

## 5.3 All Scenario Exercises

### Bright Pattern Documentation

Generated: 7/06/2022 10:03 pm

Content is available under license unless otherwise noted.

## Table of Contents

Table of Contents	2
How to Configure a Chat Scenario That Uses a Microsoft Azure Web App Bot	3
Procedure	3
Step 1: Import and edit example scenario	3
Step 2: Add secret key in Set Variable	3
Step 3: Authenticate to Direct Line API 3.0	4
Step 4: Add token to Set Variable	5
Step 5: Start a conversation	6
Step 6: Send a message to the customer	7
Step 7: Prompt for a customer response or wait	8
Step 8: POST activities	9
Step 9: Retrieve activities	11
Step 10: Set conditions for routing bot answers	13
Step 11: Pass along the bot's answer to the customer	13
Step 12: Get the next bot answer	14
Step 13: Use Goto to request input again	15
Step 14: Keep checking for bot answers	15
Step 15: End the conversation	16
Step 16: Save	17
Step 17: Configure your chat scenario entry to use the scenario	17
Sending Automatic Email Replies to Customers Who Use the "Leave a Message" Chat Form	18
Prerequisites	19
Scenario Example	19
Example Flow	19
Scenario Overview	20
Action 1: Configure the "Leave a Message" Chat Form in the Chat Widget Configuration Application	20
Action 2: Configure an If Block to Perform an After-Hours Check	21
Action 3: Add the EMail Block to the If Block's After-Hours Check Branch	22
Action 4: Configure Find Agent / Connect Chat	23
Skill-Based Call Routing with an Auto Attendant Choice	24
Scenario Example	24
Scenario Flow	24
Scenario Overview	24
Action 1: Create a Menu with Skill-Based Choices and an Auto Attendant Choice	25
Action 1a: For Skill-Based Menu Choices, Use the Request Skill or Service Block	26
Action 2: Begin Configuring Auto Attendant with Collect Digits	27
Action 2a: Use Set Variable to Alter Information from Collect Digits	27
Action 2b: Have Connect Call Receive the Altered Information from Set Variable	28
Action 3: Configure Skill-Based Routing in Find Agent	29
Action 4: Connect the Call	30

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.

#### Set Variable

Scenario has variables. The block sets a variable to a value. You may use  $\$(VarName)$  in the value field to substitute other variable values.

Variable name:	<input type="text" value="Secret key"/>
Value:	<input type="text" value="b2F0p582Ytc.sXoltIHBjib0hINNxoQ8hYqt-WQT_RDO_GVEQYfv-h4"/>

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

Fetches web content from specified URL, using specified method.  
Total result body size is subject to administrative limit

Title text:	<input type="text" value="Get token"/>
Request type:	POST <input type="button" value="v"/> select or write-in
URL to fetch:	<input type="text" value="https://directline.botframework.co"/>
Extra Headers:	Authorization : Bearer \$(secret) <a href="#">add</a>
URL parameters:	<a href="#">add</a>
Content Type:	application/json <input type="button" value="v"/>
Body:	<div style="border: 1px solid #ccc; height: 150px;"></div>
Username:	<input type="text"/>
Password:	<input type="text"/>
Initial path in the result JSON:	<input type="text" value=""/> e.g. myobject.node.list[4]
Scenario variable prefix for JSON data:	<input type="text" value="jstoken"/> e.g. jsdata, used as jsdata.object

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)

## Set Variable

Scenario has variables. The block sets a variable to a value. You may use  $\$(VarName)$  in the value field to substitute other variable values.

Variable name:	<input type="text" value="token"/>
Value:	<input type="text" 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

Fetches web content from specified URL, using specified method.  
Total result body size is subject to administrative limit

Title text:	<input type="text" value="Start conversation"/>
Request type:	POST <input type="button" value="select or write-in"/>
URL to fetch:	<input type="text" value="https://directline.botframework.co"/>
Extra Headers:	Authorization : Bearer \$(token) <a href="#">add</a>
URL parameters:	<a href="#">add</a>
Content Type:	application/json <input type="button" value=""/>
Body:	<div style="border: 1px solid #ccc; height: 150px;"></div>
Username:	<input type="text"/>
Password:	<input type="text"/>
Initial path in the result JSON:	<input type="text" value=""/> e.g. myobject.node.list[4]
Scenario variable prefix for JSON data:	<input type="text" value="conversation"/> e.g. jsdata, used as jsdata.object

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+

Send SMS or a message to chat customer

Media type

CHAT  SMS

Message:

Hello and welcome!

Wait for text input:

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

The block requests input.

Title text:

Prompt with

text message  form

Message to send:

Result variable name:

Timeout after, sec:

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 `/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](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

Fetches web content from specified URL, using specified method.  
Total result body size is subject to administrative limit

Title text:	<input type="text" value="Post Activities"/>
Request type:	POST <input type="button" value="v"/> select or write-in
URL to fetch:	<input type="text" value="https://directline.botframework.co"/>
Extra Headers:	Authorization : Bearer \$(token) <a href="#">add</a>
URL parameters:	<a href="#">add</a>
Content Type:	application/json <input type="button" value="v"/>
Body:	<pre>{   "type": "message",   "from": {     "id": "user1"   },   "text": "\$user_input" }</pre>
Username:	<input type="text"/>
Password:	<input type="text"/>
Initial path in the result JSON:	<input type="text"/> e.g. myobject.node.list[4]
Scenario variable prefix for JSON data:	<input type="text" value="message_id"/> e.g. jsdata, used as jsdata.object

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](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

Fetches web content from specified URL, using specified method.  
Total result body size is subject to administrative limit

Title text:	<input type="text" value="Get Activities"/>
Request type:	<input type="text" value="GET"/> <input type="button" value="v"/> select or write-in
URL to fetch:	<input type="text" value="https://directline.botframework.co"/>
Extra Headers:	Authorization : Bearer \$(token) <a href="#">add</a>
URL parameters:	<a href="#">add</a>
Content Type:	<input type="text" value="application/json"/> <input type="button" value="v"/>
Body:	<div style="border: 1px solid #ccc; height: 100px;"></div>
Username:	<input type="text"/>
Password:	<input type="text"/>
Initial path in the result JSON:	<input type="text" value="activities"/> e.g. myobject.node.list[4]
Scenario variable prefix for JSON data:	<input type="text" value="azure_bot_answer"/> e.g. jsdata, used as jsdata.object
	<input checked="" type="checkbox"/> Use GetNext block to loop through data (JSON data is an array, set scenario variable with the first element, and use GetNext block to access subsequent array elements)

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

Allows branching scenario on conditions. Can have a number of conditions with associated branches.

Title text:

[Add branch](#)

Has an answer

Exit label:

"azure\_bot\_answer.replyToId" = "\$(message\_id.id)" (string)

[add condition](#) [add block](#)

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)`

The image shows a workflow editor interface. On the left, a vertical list of blocks is visible, including 'Timeout', 'Comment "About Fetch URL"', 'Fetch URL "Post Activities"', 'Failed', 'No Data', 'Comment "About Fetch URL"', 'Fetch URL "Get Activities"', 'Failed', 'No Data', 'Comment "About If"', 'If', 'Has an answer', 'Send Message+', and 'Get Next Record'. The 'Send Message+' block is highlighted in blue. On the right, the configuration for the 'Send Message+' block is shown. It includes the title 'Send Message+', a subtitle 'Send SMS or a message to chat customer', a 'Media type' section with radio buttons for 'CHAT' (selected) and 'SMS', a 'Message:' text area containing the expression '\$(azure\_bot\_answer.text)', and a 'Wait for text input:' checkbox which is currently unchecked.

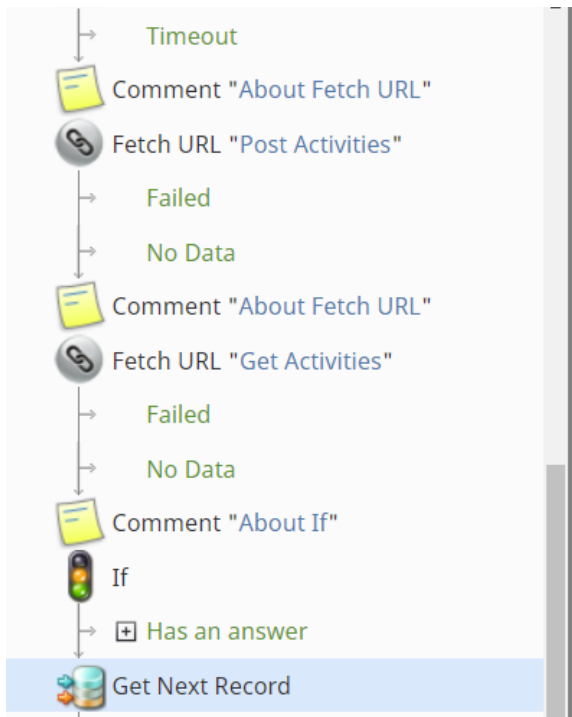
Send Message+ block being used to send the bot's answer

## 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

Retrieves the next or previous record from the recordset created by DB

The columns of the retrieved record are stored in variables <recordset\_r <column\_name>). For example, for DB Execute statement 'SELECT id, na

Title text:

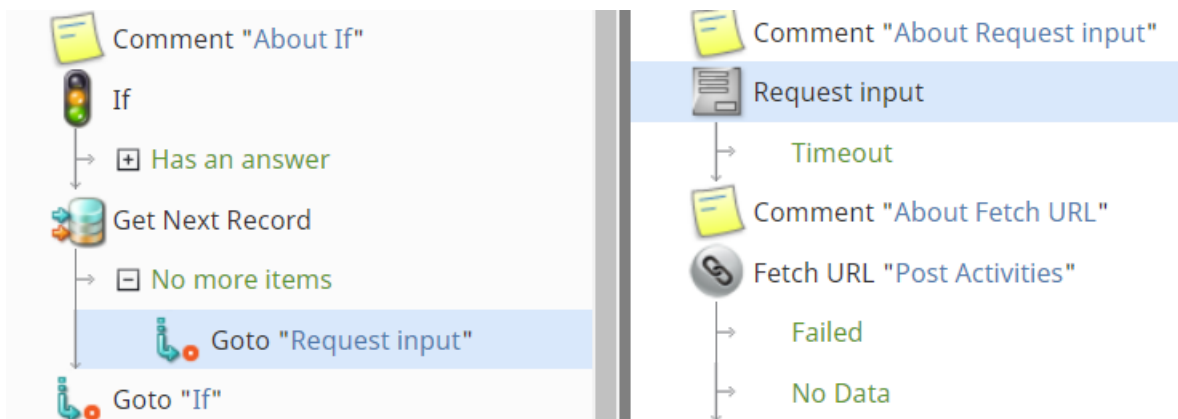
Direction:

Recordset name:

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

## Step 13: Use Goto to request input again

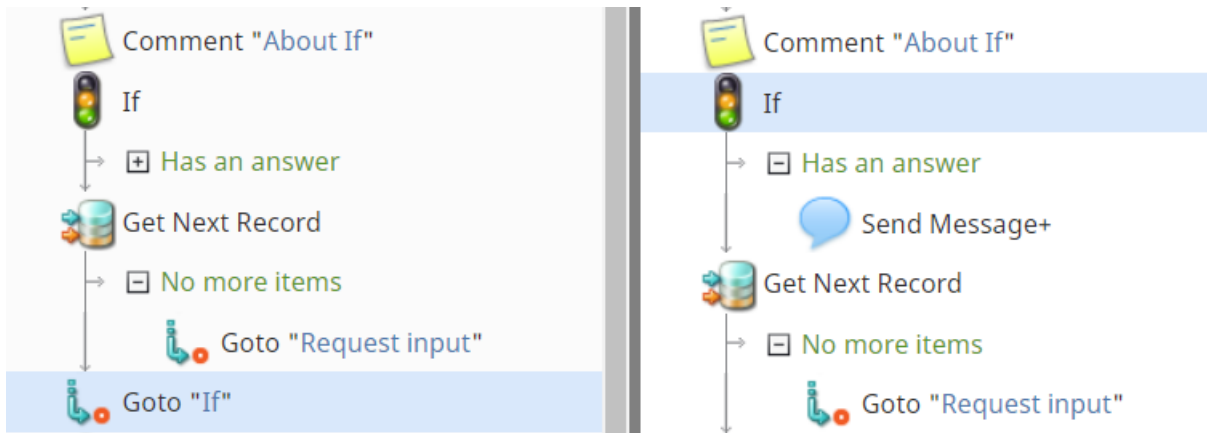
If there are no more answers from your bot, request input again by using the [Goto](#) block.



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](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

Fetches web content from specified URL, using specified method.  
Total result body size is subject to administrative limit

Title text:	<input type="text" value="End Conversation"/>
Request type:	<input type="text" value="POST"/> <input type="button" value="▼"/> select or write-in
URL to fetch:	<input type="text" value="https://directline.botframework.co"/>
Extra Headers:	Authorization : Bearer \$(token) <a href="#">add</a>
URL parameters:	<a href="#">add</a>
Content Type:	<input type="text" value="application/json"/> <input type="button" value="▼"/>
Body:	<pre>{   "type": "endOfConversation",   "from": {     "id": "user1"   } }</pre>
Username:	<input type="text"/>
Password:	<input type="text"/>
Initial path in the result JSON:	<input type="text"/> e.g. myobject.node.list[4]
Scenario variable prefix for JSON data:	<input type="text"/> e.g. jsdata, used as jsdata.object

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.

Name:	<input type="text" value="Azure Bot Chat"/>
Unique identifier:	<input type="text" value="860defca289b486387eebe9e0ecf8b"/>
Scenario:	<input type="text" value="Azure Web App Bot Example"/> <a href="#">add/edit</a>
Service:	<input type="text" value="Azure Bot Chat"/> <a href="#">add/edit</a>
Default service for voice callback:	<input type="text" value="Customer Service"/> <a href="#">add/edit</a>

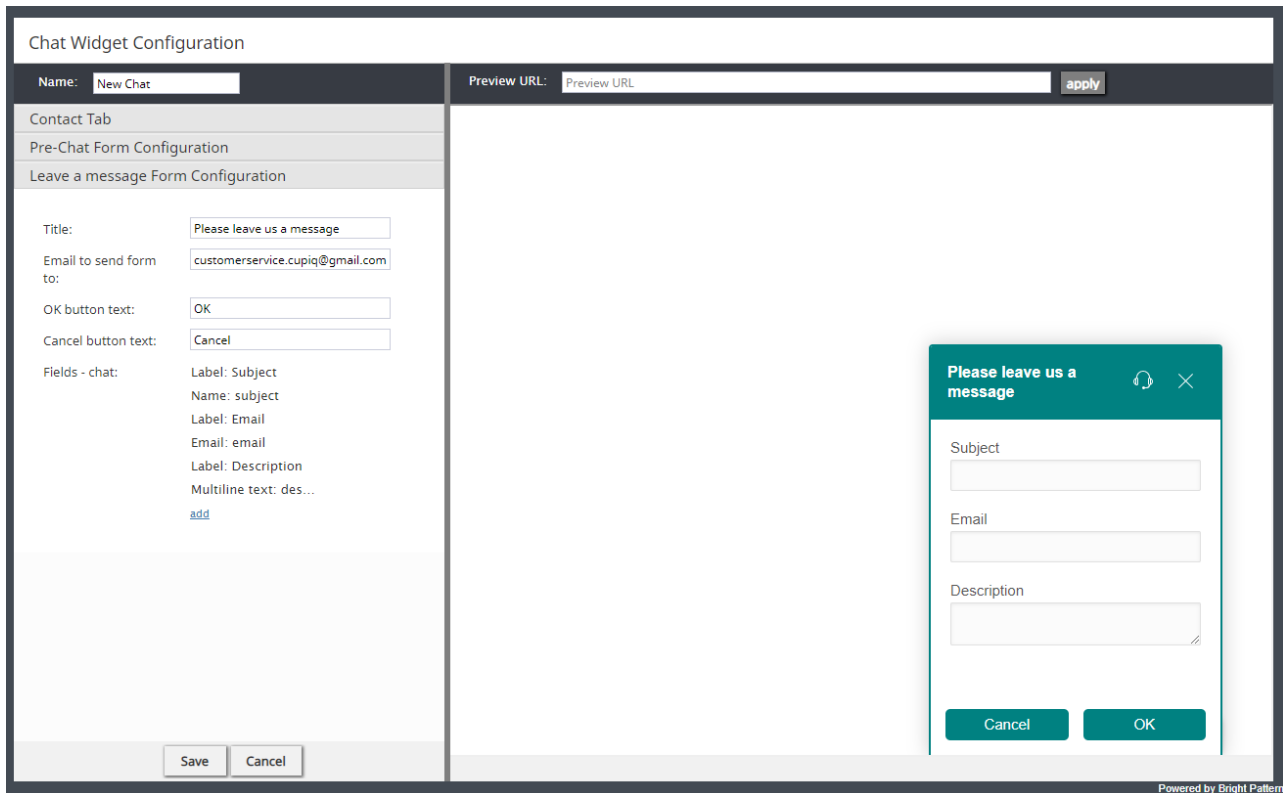
### Messaging/Chat scenario entry properties

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

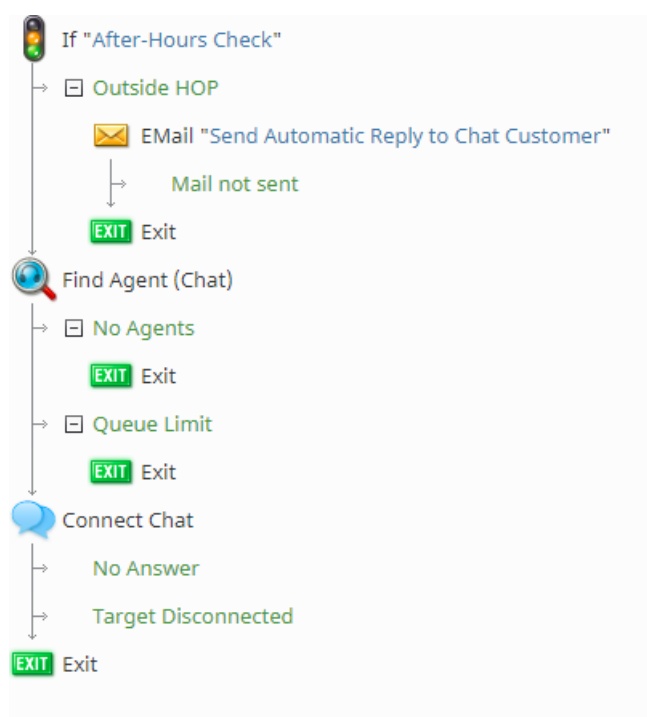
This example illustrates how to configure the [Leave a Message](#) form in the Chat Widget Configuration application, focusing on field variables. Then, in the Scenario Builder application, we configure an [If](#) block to perform an after-hours check. If messages are received outside of the configured 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)`).

Leave a message Form Configuration

Title:

Email to send form to:

OK button text:

Cancel button text:

Fields - chat: Label: Subject  
Name: subject  
Label: Email



Type:

Label text:

Name:

Rows:

Required:

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.

If

Allows branching scenario on conditions. Can have a number of conditions with associated branches.

Title text:

[Add branch](#)

Exit label:

is not

[add condition](#) [add block](#)

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)*.

## E-Mail

Title text:

### From

Display name:

Address:

### To

Address(es):

URL of the recording to attach (optional):

[add](#)

Template:  ▼

### Message

Subject:

Format:  ▼

**A** **tt** **B** **U** **A**  **Tx**  **Insert \$()** **Survey link**

Hello, and thanks for reaching out to us today! Unfortunately, we are closed. Our normal operating hours are Monday through Friday, 8 a.m. to 5 p.m.

You may try reaching us at the following:

Your Business Customer Service Phone: (555) 555-5555

Your Business Customer Service Email: your.email.address@example.com

Thanks very much and have a great day!

Your Contact Center

**Remove English template**

**Test English template**

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/howtoblacklistspecificphonenumber](#)s
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

Menu plays a prompt with choices and expects a button to be pushed, from a set of valid choices. It branches scenario execution based on the choice made.

Prompt to play:

[Top Main Menu](#)

Thank you for calling Cup IQ, for Customer Service press 1, for Accounting press 2, for Shipping and Logistics press 3. If you know your party's extension and would like to use our auto attendant press 4, otherwise stay on the line and we'll be with you shortly.;

Labels for block exits:

Valid choices:

- |                                     |   |                        |
|-------------------------------------|---|------------------------|
| <input checked="" type="checkbox"/> | 1 | Customer Service       |
| <input checked="" type="checkbox"/> | 2 | Accounting             |
| <input checked="" type="checkbox"/> | 3 | Shipping and Logistics |
| <input checked="" type="checkbox"/> | 4 | Auto Attendant         |
| <input type="checkbox"/>            | 5 |                        |
| <input type="checkbox"/>            | 6 |                        |
| <input type="checkbox"/>            | 7 |                        |
| <input type="checkbox"/>            | 8 |                        |
| <input type="checkbox"/>            | 9 |                        |
| <input type="checkbox"/>            | 0 |                        |
| <input type="checkbox"/>            | # |                        |
| <input type="checkbox"/>            | * |                        |

Invalid input prompt:

[select](#)

Allow interrupting prompt by a phone button

Ignore invalid choices

Input timeout:

seconds

Timeout prompt:

[select](#)

Short version of main prompt to play after invalid input and timeout prompts:

[Second Prompt](#)

For Customer Service press 1, for Accounting 2, for Shipping and Logistics 3, for our auto attendant 4, or stay on the line for all other needs.;

Retries:

## 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

Use this block to select the required skills for agent selection. The minimum skill level thresholds are set in Find Agent block.

Reset prior skill requirements

Specify a skill to be used for finding an agent later.

Customer Service = Customer Service

## 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

Play prompts and get a string of digits from the caller. Caller dials the string of digits by pushing buttons on their phone.

Title text: Agent Extension

Prompt to play: [Enter Extension](#)  
Enter the two-digit extension of the party you are trying to reach, then press the pound key. To clear the entry, press the star key;

Max number of digits to expect (empty=infinite): 5

Finish input button: #

Name of the variable to store the result: extension

Retries (empty=none): 0

### Details

Short version of main prompt to play after timeout: [select](#)

Clear input digit: \*

Abort input digit: None

Timeout Before First Digit is Dialed (sec): 3

Timeout Between Digits (sec): 2

## 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 beginning 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

Scenario has variables. The block sets a variable to a value. You may use  $\$(VarName)$  in the value field to substitute other variable values.

Variable name:	<input type="text" value="extension"/>
Value:	<input type="text" value="10\$(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

Connects call to a number specified in the "Destination" variable. Keeps both sides of the connection and does not exit until either side hangs up.

Title text:

### Destination

Call is connected to the number specified in the "Destination" variable

Default:

Destination: (If Destination is not set)

Override:

Destination: (block uses this number regardless of Destination value)

Mark all calls connected by this block as overflow calls

### Options

Override calling party name with:

No Answer Timeout:  seconds

Auto-answer call in:  seconds (leave empty to turn auto-answer off)

Escape button for customer to hang up the agent:

### Misc. Prompts

Custom hold music: [select](#)

Service announcement: [select](#)

number of plays:

stop announcement button:

Custom ringback: [select](#)

### Seamless Connection to an External Destination

Repeated answer-side prompt: [select](#)

Confirm answer button:

Drop connection if no answer in, min:

## 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

The block finds the best matching agent to answer the call according to requested skills. Agent's phone number is saved to "Destination" variable. Use Connect Call block to connect to the found agent.

Title text:

### Agent skills required

Intervals within a waiting call (last interval's end time may be left blank)

Wait time	Wait condition
0 - 2 sec	Customer Service >= 50
2 - 4 sec	Shipping and Logistics >= 50
4 - 5 sec	Accounting >= 25

[add Interval](#)

Overflow call handling starts at:  2 s  4 s  Never

Escape button:

### Keep call in queue

Keep the call in queue even if there are no logged in agents that match.

### Virtual Queue option

enable if EWT is greater than  seconds  
virtual Queue must also be enabled per Service in properties page of Service

Callback button:

### Prompts

Initial Prompt: [select](#)

EWT Announcement: [select](#)

Virtual Queue availability announcement: [select](#)

Periodic Reminder: [select](#)  
at  seconds, repeat every  seconds

Other reminders: [add](#)

### Music on hold

[select](#)  
 Keep playing hold music while ringing on agent  
 Play random segment

## 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](#)

