Mission 2: Connecting to a bot

We just deployed the alien bot we found on Mars. There's a problem though. The bot has it's own special set of parameters and communication methods. As a result, we need to craft a class which will help us in connecting to the bot. We need to establish the right functions in our new class for use in our application later.

Introduction

In this mission, we will be connecting to the bot. Bots built using the Microsoft Bot Framework have a Direct Line channel. This channel is essentially a REST API that allows us to interact with the bot using HTTP requests and responses.

Although it is possible to setup your own classes and models for accessing the REST API, it can be troublesome. Luckily, there is a Nuget package published by Microsoft for use with .NET projects which helps us abstract the REST API calls. The Nuget has already been installed for you in this project.

Connecting the bot

Our application needs to communicate with the bot in 2 ways: Sending and receiving messages. As a result, we will need to use methods to send messages to the bot as well as constantly receive messages from the bot.

The DirectLine nuget we installed will allow us to more easily implement this within our application without having to write any HTTP calls manually. Let's make a new class where we will contain and abstract all the connection-related code.

Right click on the main project and go to Add > New Item > Class and name it BotConnection.cs.

Imports

We need to import the relevant namespace from the DirectLine nuget package in order to use its classes and methods.

```
using Microsoft.Bot.Connector.DirectLine;
```

Fields

Let's define a few new variables in our new blank class.

```
class BotConnection
{
    public DirectLineClient Client = new DirectLineClient("Place key here");
    public Conversation MainConversation;
    public ChannelAccount Account;
}
```

First, we create a new DirectLineClient object using a DirectLine key (taken from the bot's portal in Mission 1).

Then, there are another 2 variables for storing the current conversation as well as the user's account, which we will define later.

Constructor

We will be using the constructor of this class to initialize the MainConversation and Account fields. These fields will store the information about the current conversation and user.

```
class BotConnection
{
    public BotConnection(string accountName)
    {
        MainConversation = Client.Conversations.StartConversation();
        Account = new ChannelAccount { Id = accountName, Name = accountName };
    }
```

```
}
```

In this constructor, we can take in a parameter with the user's name to create an account object. We will also start a new conversation using the client.

Message sending method

Next, we need to craft a method that allows us to send messages to the bot.

```
public void SendMessage(string message)
{
    Activity activity = new Activity
    {
        From = Account,
        Text = message,
        Type = ActivityTypes.Message
    };
    Client.Conversations.PostActivity(MainConversation.ConversationId, activity);
}
```

This method will take in a parameter with a simple text message and use that to create an Activity that will be sent to the conversation we initialized.

Message receiving method

MessageListItem Class

We will need to create a class that will be used to store the messages in an ObservableCollection.

This class will later be used for data binding in the UI.

Right click on the main project and go to Add > New Item > Class and name it MessageListItem.cs.

Change the class to look like this:

```
class MessageListItem
{
    public string Text { get; set; }
    public string Sender { get; set; }

    public MessageListItem(string text, string sender)
    {
        Text = text;
        Sender = sender;
    }
}
```

Method implementation

Lastly, we need to have a method that continuously checks the conversation on the server for new messages from the bot.

```
public async Task GetMessagesAsync(ObservableCollection<MessageListItem> collection)
{
    string watermark = null;

    //Loop retrieval
    while(true)
    {
```

```
        Debug.WriteLine("Reading message every 3 seconds");

        //Get activities (messages) after the watermark
        var activitySet = await Client.Conversations.GetActivitiesAsync(MainConversation.ConversationId,
watermark);

        //Set new watermark
        watermark = activitySet?.Watermark;

        //Loop through the activities and add them to the list
        foreach(Activity activity in activitySet.Activities)
        {
            if (activity.From.Name == "MarsBot")
            {
                collection.Add(new MessageListItem(activity.Text, activity.From.Name));
            }
        }

        //Wait 3 seconds
        await Task.Delay(3000);
    }
}
```

This method takes in an ObservableCollection typed parameter. This collection will later be binded to the UI in Xamarin, so we will need to push any new messages into this collection.

In this method, it checks for new messages every 3 seconds, establishing a watermark every iteration to ensure that we do not retrieve old messages. Whenever we retrieve a new message, we create a new MessageListItem from it and push it into the collection.

In the next mission, we will look into utilizing the class we crafted here.