Bot Builder v4 HOL

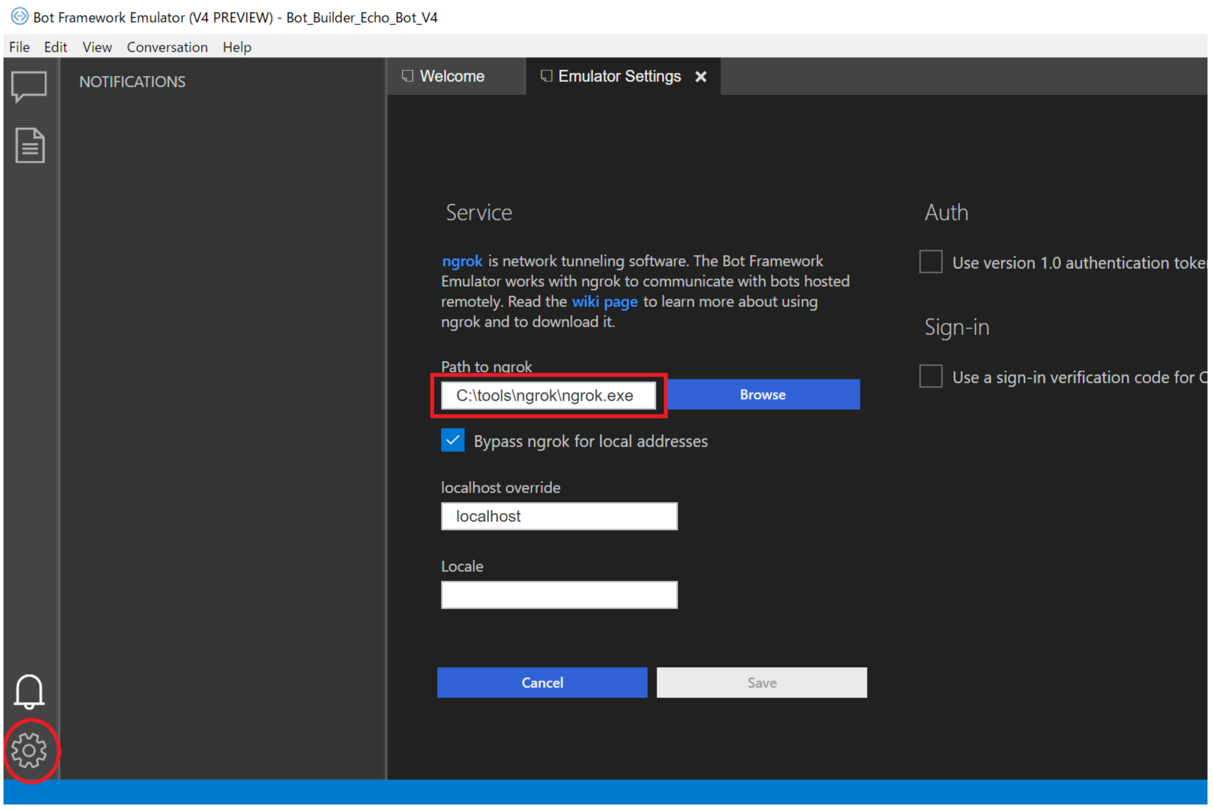
Chapter 1: Preparation

- Visual Studio 2017
- .Net Core 2.x (https://www.microsoft.com/net/download)
- Bot Builder V4 SDK Template for Visual Studio (https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4)
- Bot Emulator (https://github.com/Microsoft/BotFramework-Emulator/releases)
- Ngrok (https://ngrok.com/)

Or

You can use Azure Bot Services .

After Download the Ngrok, please launch your Bot Emulator and go to Emulator Settings and Setup as below.


Bot Framework Emulator (V4 PREVIEW) - Bot_Builder_Echo_Bot_V4
File Edit View Conversation Help
NOTIFICATIONS
Welcome
Emulator Settings
Browse
Cancel
Save

Service

ngrok is network tunneling software. The Bot Framework Emulator works with ngrok to communicate with bots hosted remotely. Read the wiki page to learn more about using ngrok and to download it.

Path to ngrok: C:\tools\ngrok\ngrok.exe

[✓] Bypass ngrok for local addresses

localhost override: localhost

Locale:

Auth

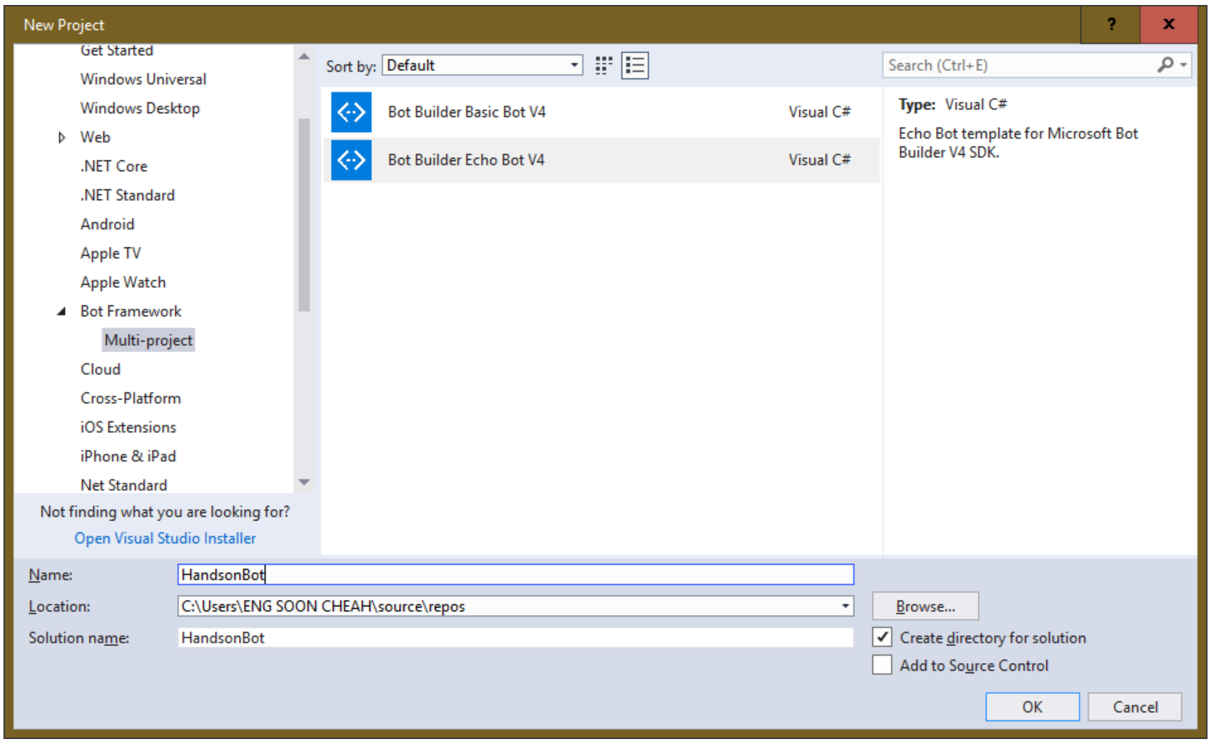
[ ] Use version 1.0 authentication toke

Sign-in

[ ] Use a sign-in verification code for O

Chapter 2: Create Project

1. Launch your Visual Studio.
2. Create a New Project, File > New > Project
3. Select **Multi-Project** and Select **Bot Builder Echo Bot v4** and Name your project, **HandsonBot** . Lastly , click **OK** button.


New Project

Get Started, Windows Universal, Windows Desktop, Web, .NET Core, .NET Standard, Android, Apple TV, Apple Watch, Bot Framework, Multi-project, Cloud, Cross-Platform, iOS Extensions, iPhone & iPad, Net Standard

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

Search (Ctrl+E)

Bot Builder Basic Bot V4	Visual C#
Bot Builder Echo Bot V4	Visual C#

Type: Visual C#

Echo Bot template for Microsoft Bot Builder V4 SDK.

Name: HandsonBot

Location: C:\Users\ENG SOON CHEAH\source\repos

Solution name: HandsonBot

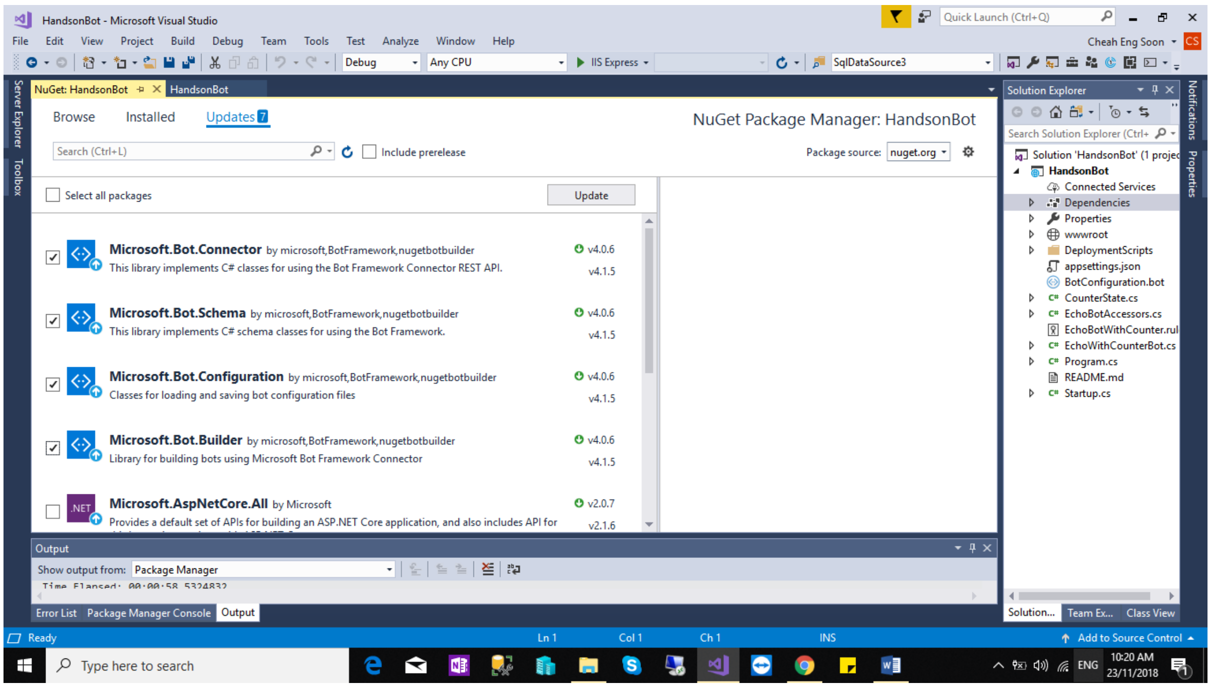
Browse...

[✓] Create directory for solution

[ ] Add to Source Control

OK Cancel

4. Make sure your bot template update to the v4.0.7


HandsonBot - Microsoft Visual Studio

NuGet Package Manager: HandsonBot

Browse Installed Updates 7

Search (Ctrl+L) Include prerelease Package source: nuget.org

Select all packages Update

✓	**Microsoft.Bot.Connector** by microsoft,BotFramework,nugetbotbuilder This library implements C# classes for using the Bot Framework Connector REST API.	v4.0.6	v4.1.5
✓	**Microsoft.Bot.Schema** by microsoft,BotFramework,nugetbotbuilder This library implements C# schema classes for using the Bot Framework.	v4.0.6	v4.1.5
✓	**Microsoft.Bot.Configuration** by microsoft,BotFramework,nugetbotbuilder Classes for loading and saving bot configuration files	v4.0.6	v4.1.5
✓	**Microsoft.Bot.Builder** by microsoft,BotFramework,nugetbotbuilder Library for building bots using Microsoft Bot Framework Connector	v4.0.6	v4.1.5
	Microsoft.AspNetCore.All by Microsoft Provides a default set of APIs for building an ASP.NET Core application, and also includes API for	v2.0.7	v2.1.6

Solution Explorer

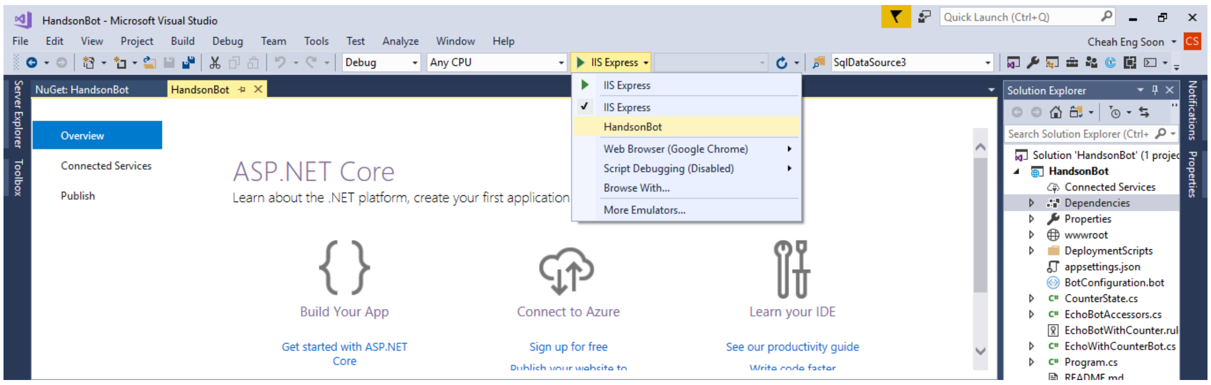
Solution 'HandsonBot' (1 project)
- HandsonBot
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - DeploymentScripts
  - appsettings.json
  - BotConfiguration.bot
  - CounterState.cs
  - EchoBotAccessors.cs
  - EchoBotWithCounter.rul
  - EchoWithCounterBot.cs
  - Program.cs
  - README.md
  - Startup.cs

Output

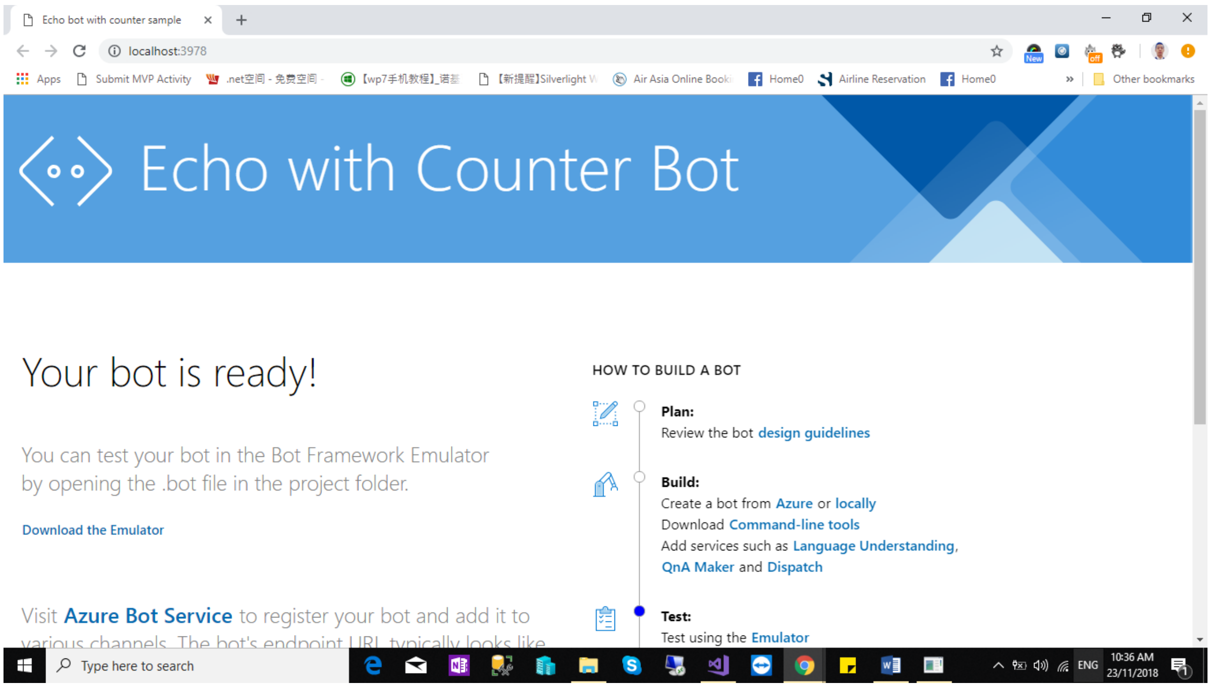
Show output from: Package Manager

```
Time Elapsed: 00:00:58.5324832
```

5. Test Run your project. Go to **dropdownlist of IIS Express** and Select **HandsonBot** , which is your project name and Press **F5** for Debug your project.



6. After Debug, you will see the localhost as below.



Your bot is ready!

You can test your bot in the Bot Framework Emulator by opening the .bot file in the project folder.

Download the Emulator

HOW TO BUILD A BOT

Plan:
Review the bot design guidelines

Build:
Create a bot from Azure or locally
Download Command-line tools
Add services such as Language Understanding, QnA Maker and Dispatch

Test:
Test using the Emulator

7. Launch your Bot Framework Emulator and Click **Open Bot** and Select **BotConfiguration.Bot** that in your project.

Chapter 3: Implementation of welcome message

1. Go to your project, right click > add new folder “SampleBot”.
2. After create the folder , right click > add class > name the class as “SampleBot”.
3. Implement the code as below.

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using Microsoft.Extensions.Logging;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace HandsonBot.SampleBot
{
    public class SampleBot : IBot
    {
        private const string WelcomeText = "Welcome to Sample Bot";

        private readonly ILogger _logger;
        private readonly SampleBotAccessors _accessors; // Added
        private readonly DialogSet _dialogs; // Added

        public SampleBot(SampleBotAccessors accessors, ILoggerFactory
loggerFactory) // Updated
        {
            _accessors = accessors ?? throw new
ArgumentException(nameof(accessors)); // Added

            _dialogs = new DialogSet(accessors.ConversationDialogState); // Added
            _dialogs.Add(new TextPrompt("name", ValidateHandleNameAsync));

            _logger = loggerFactory.CreateLogger<SampleBot>();
            _logger.LogInformation("Start SampleBot");
        }

        private Task<bool> ValidateHandleNameAsync(PromptValidatorContext<string>
promptContext, CancellationToken cancellationToken)
        {
            var result = promptContext.Recognized.Value;

            if (result != null && result.Length >= 3)
            {
                var upperValue = result.ToUpperInvariant();
                promptContext.Recognized.Value = upperValue;
                return Task.FromResult(true);
            }

            return Task.FromResult(false);
        }

        public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken
cancellationToken = default(CancellationToken))
        {
            if (turnContext.Activity.Type == ActivityTypes.Message)
            {
                // We will exchange normal messages here.
                await SendMessageActivityAsync(turnContext, cancellationToken); //
updated
            }
            else if (turnContext.Activity.Type ==
ActivityTypes.ConversationUpdate)
            {
                await SendWelcomeMessageAsync(turnContext, cancellationToken);
            }
            else
            {
                _logger.LogInformation($"passed:{turnContext.Activity.Type}");
            }

            await _accessors.ConversationState.SaveChangesAsync(turnContext,
false, cancellationToken);
            await _accessors.UserState.SaveChangesAsync(turnContext, false,
cancellationToken);
        }

        private static async Task SendWelcomeMessageAsync(ITurnContext
turnContext, CancellationToken cancellationToken)
        {
            foreach (var member in turnContext.Activity.MembersAdded)
            {
                if (member.Id != turnContext.Activity.Recipient.Id)
                {
                    await turnContext.SendActivityAsync(WelcomeText,
cancellationToken: cancellationToken);
                }
            }
        }

        private async Task SendMessageActivityAsync(ITurnContext turnContext,
CancellationToken cancellationToken)
        {
            var dialogContext = await _dialogs.CreateContextAsync(turnContext,
cancellationToken);
            var dialogTurnResult = await
dialogContext.ContinueDialogAsync(cancellationToken);

            var userProfile = await _accessors.UserProfile.GetAsync(turnContext,
() => new UserProfile(), cancellationToken);

            // If the handle name is not registered in UserState
            if (userProfile.HandleName == null)
            {
                await GetHandleNameAsync(dialogContext, dialogTurnResult,
userProfile, cancellationToken);
            }
            else
            {
                await turnContext.SendActivityAsync($"Hello
{userProfile.HandleName}", cancellationToken: cancellationToken);
            }
        }

        private async Task GetHandleNameAsync(DialogContext dialogContext,
DialogTurnResult dialogTurnResult, UserProfile userProfile, CancellationToken
cancellationToken)
        {
            if (dialogTurnResult.Status is DialogTurnStatus.Empty)
            {
                await dialogContext.PromptAsync(
                    "name",
                    new PromptOptions
                    {
                        Prompt = MessageFactory.Text("Please tell me your handle 
name first."),
                        RetryPrompt = MessageFactory.Text("The handle name must be 
at least 3 words long."),
                    },
                    cancellationToken);
            }
            else if (dialogTurnResult.Status is DialogTurnStatus.Complete)
            {
                // Register your handle name with UserState
                userProfile.HandleName = (string)dialogTurnResult.Result;
                _logger.LogInformation($"Handle Name registration: 
{userProfile.HandleName}");
            }
        }
    }
}
```

4. Go to Startup.cs
   Change near line 57
   From

```
services.AddBot<EchoWithCounterBot>(options =>
```

   to

```
services.AddBot<SampleBot.SampleBot>(options =>
```

   &
   Change near line 78

   From

```
ILogger logger = _loggerFactory.CreateLogger<EchoWithCounterBot>();
```

   to

```
  ILogger logger = _loggerFactory.CreateLogger<SampleBot.SampleBot>();
```

Chapter 4: State Management

1. Right Click “**SampleBot**” Folder > **right click** > **Add** > **Create UserProfile.class** and Implement the code as below.

```
namespace HandsonBot.SampleBot
{
    public class UserProfile
    {
        public string HandleName { get; set; }
    }
}
```

2. Install the Microsoft.Bot.Builder.Dialogs Nuget Package.

Go to **Tools**> **Nuget Package Manager**> **Package Manager Console** and type the command as below.

```
Install-Package Microsoft.Bot.Builder.Dialogs
```

3. Implement of State management Accessor
   Right Click “**SampleBot**” Folder > **right click** > **Add** > Create **SampleBotAccessors.class** and Implement the code as below.

```
using System;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;

namespace HandsonBot.SampleBot
{
    public class SampleBotAccessors
    {
        public IStatePropertyAccessor<DialogState> ConversationDialogState { get;
set; }

        public IStatePropertyAccessor<UserProfile> UserProfile { get; set; }

        public ConversationState ConversationState { get; }

        public UserState UserState { get; }

        public SampleBotAccessors(ConversationState conversationState, UserState
userState)
        {
            ConversationState = conversationState ?? throw new
ArgumentNullException(nameof(conversationState));
            UserState = userState ?? throw new ArgumentException(nameof(userState));
        }
    }
       }
```

4. Change Startup.cs near line 111, to implement Application of UserState Class
   From

```
var conversationState = new ConversationState(dataStore);

options.State.Add(conversationState);
```

   To

```
var conversationState = new ConversationState(dataStore);
options.State.Add(conversationState);

var userState = new UserState(dataStore);
options.State.Add(userState);
```

5. Implement of SampleBotAccessors class in Startup.cs , add the following code as below.

```
using System;
using System.Linq;
using HandsonBot.SampleBot;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Integration;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Bot.Configuration;
using Microsoft.Bot.Connector.Authentication;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

namespace HandsonBot
{
    /// <summary>
    /// The Startup class configures services and the request
pipeline.
    /// </summary>
    public class Startup
    {
        private ILoggerFactory _loggerFactory;
        private bool _isProduction = false;

        public Startup(IHostingEnvironment env)
        {
            _isProduction = env.IsProduction();
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true,
reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json"
, optional: true)
                .AddEnvironmentVariables();
            Configuration = builder.Build();
        }

        /// <summary>
        /// Gets the configuration that represents a set of key/value
application configuration properties.
        /// </summary>
        /// <value>
        /// The <see cref="IConfiguration"/> that represents a set of
key/value application configuration properties.
        /// </value>
        public IConfiguration Configuration { get; }

        /// <summary>
        /// This method gets called by the runtime. Use this method to
add services to the container.
        /// </summary>
        /// <param name="services">The <see
cref="IServiceCollection"/> specifies the contract for a collection of
service descriptors.</param>
        /// <seealso cref="IStatePropertyAccessor{T}"/>
        /// <seealso cref="https://docs.microsoft.com/en-
us/aspnet/web-api/overview/advanced/dependency-injection"/>
        /// <seealso cref="https://docs.microsoft.com/en-us/azure/bot-
service/bot-service-manage-channels?view=azure-bot-service-4.0"/>
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddBot<SampleBot.SampleBot>(options =>
            {
                var secretKey =
Configuration.GetSection("botFileSecret")?.Value;
                var botFilePath =
Configuration.GetSection("botFilePath")?.Value;

                // Loads .bot configuration file and adds a singleton
that your Bot can access through dependency injection.
                var botConfig = BotConfiguration.Load(botFilePath ??
@".\BotConfiguration.bot", secretKey);
                services.AddSingleton(sp => botConfig ?? throw new
InvalidOperationException($"The .bot config file could not be loaded.
({botConfig})"));

                // Retrieve current endpoint.
                var environment = _isProduction ? "production" :
"development";
                var service = botConfig.Services.Where(s => s.Type ==
"endpoint" && s.Name == environment).FirstOrDefault();
                if (!(service is EndpointService endpointService))
                {
                    throw new InvalidOperationException($"The .bot
file does not contain an endpoint with name '{environment}'.");
                }

                options.CredentialProvider = new
SimpleCredentialProvider(endpointService.AppId,
endpointService.AppPassword);

                // Creates a logger for the application to use.
                ILogger logger =
_loggerFactory.CreateLogger<SampleBot.SampleBot>();
```

```
                // Catches any errors that occur during a conversation
turn and logs them.
                options.OnTurnError = async (context, exception) =>
                {
                    logger.LogError($"Exception caught :
{exception}");
                    await context.SendActivityAsync("Sorry, it looks
like something went wrong.");
                };

                // The Memory Storage used here is for local bot
debugging only. When the bot
                // is restarted, everything stored in memory will be
gone.
                IStorage dataStore = new MemoryStorage();

                // For production bots use the Azure Blob or
                // Azure CosmosDB storage providers. For the Azure
                // based storage providers, add the
Microsoft.Bot.Builder.Azure
                // Nuget package to your solution. That package is
found at:
                //
https://www.nuget.org/packages/Microsoft.Bot.Builder.Azure/
                // Uncomment the following lines to use Azure Blob
Storage
                // //Storage configuration name or ID from the .bot
file.
                // const string StorageConfigurationId = "<STORAGE-
NAME-OR-ID-FROM-BOT-FILE>";
                // var blobConfig =
botConfig.FindServiceByNameOrId(StorageConfigurationId);
                // if (!(blobConfig is BlobStorageService
blobStorageConfig))
                // {
                //    throw new InvalidOperationException($"The .bot
file does not contain an blob storage with name
'{StorageConfigurationId}'.");
                // }
                // // Default container name.
                // const string DefaultBotContainer = "<DEFAULT-
CONTAINER>";
                // var storageContainer =
string.IsNullOrWhiteSpace(blobStorageConfig.Container) ?
DefaultBotContainer : blobStorageConfig.Container;
                // IStorage dataStore = new
Microsoft.Bot.Builder.Azure.AzureBlobStorage(blobStorageConfig.Connect
ionString, storageContainer);

                // Create Conversation State object.
                // The Conversation State object is where we persist
anything at the conversation-scope.
                var conversationState = new
ConversationState(dataStore);
                options.State.Add(conversationState);

                var userState = new UserState(dataStore);
                options.State.Add(userState);
            });
            // Create and register state accesssors.
            // Acessors created here are passed into the IBot-derived
class on every turn.
            services.AddSingleton<EchoBotAccessors>(sp =>
            {
                var options =
sp.GetRequiredService<IOptions<BotFrameworkOptions>>().Value;
                if (options == null)
                {
                    throw new
InvalidOperationException("BotFrameworkOptions must be configured
prior to setting up the state accessors");
                }

                var conversationState =
options.State.OfType<ConversationState>().FirstOrDefault();
                if (conversationState == null)
                {
                    throw new
InvalidOperationException("ConversationState must be defined and added
before adding conversation-scoped state accessors.");
                }

                // Create the custom state accessor.
                // State accessors enable other components to read and
write individual properties of state.
                var accessors = new
EchoBotAccessors(conversationState)
                {
                    CounterState =
conversationState.CreateProperty<CounterState>(EchoBotAccessors.Counte
rStateName),
                };

                return accessors;
            });

            // Create and register state accesssors.
            // Acessors created here are passed into the IBot-derived
class on every turn.
            services.AddSingleton<SampleBotAccessors>(sp =>
            {
                var options =
sp.GetRequiredService<IOptions<BotFrameworkOptions>>().Value
                                 ?? throw new
InvalidOperationException("BotFrameworkOptions must be configured
prior to setting up the state accessors");


                var conversationState =
options.State.OfType<ConversationState>().FirstOrDefault()
                                        ?? throw new
InvalidOperationException("ConversationState が ConfigureServices で設
定されていません。");

                var userState =
options.State.OfType<UserState>().FirstOrDefault()
```

```
                                ?? throw new
InvalidOperationException("UserState が ConfigureServices で設定されていません。");

                var accessors = new
SampleBotAccessors(conversationState, userState)
                {
                    ConversationDialogState =
conversationState.CreateProperty<DialogState>(nameof(DialogState)),
                    UserProfile =
userState.CreateProperty<UserProfile>(nameof(UserProfile)),
                };

                return accessors;
            });
        }

        public void Configure(IApplicationBuilder app,
IHostingEnvironment env, ILoggerFactory loggerFactory)
        {
            _loggerFactory = loggerFactory;

            app.UseDefaultFiles()
                .UseStaticFiles()
                .UseBotFramework();
        }
    }
}
```

6. Go back to SampleBot.cs added the following code with comment and Waterfallsteps.
   //Added: Part where code is added
   //Updated: Changing parts

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using Microsoft.Extensions.Logging;
using System;
using System.Threading;
using System.Threading.Tasks;


namespace HandsonBot.SampleBot
{
    public class SampleBot : IBot
    {
        private const string WelcomeText = "Welcome to Sample Bot";

        private readonly ILogger _logger;
        private readonly SampleBotAccessors _accessors; // Added
        private readonly DialogSet _dialogs; // Added

        public SampleBot(SampleBotAccessors accessors, ILoggerFactory
loggerFactory)
        {
            _accessors = accessors ?? throw new
ArgumentException(nameof(accessors));
            _dialogs = new DialogSet(accessors.ConversationDialogState);

            var waterfallSteps = new WaterfallStep[]
            {
        ConfirmAgeStepAsync,
        ExecuteAgeStepAsync,
        ExecuteFinalConfirmStepAsync,
        ExecuteSummaryStepAsync,
            };

            _dialogs.Add(new TextPrompt("name", ValidateHandleNameAsync));
            _dialogs.Add(new ConfirmPrompt("confirm"));
            _dialogs.Add(new NumberPrompt<int>("age"));
            _dialogs.Add(new WaterfallDialog("details", waterfallSteps));

            _logger = loggerFactory.CreateLogger<SampleBot>();
            _logger.LogInformation("Start SampleBot");
        }

        private async Task GetHandleNameAsync(DialogContext dialogContext,
DialogTurnResult dialogTurnResult, UserProfile userProfile,
CancellationToken cancellationToken)
        {
            if (dialogTurnResult.Status is DialogTurnStatus.Empty)
            {
                await dialogContext.PromptAsync(
                    "name",
                    new PromptOptions
                    {
                        Prompt = MessageFactory.Text("Please tell me your
handle name first."),
                        RetryPrompt = MessageFactory.Text("The handle name
must be at least 3 words long."),
                    },
                    cancellationToken);
            }

            // If you enter a handle name
            else if (dialogTurnResult.Status is DialogTurnStatus.Complete)
            {
                if (dialogTurnResult.Result != null)
                {
                    // Register your handle name with UserState
                    userProfile.HandleName =
(string)dialogTurnResult.Result;
                    await dialogContext.BeginDialogAsync("details", null,
cancellationToken); // added
                }
            }
        }

        private Task<bool>
ValidateHandleNameAsync(PromptValidatorContext<string> promptContext,
CancellationToken cancellationToken)
        {
            var result = promptContext.Recognized.Value;

            if (result != null && result.Length >= 3)
            {
                var upperValue = result.ToUpperInvariant();
                promptContext.Recognized.Value = upperValue;
                return Task.FromResult(true);
            }

            return Task.FromResult(false);
        }

        public async Task OnTurnAsync(ITurnContext turnContext,
CancellationToken cancellationToken = default(CancellationToken))
        {
            if (turnContext.Activity.Type == ActivityTypes.Message)
            {
                // We will exchange normal messages here.
                await SendMessageActivityAsync(turnContext,
cancellationToken); // updated
            }
            else if (turnContext.Activity.Type ==
ActivityTypes.ConversationUpdate)
            {
                await SendWelcomeMessageAsync(turnContext,
cancellationToken);
            }
            else
            {

_logger.LogInformation($"passed:{turnContext.Activity.Type}");
            }

            await
_accessors.ConversationState.SaveChangesAsync(turnContext, false,
cancellationToken);
            await _accessors.UserState.SaveChangesAsync(turnContext, false,
cancellationToken);
        }
        private static async Task SendWelcomeMessageAsync(ITurnContext
turnContext, CancellationToken cancellationToken)
        {
            foreach (var member in turnContext.Activity.MembersAdded)
            {
                if (member.Id != turnContext.Activity.Recipient.Id)
                {
                    await turnContext.SendActivityAsync(WelcomeText,
cancellationToken: cancellationToken);
                }
            }
        }

        public async Task SendMessageActivityAsync(ITurnContext
turnContext, CancellationToken cancellationToken)
        {
            var dialogContext = await
_dialogs.CreateContextAsync(turnContext, cancellationToken);
            var dialogTurnResult = await
dialogContext.ContinueDialogAsync(cancellationToken);

            var userProfile = await
_accessors.UserProfile.GetAsync(turnContext, () => new UserProfile(),
cancellationToken);

            // If the handle name is not registered in UserState
            if (userProfile.HandleName == null)
            {
                await GetHandleNameAsync(dialogContext, dialogTurnResult,
userProfile, cancellationToken);
            }

            // If you have a handle name registered with UserState
            else
            {
                // added
                if (dialogTurnResult.Status == DialogTurnStatus.Empty)
                {
                    await dialogContext.BeginDialogAsync("details", null,
cancellationToken);
                }
            }
        }

        //---------------------------------------------
        //Lab

        private async Task<DialogTurnResult>
ConfirmAgeStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
        {
            var userProfile = await
_accessors.UserProfile.GetAsync(stepContext.Context, () => new
UserProfile(), cancellationToken);

            return await stepContext.PromptAsync(
                "confirm",
                new PromptOptions
                {
                    Prompt = MessageFactory.Text($"{userProfile.HandleName}
May I ask your age?"),
                    RetryPrompt = MessageFactory.Text("Answer yes or No."),
                },
                cancellationToken);
        }

        private async Task<DialogTurnResult>
ExecuteAgeStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
        {
            if ((bool)stepContext.Result)
            {
                return await stepContext.PromptAsync(
                    "age",
                    new PromptOptions
                    {
                        Prompt = MessageFactory.Text("What is your age?"),
                        RetryPrompt = MessageFactory.Text("Enter the age in
numbers."),
                    },
                    cancellationToken);
            }
            else
            {
                return await stepContext.NextAsync(-1, cancellationToken);
            }
        }

        private async Task<DialogTurnResult>
ExecuteFinalConfirmStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
        {
            var userProfile = await
_accessors.UserProfile.GetAsync(stepContext.Context, () => new
UserProfile(), cancellationToken);
            userProfile.Age = (int)stepContext.Result;

            var message = GetAgeAcceptedMessage(userProfile);
            await stepContext.Context.SendActivityAsync(message,
cancellationToken);

            return await stepContext.PromptAsync(
                "confirm",
                new PromptOptions { Prompt = MessageFactory.Text("Is this
the registration information you want?") },
                cancellationToken);
        }

        private static IActivity GetAgeAcceptedMessage(UserProfile
userProfile)
        {
            return MessageFactory.Text(userProfile.Age == -1 ? "Age is
private, isn't it?" : $"I'm {userProfile.Age} year old.");
        }

        private async Task<DialogTurnResult>
ExecuteSummaryStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
        {
            if ((bool)stepContext.Result)
            {
                var userProfile = await
_accessors.UserProfile.GetAsync(stepContext.Context, () => new
UserProfile(), cancellationToken);
                var summaryMessages = GetSummaryMessages(userProfile);
                await
stepContext.Context.SendActivitiesAsync(summaryMessages,
cancellationToken);

                // End of Detail dialog
                return await stepContext.EndDialogAsync(cancellationToken:
cancellationToken);
            }
            else
            {
                // Redo the Details dialog.
                await
stepContext.Context.SendActivityAsync(MessageFactory.Text("I will visit you
again."), cancellationToken);
                return await stepContext.ReplaceDialogAsync("details",
cancellationToken: cancellationToken);
            }
        }

        private static IActivity[] GetSummaryMessages(UserProfile
userProfile)
        {
            IActivity summaryMessage = MessageFactory.Text(userProfile.Age
== -1
                ? $"{userProfile.HandleName} Your age is private."
                : $"{userProfile.HandleName} , {userProfile.Age} year
old.");
            IActivity thanksMessage = MessageFactory.Text("Thank you for
your input.");
            return new[] { summaryMessage, thanksMessage };
        }


        //----------------------------
    }
}
```

Chapter 5: Implementation Age Property

1. Go to **UserProfile.cs** , implement **Age** Property as shown below.

```
namespace HandsonBot.SampleBot
{
    public  class  UserProfile
    {
        public string HandleName { get; set; }
        public int Age { get; set; }
    }
}
```

2. Debug your application again.

Happy Coding !!!