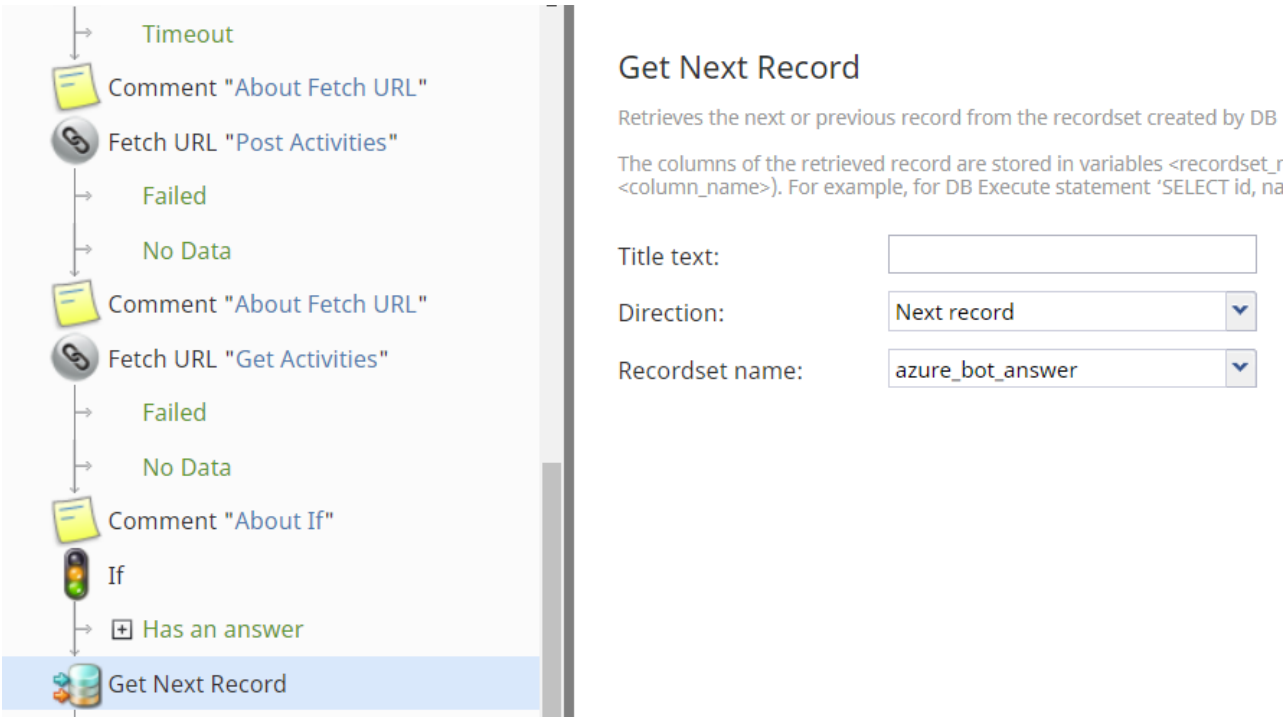
Send Message+ block being used to send the bot's answer

### Step 12: Get the next bot answer

In the Get Next Record block, we will get the next bot answer, if there is one.

Set the following properties:

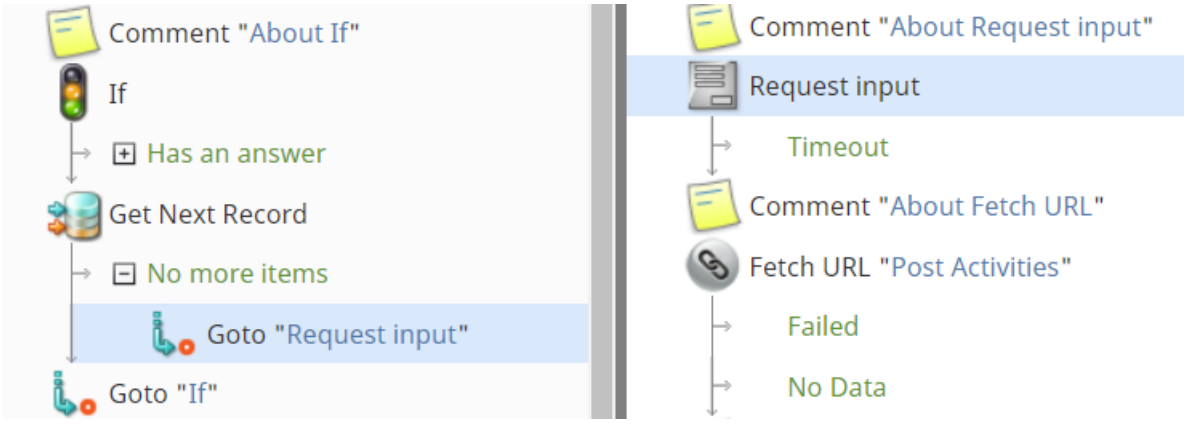
- **Title text** - Any name
- **Direction** - Next record
- **Recordset name** - azure_bot_answer




Get Next Record block being used to request the next bot answer

### Step 13: Use Goto to request input again

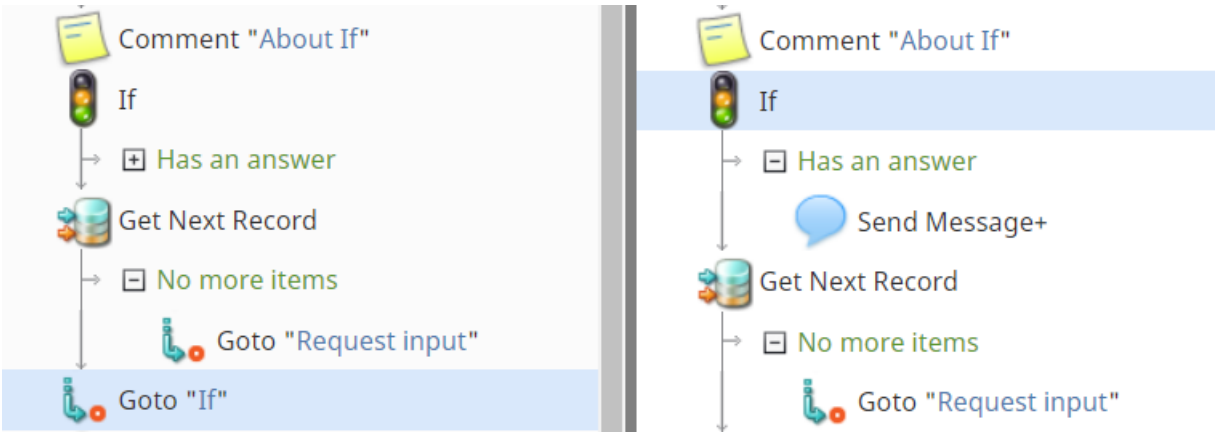
If there are no more answers from your bot, request input again by using the Goto block.




Goto block pointing to Request Input

### Step 14: Keep checking for bot answers

Add another Goto block and point it to the If block. Doing so will create a loop that gets the bot's answer, sends it in chat, and checks for more bot answers until there are no more.



Goto block pointing to If block

### Step 15: End the conversation

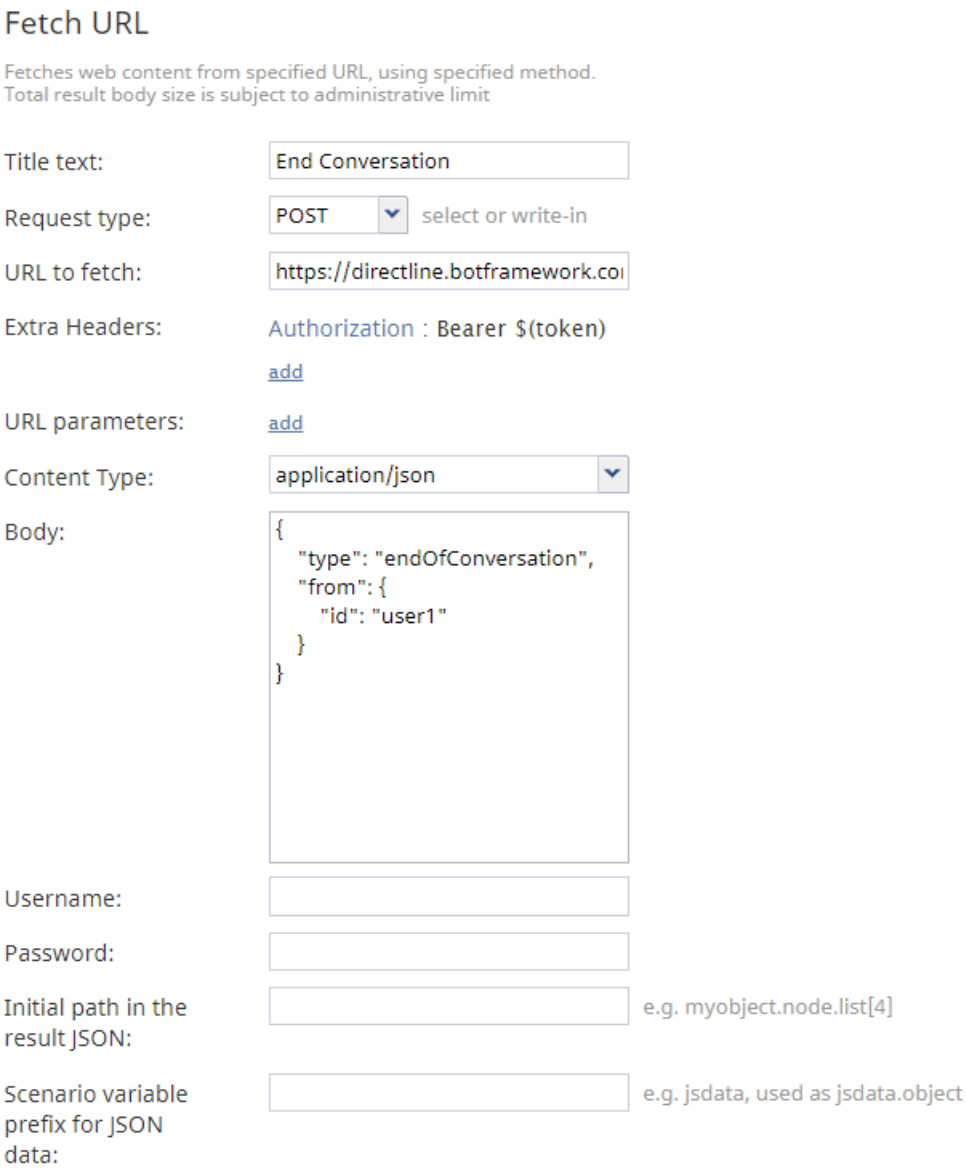
In the last Fetch URL block, we will end the conversation by sending a POST with the “endOfConversation” activity to the channel's messaging endpoint.

Set the following properties:

- **Title text** - Name the block (e.g., “End Conversation”)
- **Request type** - POST
- **URL to fetch** - https://directline.botframework.com/v3/directline/conversations/$(conversation.conversationId)/activities
- **Extra Headers** - Authorization: Bearer $(token)
- **URL parameters** - None
- **Content Type** - application/JSON
- **Body** - Specify the “endOfConversation” activity in the body:

```
{
   "type": "endOfConversation",
   "from": {
      "id": "user1"
   }
}
```

1. **Username** - Leave blank
2. **Password** - Leave blank
3. **Initial path in the result JSON** - Leave blank
4. **Scenario variable prefix for JSON data** - Leave blank



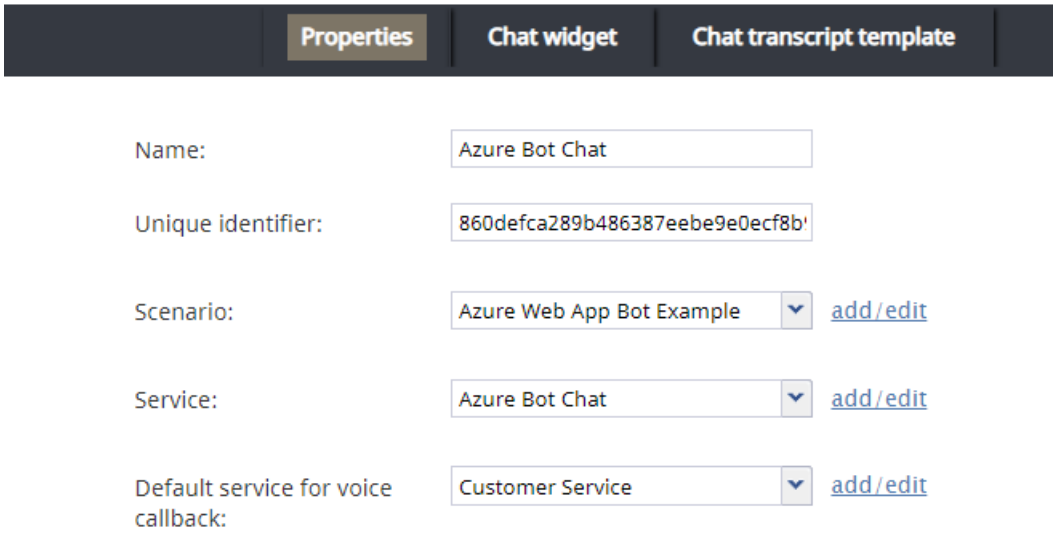

Fetch URL being used to end the conversation

### Step 16: Save

Save the scenario.

### Step 17: Configure your chat scenario entry to use the scenario

In the messaging/chat scenario entry you wish to use, make sure to set the correct scenario and service for your Web App Bot-enabled chat.



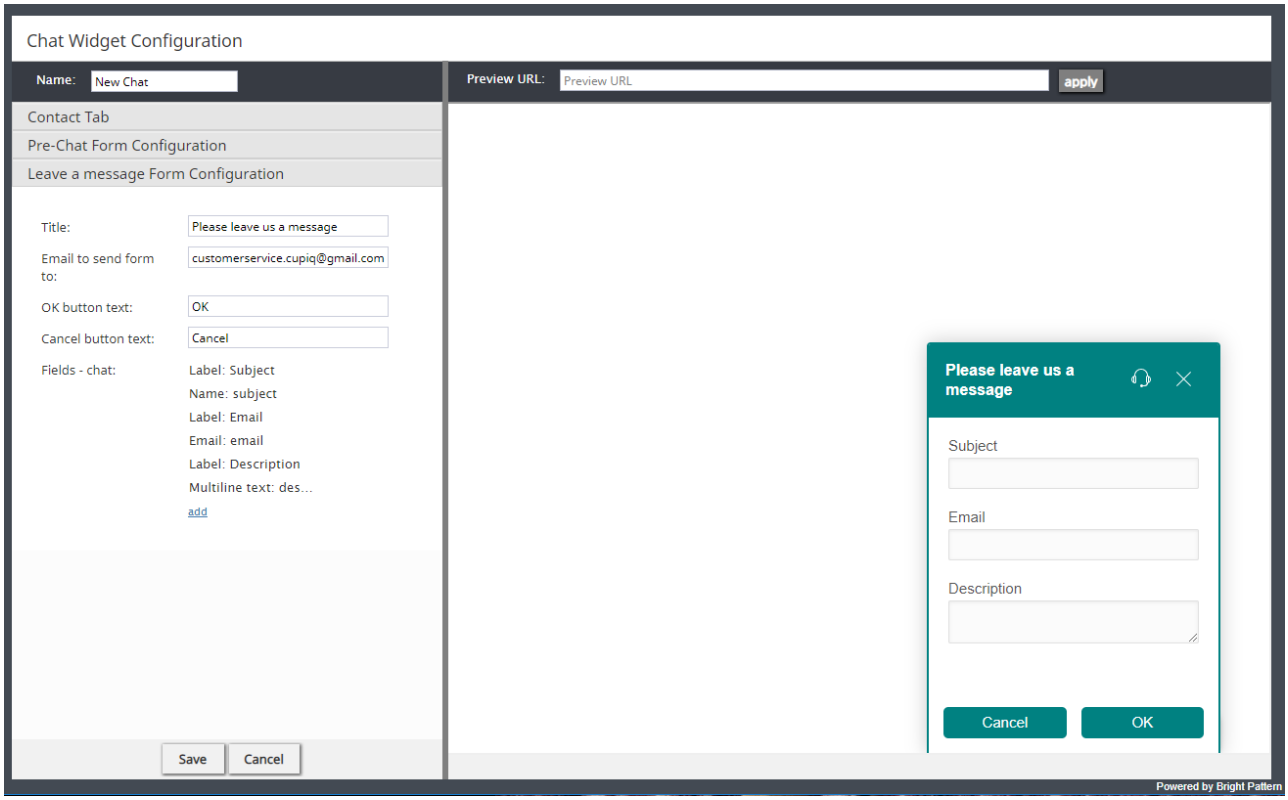

Messaging/Chat scenario entry properties

Your configuration is now complete. You can now launch a chat and try it.

# Sending Automatic Email Replies to Customers Who Use the "Leave a Message" Chat Form

If your contact center offers chat services within a set of operating hours, the Leave a Message Form option allows customers to use the chat widget to send you email messages outside of these hours of operation.

Using the Scenario Builder application, you can configure the system to send automatic email replies to customers who send after-hours messages. This ensures you're providing great customer service even when your contact center is closed.



Configuring the Leave a Message chat form

## Prerequisites

Note that this scenario requires you to have configured chat and email services, configured chat and email scenario entries, as well as a working SMTP configuration.

Because a key part of this example is based on hours of operation, it is helpful to understand how to configure them in your contact center. And finally, you may find it helpful to review the variable $(item.externalChatData) as it refers specifically to fields in the chat widget.

For more information about configuring the chat widget, see *Administration Tutorials*, section Chat, as well as the *Chat Widget Configuration Guide*.

## Scenario Example

**Click the following link to download an annotated version of this chat scenario example.**
File:App After-Hours Send an Email Reply to a Chat Customer.zip

For instructions on how to import this file into your contact center, see the *Contact Center Administrator Guide*, section Scenarios Overview > How to Export and Import Scenarios.

For general information about scenarios, refer to section Scenario Builder Overview.

## Example Flow

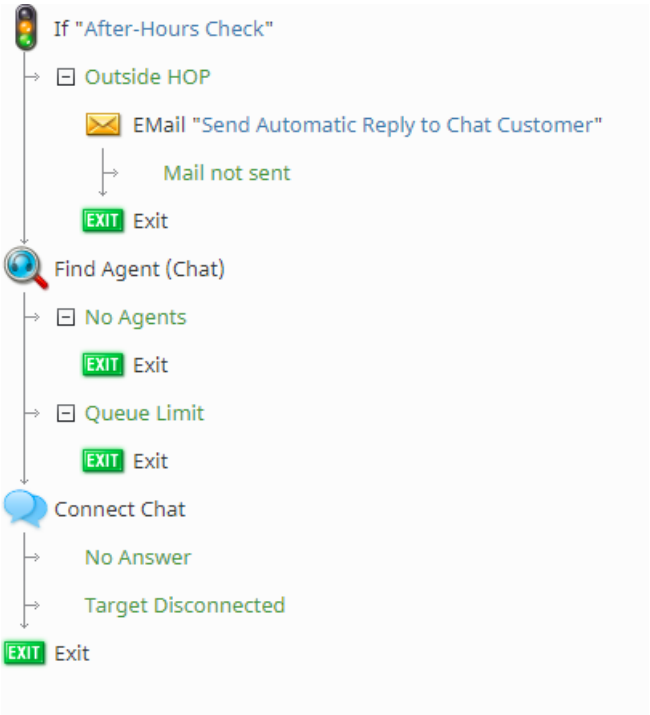
This example illustrates how to configure the Leave a Message form in the Chat Widget Configuration application, focusing on field variables. Then, in the Scenario Builder application, we configure an If block to perform an after-hours check. If messages are received outside of the configured hours of operation, we use the EMail block with the chat widget's field variables to send email replies to customers.

**Designer's note**: The same scenario blocks can be used to route the chat message directly to your contact center's email address, for troubleshooting purposes.

Note that this is an example scenario only and not intended for production use. All conditional exits should be defined with actions for production use.

## Scenario Overview

The diagram shown illustrates what the complete scenario looks like when designed in the Scenario Builder application.

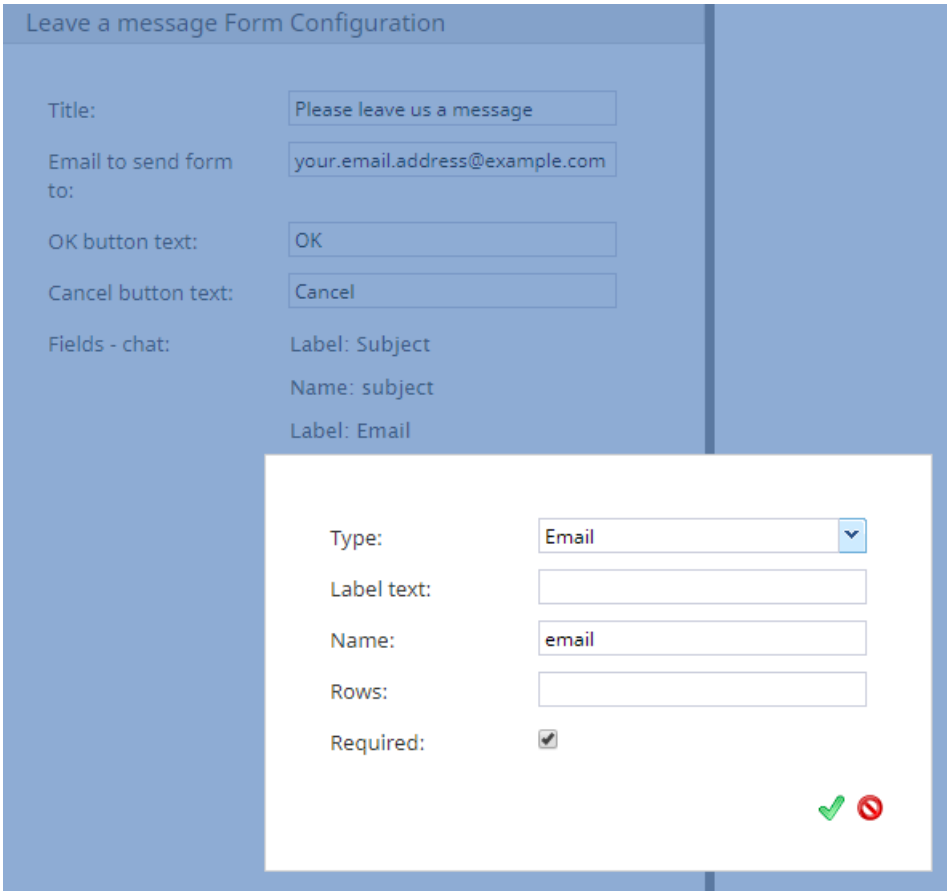



### Action 1: Configure the "Leave a Message" Chat Form in the Chat Widget Configuration Application

In order for our scenario to work, we must configure the Leave a Message form in the Chat Widget Configuration application first. To launch the application, in the Contact Center Administrator application, go to section Scenario Entries > Messaging/Chat > Chat Widget tab, and click **add** under Chat Initiation via Contact Tabs.

In the application, configure the fields you want your *Leave a Message* form to have. Note that the Email to send form to field should contain the email address you want to receive the results of these forms.

When adding fields to your form, note that the *name* can be added to the end of the variable $(item.externalChatData) as a way to reference back to a specific field (e.g., the field named *subject* can be added like so: *$(item.externalChatData.subject)*).



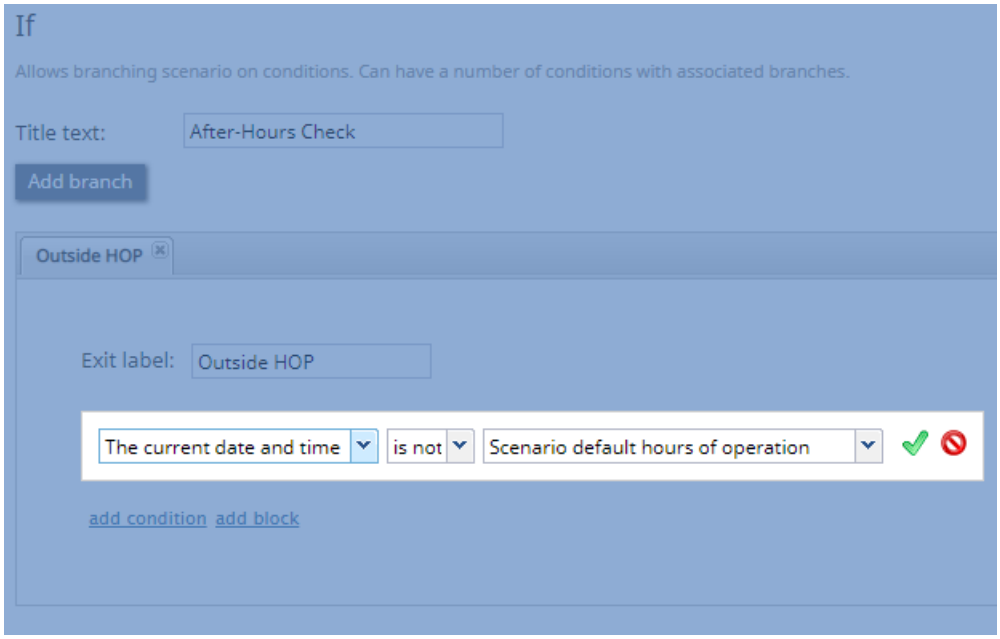
Field names are used with the variable $(item.externalChatData), so make sure you keep track of them

**Action 2: Configure an If Block to Perform an After-Hours Check**

After configuring the chat form, in the Scenario Builder application, we want the system to check if the chat is received outside of the service's hours of operation. To do this, add an If block, add a new branch, enter an exit label, then add a condition that specifically looks at hours of operation. The condition is configured like so:

- "The current date and time"
- "is not"
- "Scenario default hours of operation"

Additionally, note that you may configure specific hours of operation entries in the final field.



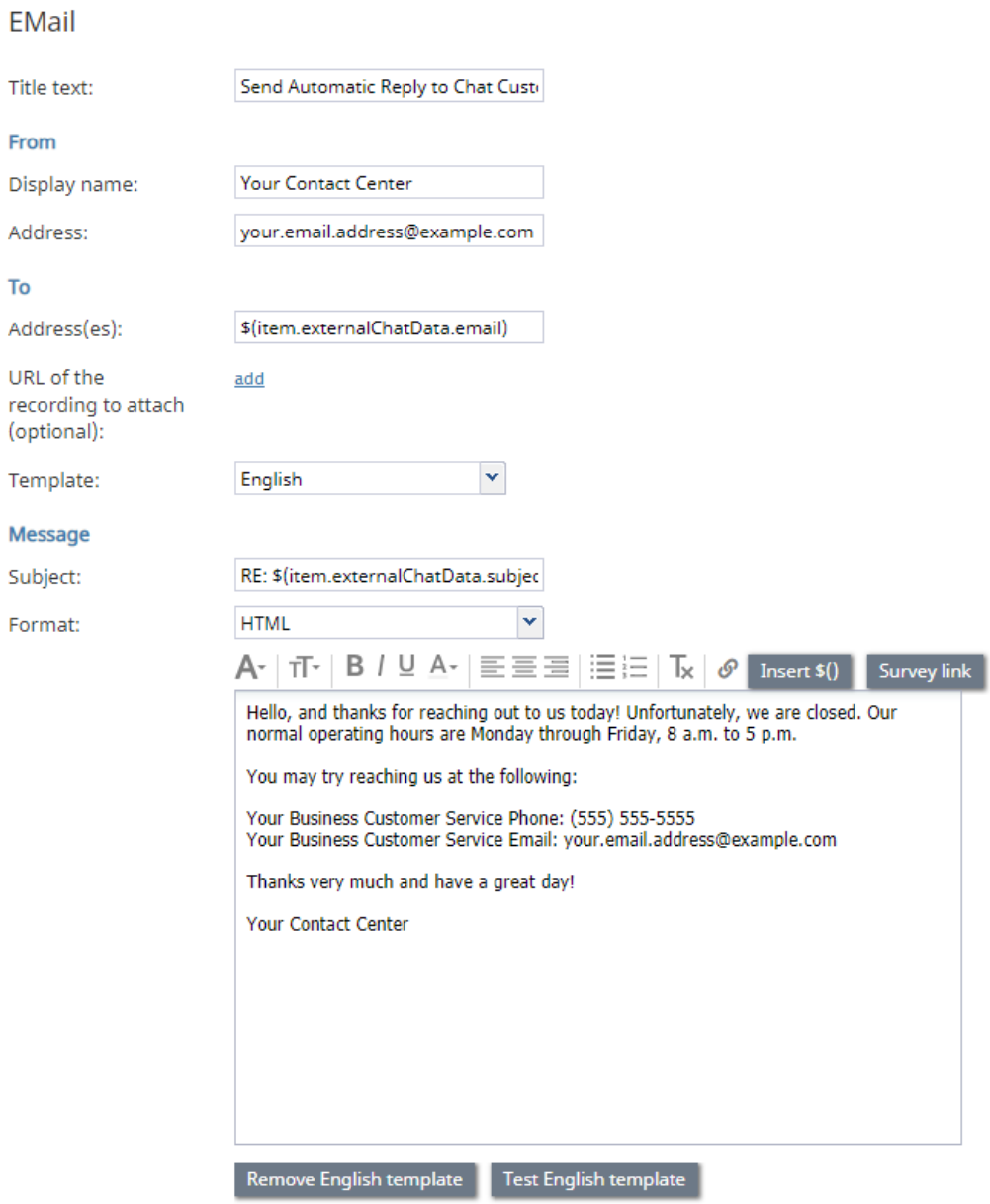
Configure an after-hours check using the If block

**Action 3: Add the EMail Block to the If Block's After-Hours Check Branch**

If the chat is received outside of the specified hours of operation, we want the system to send out an automatic email reply; this is done by adding the EMail block to the after-hours branch we created with the If block.

When configuring the EMail block, and to ensure the email populates with information the customer entered in the widget, use the variable $(item.externalChatData) where applicable.

The field names from the widget should be added to the end of the variable to pull the correct information, for example, if the widget's email field is named *email*, the variable would be *$(item.externalChatData.email)*.




Populate the EMail block with the variable $(item.externalChatData) where applicable

**Action 4: Configure Find Agent / Connect Chat**

If the chat is received during normal hours of operation, the customer will need to be connected to an agent. To do this, configure your scenario with the two most basic chat scenario blocks: Find Agent and Connect Chat. *Find Agent* is where the system finds the most appropriate agent to route the chat to; *Connect Chat* opens the chat channel between the found agent and the customer. All chat scenarios will contain the Find Agent block and Connect Chat block; note that they should be arranged in consecutive order.

For more information about these blocks, see How to Create a Basic Scenario. As a reminder, define all conditional exits.

1. REDIRECT #topic_scenario-builder-reference-guide/exercises/howtoblacklistspecificphonenumbers

1. REDIRECT 5.3:Scenario-builder-reference-guide/Exercises/HowtoCreateaVoiceScenarioThatDistributesSurveystoaPercentageofRandomCustomers

1. REDIRECT 5.3:Scenario-builder-reference-guide/Exercises/HowtoCreateaVoiceScenarioThatRoutesCallerstoLastAgentwithVoicemail

1. REDIRECT 5.3:Scenario-builder-reference-guide/Exercises/ScenarioExample

# Skill-Based Call Routing with an Auto Attendant Choice

In the Bright Pattern Contact Center realm, *skills* are the various abilities of users that can be defined in the Contact Center Administrator application. Skills can be defined as any topic or language pertinent to your contact center (e.g., accounting, Arabic, IT, etc.) Skills are rated in section Skill Levels per user from 0 to 100 (i.e., 0 being the lowest skill possible, and 100 being the highest skill possible). Additionally, users can be skilled for a service or campaign.

Through scenarios, it is possible to route interactions to higher-skilled agents before lower-skilled ones. This lets your customers interact with your most knowledgeable agents, which provides the best customer service experience. Additionally, skill-based routing is great to use when training new, lower-skilled agents; this provides them the opportunity to learn while preventing them from being overwhelmed with interactions.

This scenario details how to configure basic skill-based routing with an option for customers to enter in an extension directly (i.e., "auto attendant"). The skills used in this example are Auxiliary Skills but they may be changed to Language Skills (e.g., your contact center provides services in English, French, and Spanish).

## Scenario Example

**Click the following link to download an annotated version of this chat scenario example.**
File:App Skill-Based Call Routing + Auto Attendant.zip

For instructions on how to import this file into your contact center, see the *Contact Center Administrator Guide*, section Scenarios Overview > How to Export and Import Scenarios.

For general information about scenarios, refer to section Scenario Builder Overview.

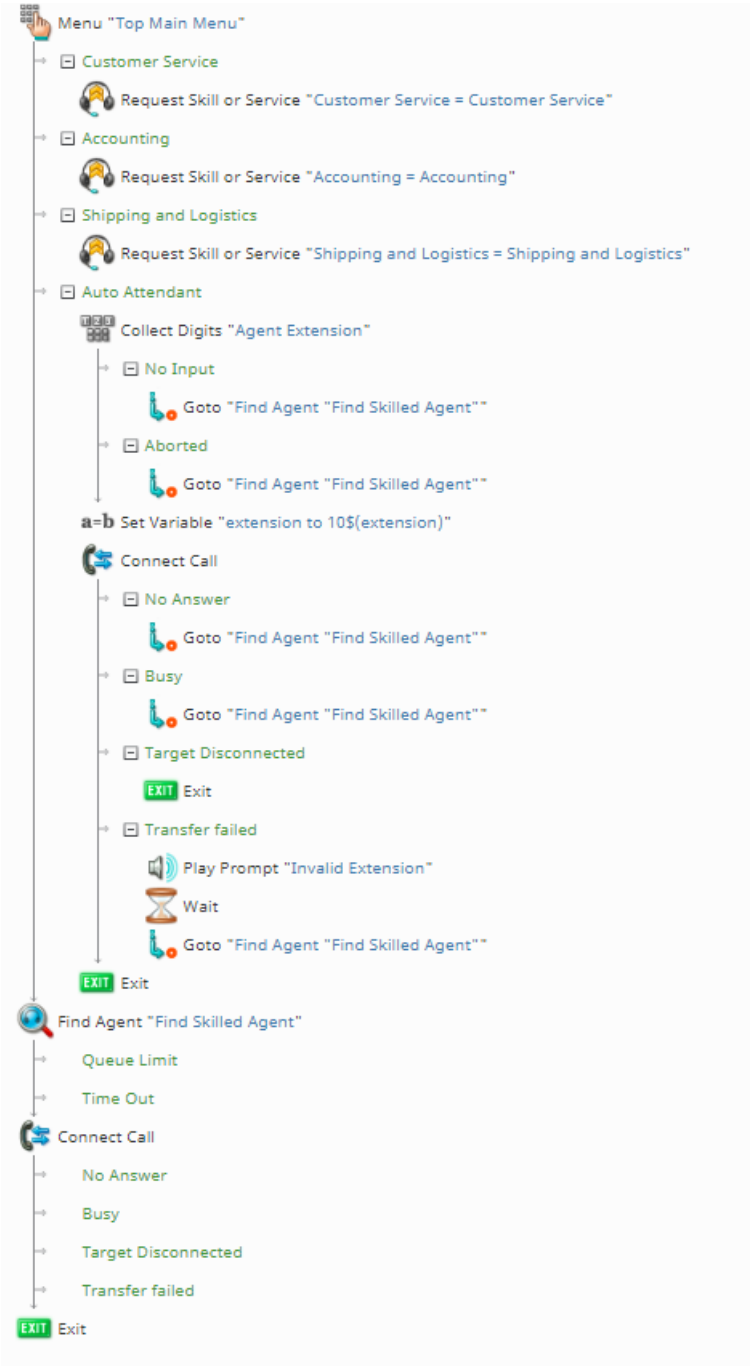
## Scenario Flow

This voice scenario uses a menu to route customers to skilled agents and includes an option to dial the extension of a specific agent. This is accomplished by using the following blocks: Menu, Request Skill or Service, Collect Digits, Set Variable, Find Agent, and two instances of Connect Call.

Note that this is an example scenario only and not intended for production use. All conditional exits should be defined with actions for production use.

### Scenario Overview

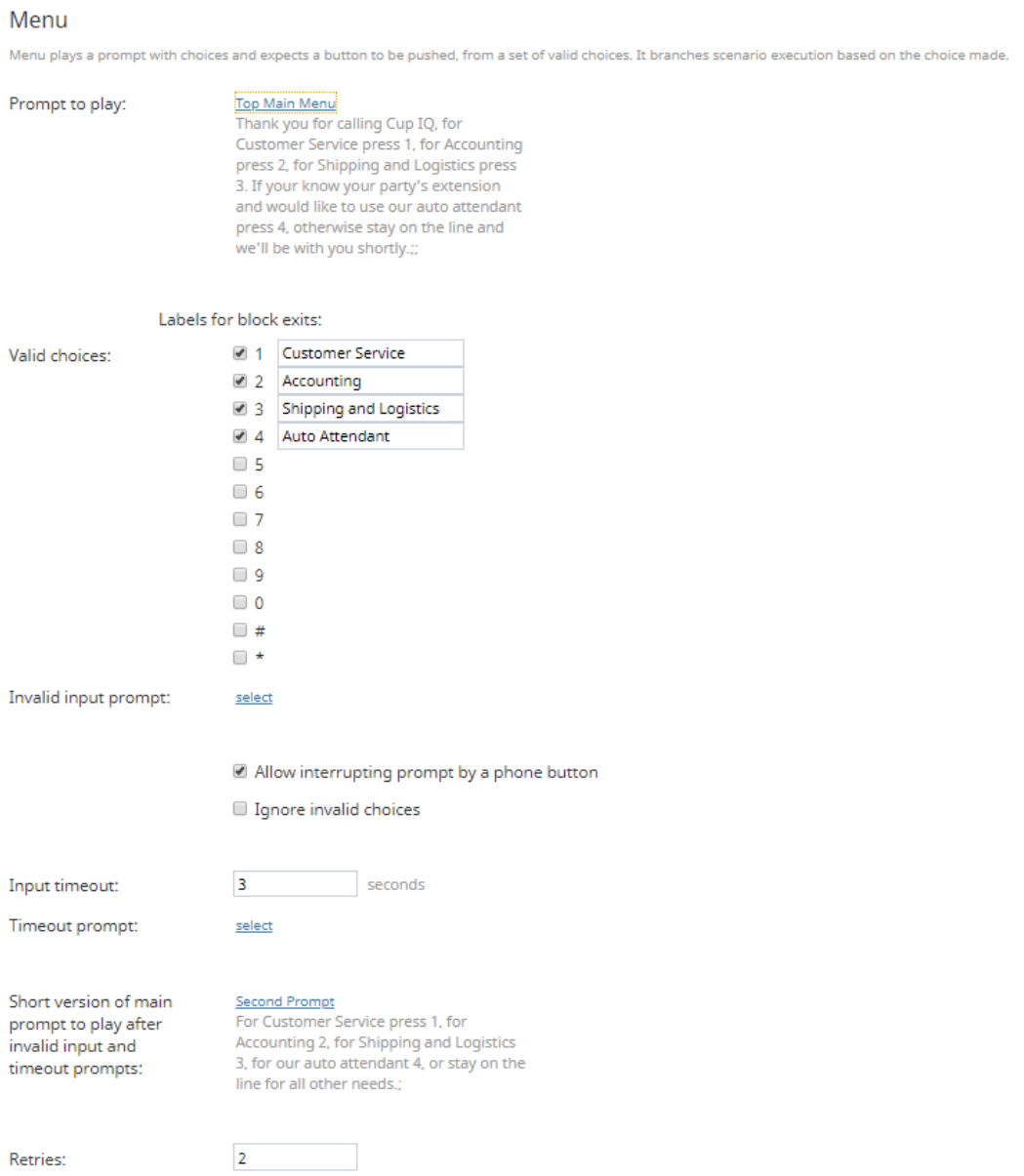
The diagram shown illustrates what the complete scenario looks like when designed in the Scenario Builder application.




### Action 1: Create a Menu with Skill-Based Choices and an Auto Attendant Choice

In this scenario, the Menu block allows callers to select a menu choice that routes them to either a skilled agent or allows them to enter the extension of a specific agent directly.

When configuring the Menu block, we create a prompt to play, then configure our valid menu choices. In this example call center, the skills are "Customer Service," "Accounting," and "Shipping and Logistics," so we create valid choices in the block for each of these. Additionally, we create the choice for the "Auto Attendant" (i.e., the caller can enter an extension).

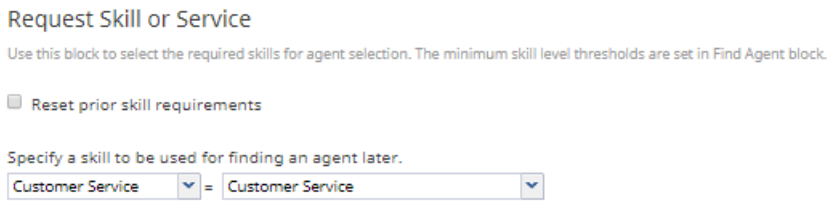



Menu block configuration

## Action 1a: For Skill-Based Menu Choices, Use the Request Skill or Service Block

Next, for each skill-based choice we added in the Menu block, we add the Request Skill or Service block and configure the corresponding skill. This is done by selecting the skill group from the first drop-down menu (i.e., auxiliary, language, etc.), then the specific skill from the skill group in the next drop-down menu.

Note that the Request Skill or Service block acts as a connector between the Menu block and the Find Agent block, where the skill-based routing happens.



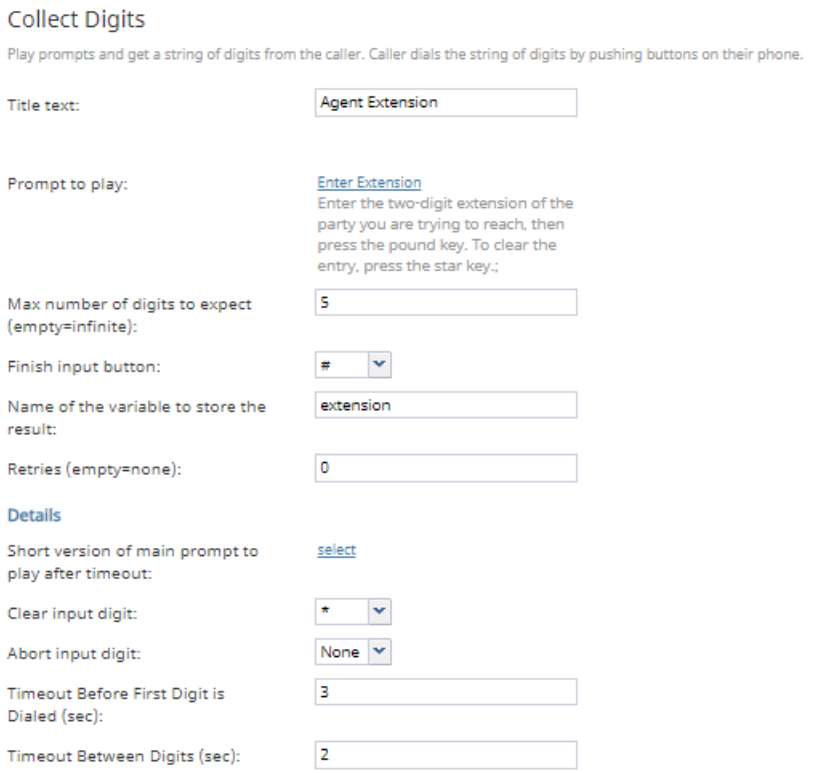

Request Skill or Service block configuration

## Action 2: Begin Configuring Auto Attendant with Collect Digits

For the Menu block choice "Auto Attendant," we allow customers to connect to a specific agent via the agent's extension number. This is accomplished by using the Collect Digits block, the Set Variable block, and a Connect Call block.

All extensions in our example call center begin with 10 (e.g., 1003), so, rather than have the customer enter in the full four-digit extension, we configure a prompt to play that tells them to enter the last two digits of the extension.

The Collect Digits block records the two digits the customer enters and stores them in the variable configured in the field Name of the variable to store the result; in our example, we call this variable *extension*.

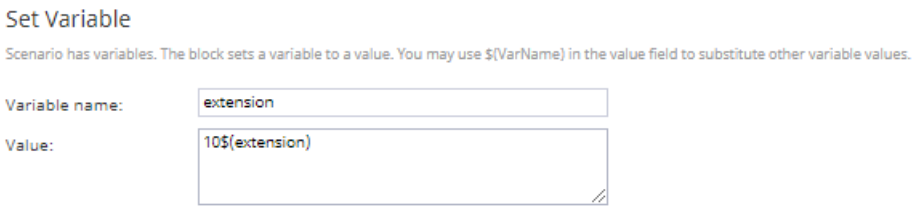



Collect Digits block configuration

## Action 2a: Use Set Variable to Alter Information from Collect Digits

Next, the variable *extension* we created in the Collect Digits block is passed to the Set Variable block; this is where the first two digits of the actual extension are added back on to the variable. This is done by appending the number "10" before the begining of the variable like so: *10$(extension)*

For example, a customer enters in *39* in the Collect Digits block, which, when passed through Set Variable, becomes *1039*, the actual extension.

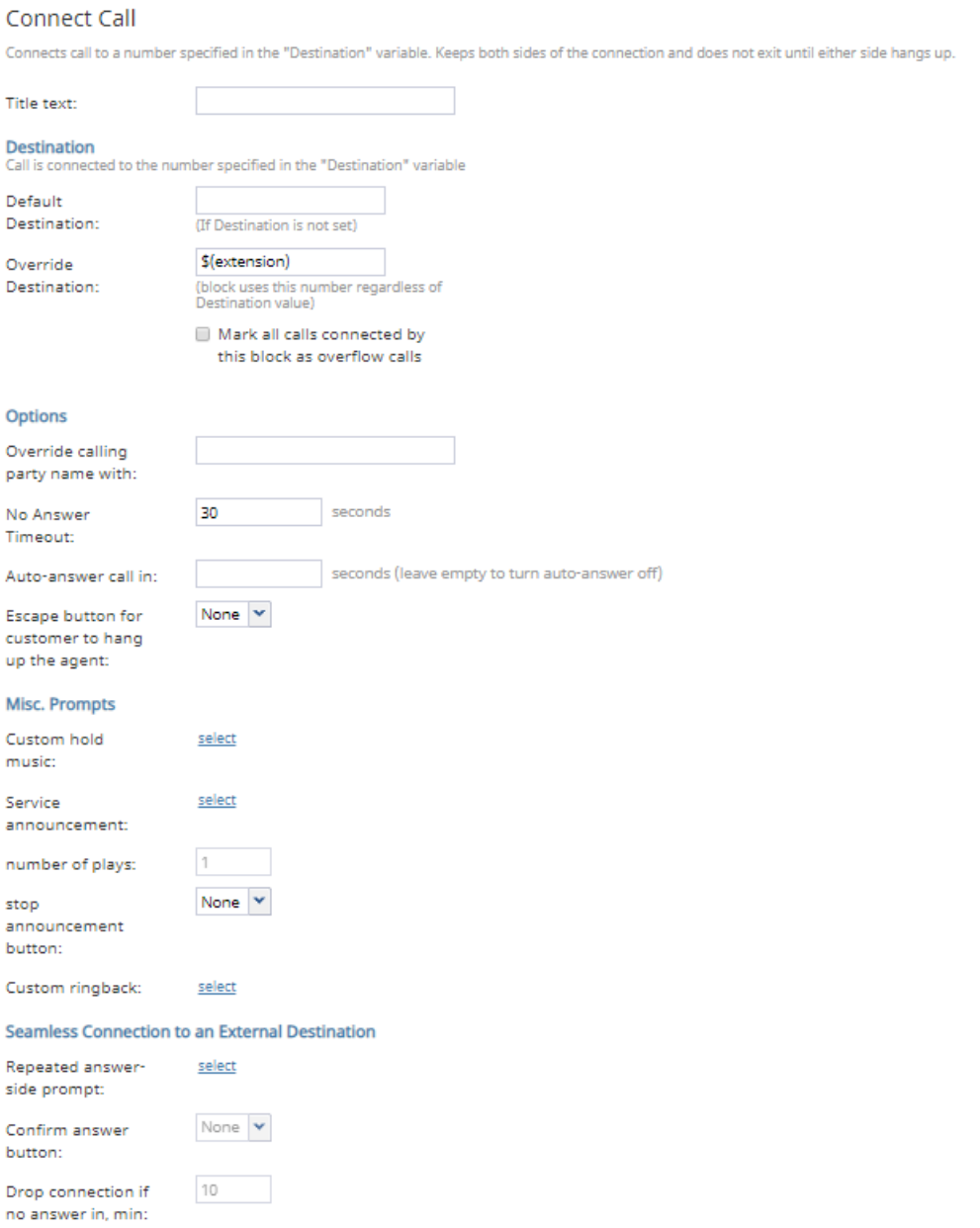



Set Variable configuration

## Action 2b: Have Connect Call Receive the Altered Information from Set Variable

Finally, the variable *$(extension)* we created in the Set Variable block is passed to the following Connect Call block and entered in the Override Destination field. This ensures the call will go to the specific extension rather than any available agent.

Note that in this branch of the scenario (i.e., the "Auto Attendant" menu choice), if the customer does not enter an extension or the system does not recognize it, we use Goto blocks to route the caller to the Find Agent block.

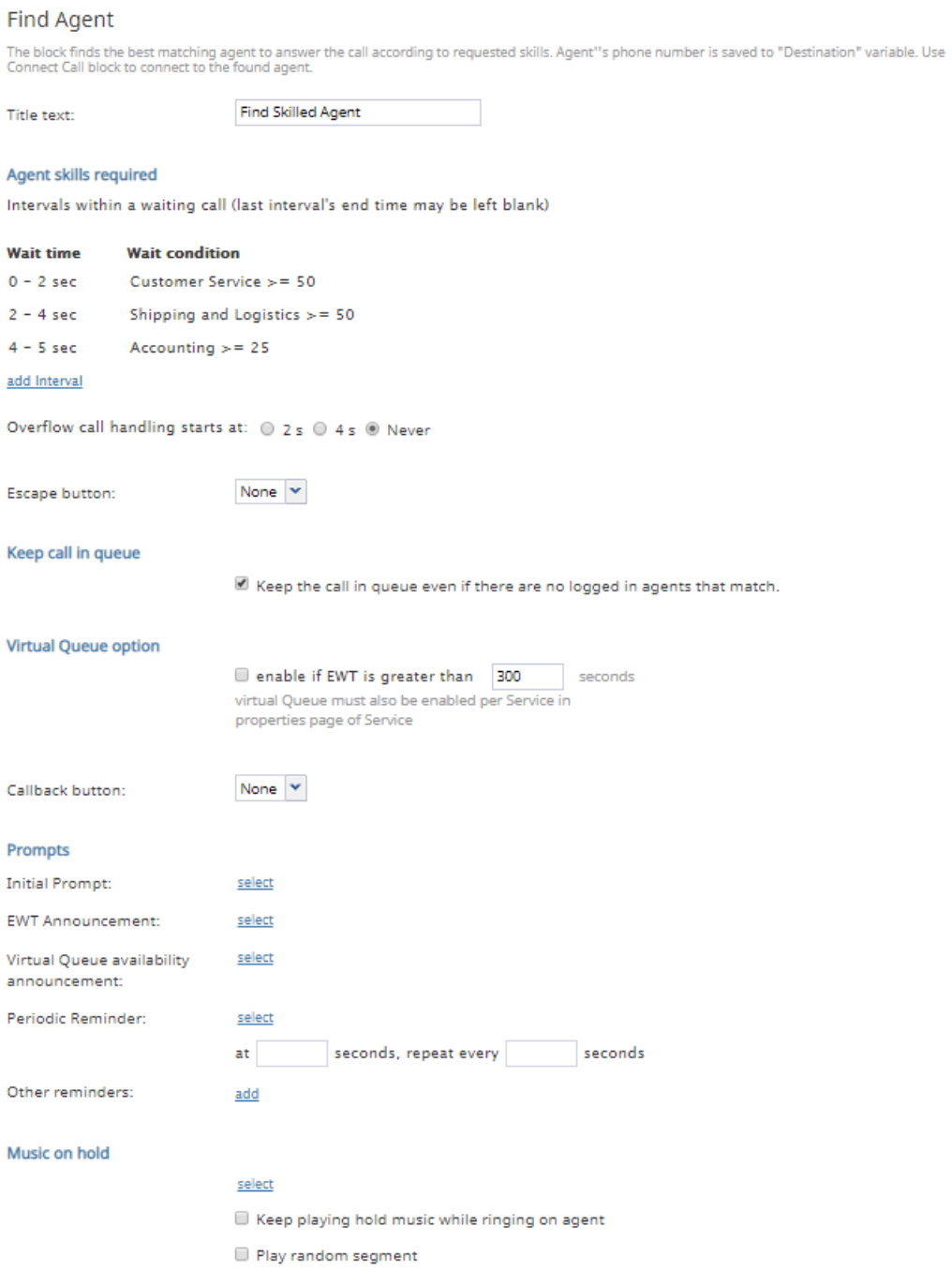


Connect Call block configuration

## Action 3: Configure Skill-Based Routing in Find Agent

If the caller selected a skill-based choice from the menu (i.e., they did not select the "Auto Attendant" choice), we will need the call to go to the most appropriately skilled agent.

To accomplish this, the Request Skill or Service blocks we configured for each Menu block choice are configured as Wait Conditions in the Find Agent block. The *Wait Condition* allows the system to find the most appropriately skilled agent for the interaction; however, if the skill condition is not met within the Wait time, the caller will be routed to the next skill group, and ultimately, any available agent.




Find Agent block configuration

## Action 4: Connect the Call

After an agent has been found, the call is connected in the[Connect Call](#) block. For more information about Find Agent and Connect Call, see [How to Create a Basic Scenario](#). As a reminder, define all [conditional exits](#).

1. REDIRECT [5.3:Scenario-builder-reference-guide/Exercises/HowtoCreateaVoiceScenarioSurvey](#)