Contents

What is Bot Framework Composer? 2

Insurance Bot 4

Bot Framework

What is Bot Framework Composer?

Bot Framework Composer, built on the Bot Framework SDK, is an open-source IDE for developers to author, test, provision, and manage conversational experiences. It provides a powerful visual authoring canvas enabling dialogs, language-understanding models, QnAMaker knowledge bases, and language generation responses to be authored from within one canvas and crucially, enables these experiences to be extended with code for more complex tasks such as system integration. Resulting experiences can then be tested within Composer and provisioned into Azure along with any dependent resources.

Composer is available as a desktop application for Windows, macOS, and Linux. If the desktop app isn't suited to your needs, you can build Composer from source or host Composer in the cloud.

Authoring dialog experiences with a visual designer is more efficient and enables easier modeling of more sophisticated conversational experiences where context switching, interruption, and more natural and dynamic conversation flows are important. More complex activities such as integrating with dependencies such as REST Web Services are best suited towards code and we provide an easy mechanism to extend Composer bots with code bringing the best of both together.

Composer is a visual editing canvas for building bots. With Composer, you can:

- Create a new bot using a template, which incorporates the Virtual Assistant capabilities directly into Composer.
- Add natural language understanding capabilities to your Bot using LUIS and QnA and FAQ capabilities using QnA Maker.
- Author text and if needed speech variation responses for your Bot using language generation templates.
- Author bots in multiple languages.
- Test directly inside Composer using embedded Web Chat.
- Publish bots to *Azure App Service* and *Azure Functions*.
- Extend Power Virtual Agents with Composer (*Preview*).
- Integrate external services such as QnA Maker knowledge base.

Beyond a visual editing canvas, you can use Composer to do the following:

- Import and export dialog assets to share with other developers.
- Package manager provides a range of reusable conversational assets and code built by Microsoft and third parties. These assets can quickly add functionality to your project.

- Make any Bot available as a Skill for other Bots to call.
- Connect to a skill.
- Extend the dialog authoring canvas with Create custom actions.
- Integrate Orchestrator, which is an advanced transformer model-based router that can delegate from a parent bot to skills based on a user's utterance.
- Host Composer in the cloud.
- Extend Composer with plugins.

Key terms used in the hackathon:

Adaptive Dialogs

Dialogs provide a way for bots to manage conversations with users. Adaptive dialogs and the event model simplify sophisticated conversation modeling enabling more natural, dynamic conversation flow, interruption, and context switching. They also help you focus on the model of the conversation rather than the mechanics of dialog management. Read more in the dialog concept article.

Language Understanding

Language understanding is a core component of Composer that allows developers and conversation designers to train language understanding models directly in the context of editing a dialog. As dialogs are edited in Composer, developers can continuously add to their bots' natural language capabilities using the .lu file format, a simple Markdown-like format that makes it easy to define new intents and entities, and provide sample utterances. In Composer, you can use regular expression, LUIS, and Orchestrator recognizers. Composer detects changes and updates the bot's cloud-based natural language understanding model automatically so it's always up to date. Read more in the language understanding concept article.

Language Generation

Creating grammatically correct, data-driven responses that have a consistent tone and convey a clear brand voice has always been a challenge for bot developers. Composer's integrated bot response generation allows developers to create bot replies with a great deal of flexibility, using the editor in the Bot Responses page or the response editor in the Properties pane. Read more in the language generation concept article.

Bot Framework Emulator

Creating grammatically correct, data-driven responses that have a consistent tone and convey a clear brand voice has always been a challenge for bot developers. Composer's integrated bot response generation allows developers to create bot replies with a great deal of flexibility, using the editor in the **Bot Responses** page or the **response editor** in the **Properties** pane. Read more in the language generation concept article.

Insurance Bot

You can create a bot from the template to get quickly started. To create a bot from the template

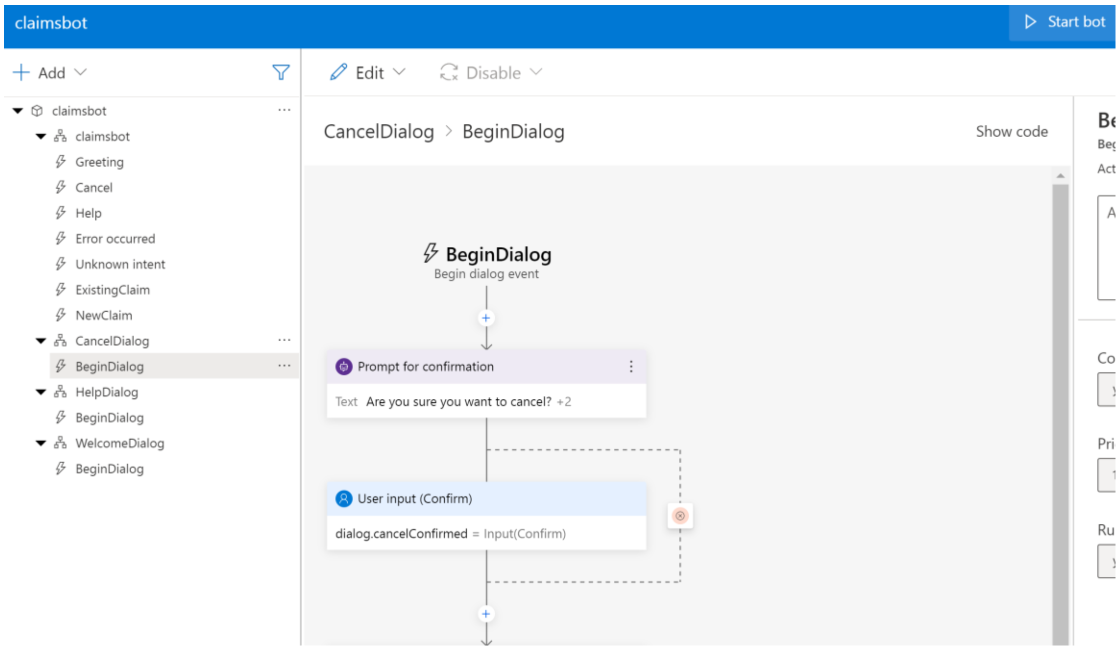
1. Open Composer.
2. Select **Create New** (+) on the homepage.
3. A list of templates is then shown which provides a starting point for your new bot. For the purposes of this quick-start, select the **Empty bot** template under the **C#** section. This template creates a bot containing a root dialog, initial greeting dialog, and an unknown intent handler. Then select **Next**.
4. Fill in the **Name** for the bot as *Menu_bot*. Then select Azure Web App from the **Runtime type**, and a location for your bot on your machine.
5. Select **OK**. It will take a few moments for Composer to create your bot from the template

Once the bot is created you can add following functionality.

Now you can add functionality for intents that you want your bot to recognize.

Dialogs are a convenient way to organize conversational logic. You add a trigger to the bot's main dialog to call the new dialog.

- Click the three dots next to your bot project, Menu_bot, and select + Add a dialog.
- In the Name field, enter Menu, and in the Description field, enter A dialog that shows the menu.. Then select OK.
- Select BeginDialog underneath the Menu dialog.
- In the authoring canvas, select the + button under BeginDialog. Then select Send a response.



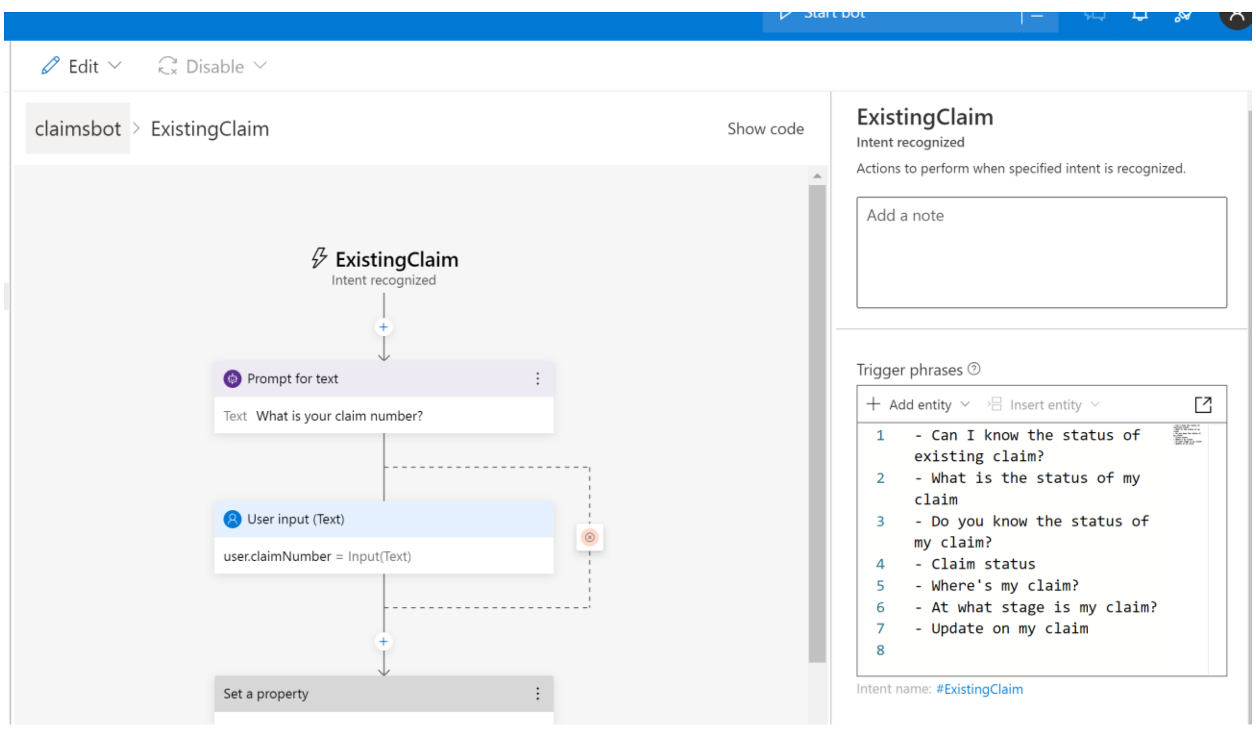
Text Are you sure you want to cancel? +2

dialog.cancelConfirmed = Input(Confirm)

You need to create then the trigger so that your bot recognizes when users want the menu item displayed.

- Select your bot project, **Menu_bot**.
- On the right, select the **Regular expression recognizer** for the **Recognizer type**. This recognizer provides a simple example without adding natural language understanding. For a real-world bot, explore language understanding.
- Now, add a trigger to the **Menu_bot** dialog.
- The default trigger type is **Intent recognized**.
- Enter Menu in the **What is the name of this trigger (RegEx)** field.
- Enter menu in the **Please input regEx pattern** field.
- Select **Submit**. Composer creates a trigger, named **Menu**, which will fire when the user sends a *menu* message to the bot.
- Now you need to start the **Menu** dialog you created previously from the trigger. Select the **+** under the **Menu** trigger on the authoring canvas. Go to **Dialog management** and select **Begin a new dialog**.

Go to the right and select the box underneath **Dialog name**. You should see the **Menu** dialog you created previously; select it. Your authoring screen should look like the following:


claimsbot > ExistingClaim

ExistingClaim
Intent recognized

Prompt for text
Text What is your claim number?

User input (Text)
user.claimNumber = Input(Text)

Set a property

ExistingClaim
Intent recognized
Actions to perform when specified intent is recognized.

Trigger phrases

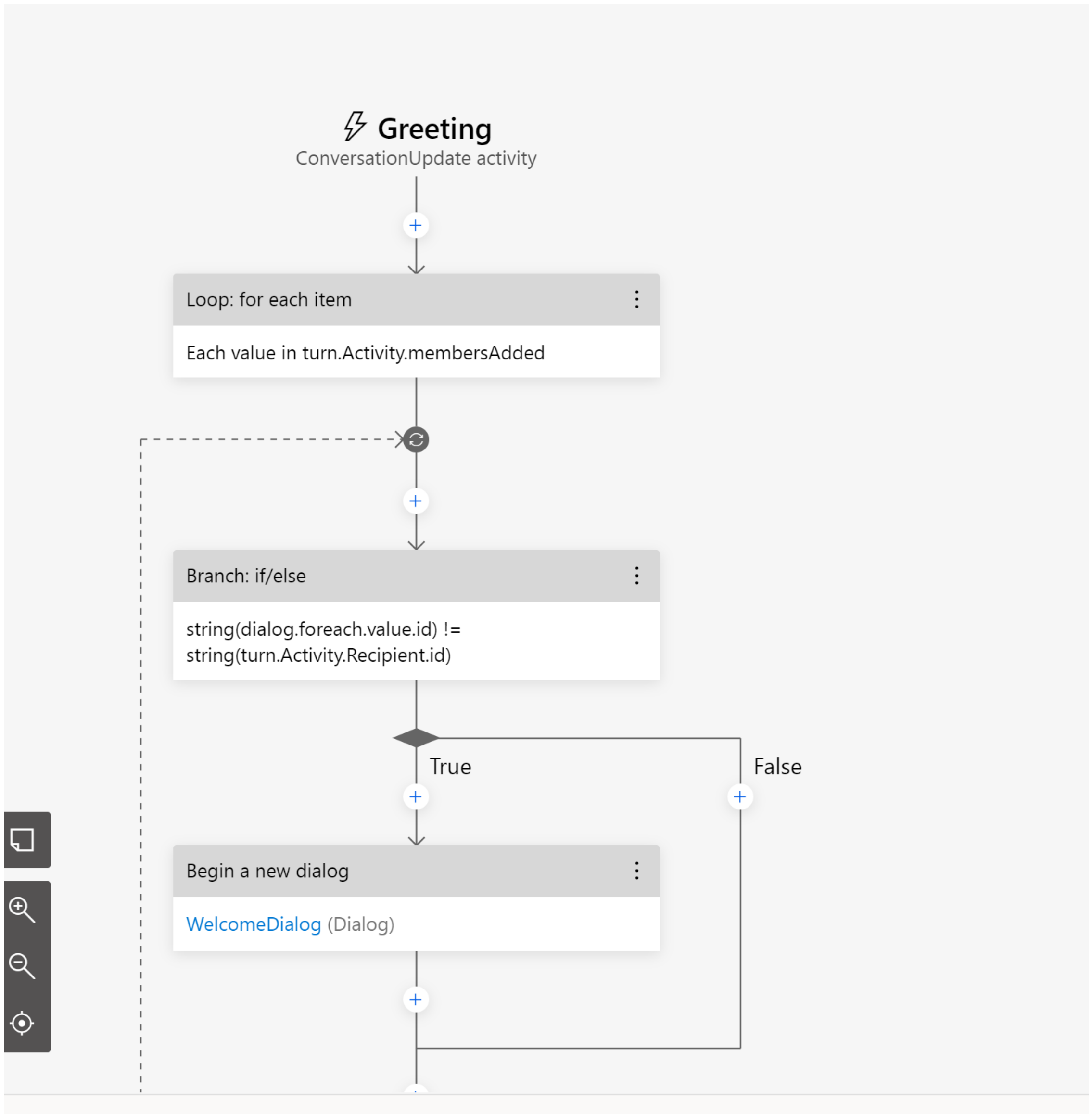
```
- Can I know the status of existing claim?
- What is the status of my claim
- Do you know the status of my claim?
- Claim status
- Where's my claim?
- At what stage is my claim?
- Update on my claim
```

Intent name: #ExistingClaim

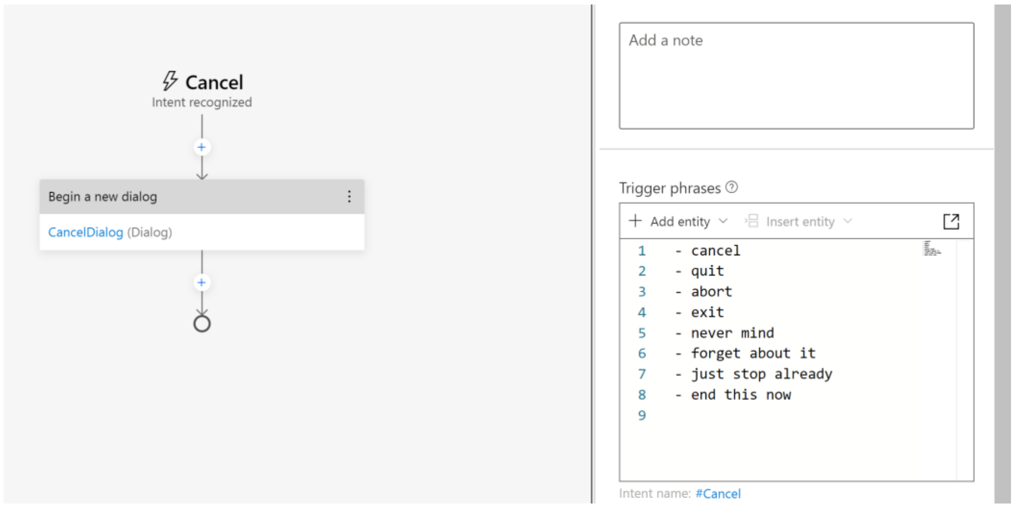
For our hackathon we will use the existing bot that is created ahead of the time.

Following are the main component for the bot :

- Bot consist of the Greetings trigger that is called initially (conversationupdate activity), which will display the welcome message (via the dialog)

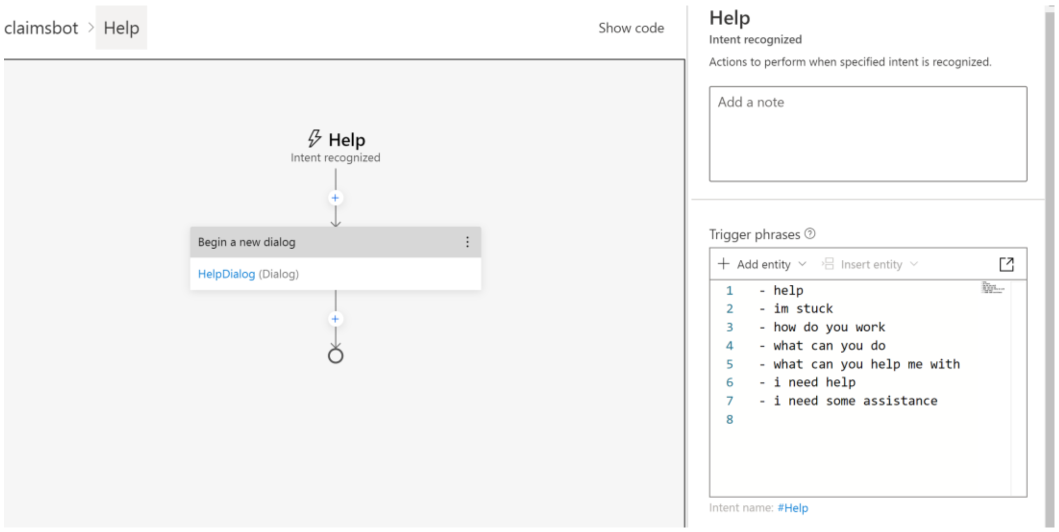

Greeting
ConversationUpdate activity
Loop: for each item
Each value in turn.Activity.membersAdded
Branch: if/else
string(dialog.foreach.value.id) != string(turn.Activity.Recipient.id)
True
False
Begin a new dialog
WelcomeDialog (Dialog)

-
- There is "Cancel" dialog created (using the LUIS NLP model) that will be recognized (intent recognition) when user is looking to cancel the conversation. Notice the trigger phase, those are the sample utterances that are used to build the NLP model

- 

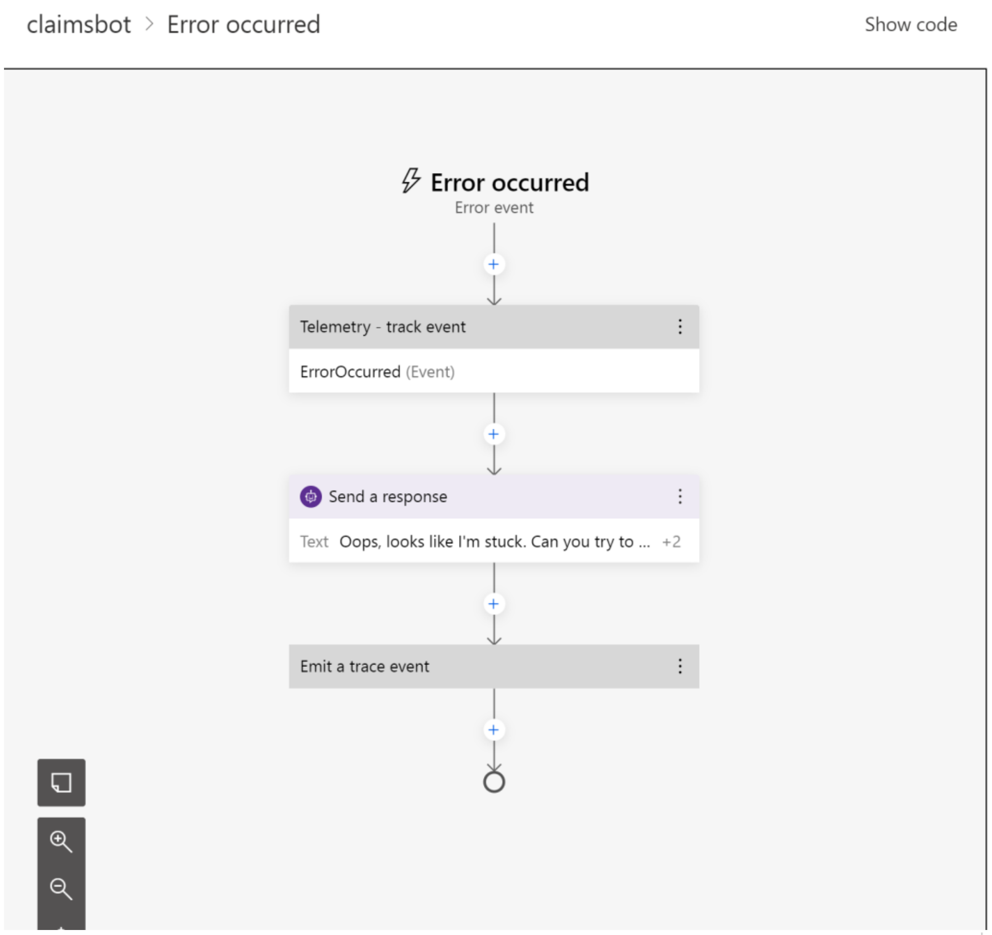
```
- cancel
- quit
- abort
- exit
- never mind
- forget about it
- just stop already
- end this now
```

Intent name: #Cancel

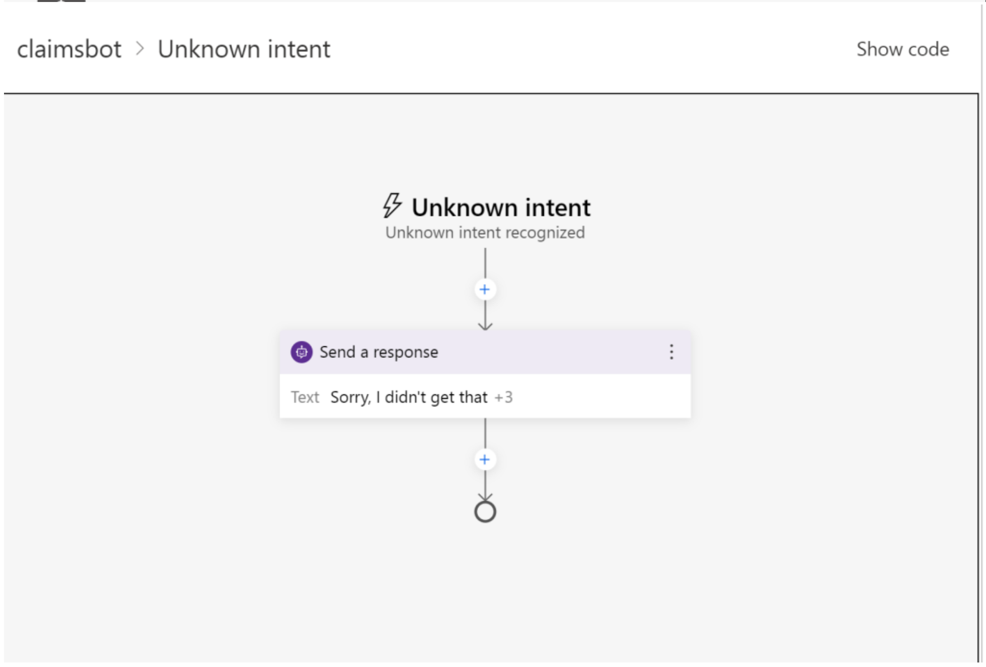
- Just like the Cancel Dialog we also have Help, Error Occurred and Unknown Intent triggers
- 

```
- help
- im stuck
- how do you work
- what can you do
- what can you help me with
- i need help
- i need some assistance
```

Intent name: #Help

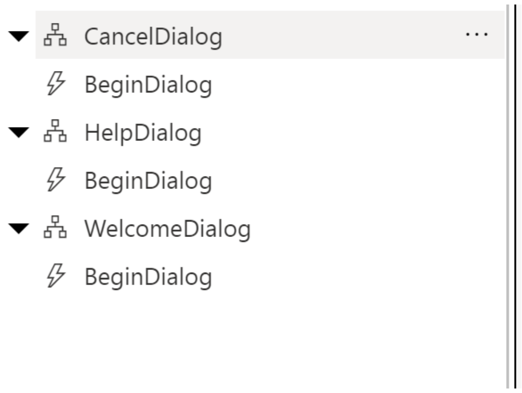
- 
claimsbot > Error occurred

Show code

- 
claimsbot > Unknown intent

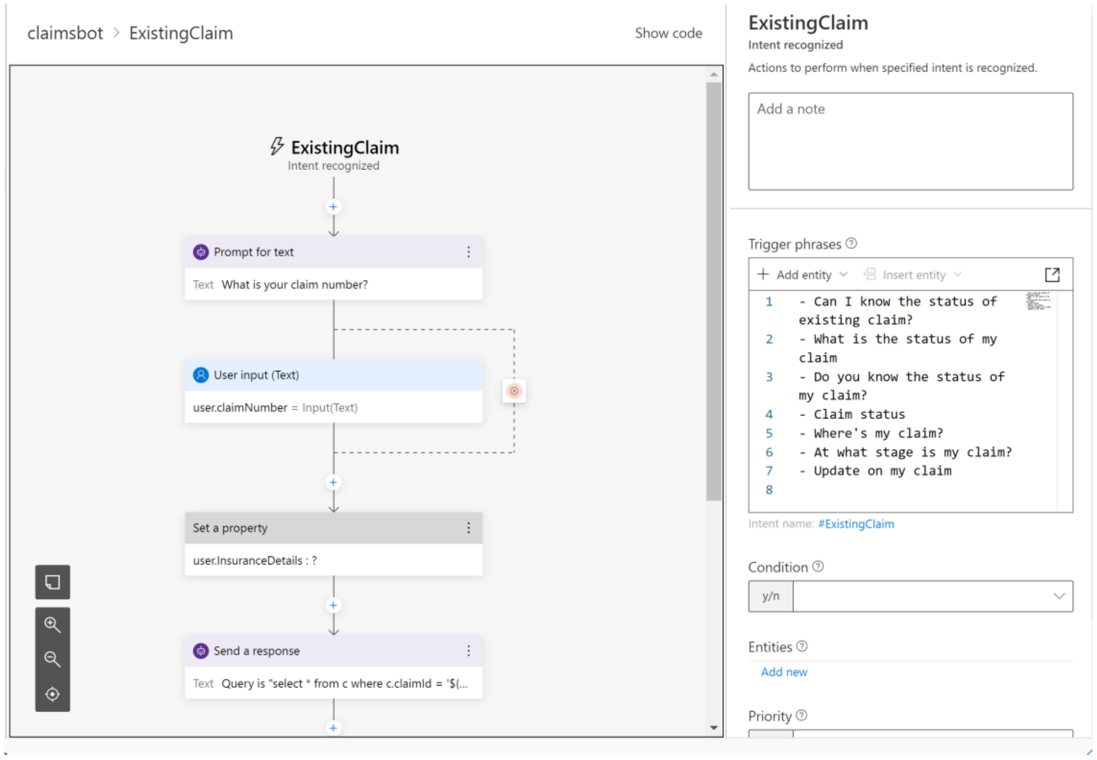
Show code

- Help, Welcome and Cancel triggers are calling appropriate Dialogs



CancelDialog
BeginDialog
HelpDialog
BeginDialog
WelcomeDialog
BeginDialog

-
- There are also custom triggers defined for New Claims and Existing Claims. That is where all the customization is built for our bot



claimsbot > ExistingClaim   Show code

ExistingClaim
Intent recognized

Prompt for text
Text What is your claim number?

User input (Text)
user.claimNumber = Input(Text)

Set a property
user.InsuranceDetails : ?

Send a response
Text Query is "select * from c where c.claimId = '${...

ExistingClaim
Intent recognized
Actions to perform when specified intent is recognized.

Add a note

Trigger phrases

```
- Can I know the status of existing claim?
- What is the status of my claim
- Do you know the status of my claim?
- Claim status
- Where's my claim?
- At what stage is my claim?
- Update on my claim
```

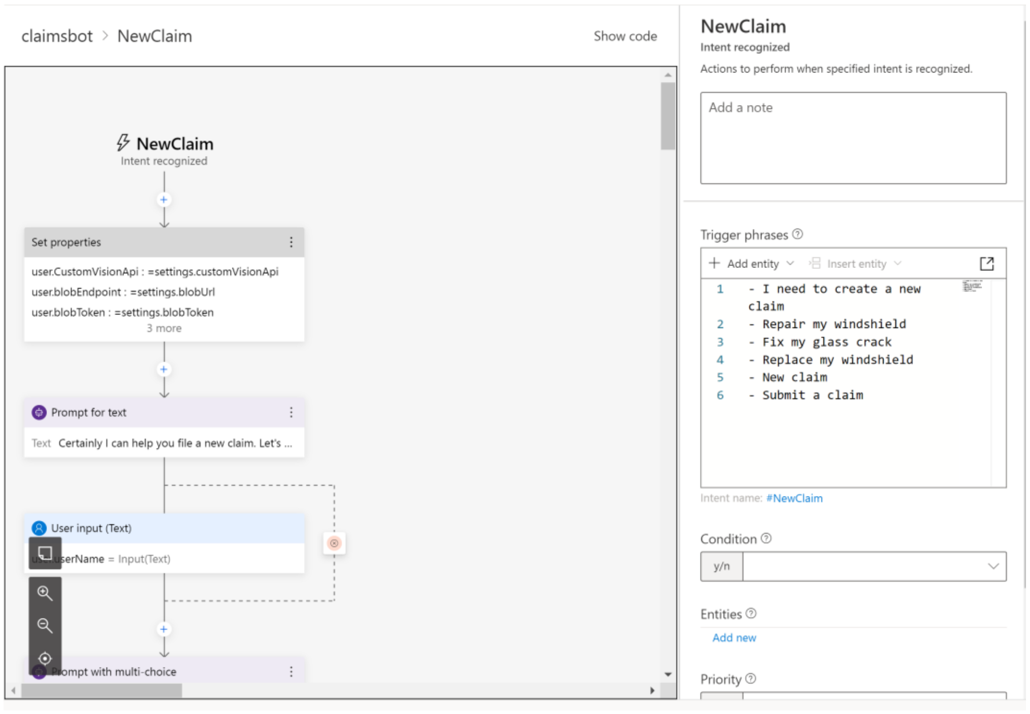
Intent name: #ExistingClaim

Condition
y/n

Entities
Add new

Priority

-



user.CustomVisionApi : =settings.customVisionApi
user.blobEndpoint : =settings.blobUrl
user.blobToken : =settings.blobToken
3 more

Text Certainly I can help you file a new claim. Let's ...

userName = Input(Text)

NewClaim
Intent recognized
Actions to perform when specified intent is recognized.

Add a note

Trigger phrases

```
- I need to create a new
claim
- Repair my windshield
- Fix my glass crack
- Replace my windshield
- New claim
- Submit a claim
```

Intent name: #NewClaim

Condition
y/n

Entities

Priority

-
- For the new claims, following is the flow
  - Set the appropriate properties that we will use throughout the bot (that are read for the application settings)
  - Prompt user for name and store that in a user variable (memory)
  - Provide the choice input to user on type of claim that they want to file (Windshield, Accident, Home Repair)
  - Switch condition based on the type of the claim user is looking to file
    - For Windishield claims it will go through series of questions asking to provide Damage of windshield, Insurance Card, Driving License and Service Estimate. Damage of windshield calls he custom model we created earlier to find the type of damage and all other documents are uploaded to storage account.
    - As soon as document is uploaded to the storage account, it will trigger the Logic app workflow that we built
    - Upon completion of all document request, Bot will display the claimId back to end user