內容

[前言 1](#_Toc486937355)

[事前準備 2](#_Toc486937356)

[所需使用者權限與資訊 3](#_Toc486937357)

[設計Bot流程 4](#_Toc486937358)

[實作 10](#_Toc486937359)

[建立並註冊Bot 10](#_Toc486937360)

[Cortana Channel設定 13](#_Toc486937361)

[設計語意處理 15](#_Toc486937362)

[實作Bot Application 16](#_Toc486937363)

[發佈 24](#_Toc486937364)

[Cortana Channel 24](#_Toc486937365)

[Skype Channel 28](#_Toc486937366)

[使用Cortana Skill 30](#_Toc486937367)

## **前言**

這份文件包含Source Code與建立與展示一個會議室機器人的必要的步驟：

* 透過Bot Framework建立Chatbot
* 設計Chatbot時的考量
  + 時區
  + 對話紀錄
  + 中斷對話
  + DialogBox控制
* 支援Azure AD身分認證
* Cortana Channel

由於Cortana Skill目前僅支援美國地區，且並非所有語言Cortana都能支援，以下的文件內容將會以英文語系為主。但是您可以透過修改LUIS的訓練資料讓機器人支援中文或其他非英文語系。

這份文件範例中並不包含會議室API的實作細節；文件的主要目的在於探討如何透過Bot Framework、LUIS以及其他Azure服務建置您的機器人，以及建置時需要考慮的事項。

## **事前準備**

* Microsoft Account(@Hotmail.com、@Msn.COM…等)
* 在Windows 10上可以使用Cortana的地區與語言(英文、日文、簡體中文等)
* 有Azure訂閱
  + LUIS
  + Web App
* Visual Studio 2015或2017(Community以上版本)

## **所需使用者權限與資訊**

* 使用者的位置
* 使用者的姓名
* 使用者的Email
* Azure AD上使用者的基本資料

## **設計Bot流程**

#### 參考資源

* AutoBot: <https://github.com/MicrosoftDX/AuthBot>
* Meeting Room Bot Sample Code: <https://github.com/michael-chi/meetingroombotdemo>

#### 需求

* 我們希望可以讓使用者透過下面的語句指定會議室：

Book A901 tomorrow at 10 am for 1 hour

I need A902 next Monday at 10 am for 2 hours

* 或是提供一些資料，讓系統來尋找會議室：

I need a room next Friday at 10 am

Book a room next Friday at 10 am for 1 hour for 2 people

* 如果使用者指定了會議室，則系統要確認指定時間該會議室是否可用
  + 如果可用，直接把會議室定下來
  + 如果不可用，告訴使用者該會議室不可使用
* 如果使用者沒有指定會議室
  + 並且提供了完整的資訊
    - 系統要尋找符合條件的會議室，並列出供使用者選擇
  + 如果沒有提供完整資訊
    - 系統透過對話取得所有必要資訊，然後搜尋所有可用會議室供使用者選擇

#### 流程圖



#### Bot設計考量

使用者可能在第一次對話就提供了所有資訊，也有可能分段提供；考慮上述的流程，我們將機器人分成幾個DialogBox來取得相關資訊以及訂定會議室。

基本DialogBox流程如下



* MainDialog負責分析目前使用者輸入的語句中已經包含了哪些資訊，還需要哪些資訊
* DetailDialog負責收集使用者尚未輸入的必要資訊
* ConfirmDialog負責呼叫Meeting Room API，如果會議室確認成功，回傳結果以及iCS檔案給使用者；如果沒有訂到會議室，或是有其他需要使用者確認的事項，也會在這個Dialog處理。

##### 系統設計

* 收集使用者需要的會議室資訊
  + 為了訂會議室，我們會需要幾個資訊
    - (Optional)MeetingRoomID
    - Meeting Date
    - Meeting Start Time
    - Attendee size (如果使用者指定了MeetingRoomID則此項為Optional)
    - Meeting Duration
  + 我們預期使用者可能會用下面的方式直接預定指定的會議室
    - 包含完整資訊
      * Book A901 next Monday at 10 am for 2 hours
      * Book a room tomorrow at 10 am for 1 hour for 2 people
    - 不包含完整資訊
      * Book a room next Monday at 10 Am
      * Book A902 for 1 hour
      * I need a room at next Friday 5 PM
  + 假如使用者的輸入不包含完整資訊，在DetailDialog收集資訊時，我們會希望使用者能夠用明確的字句表達資訊；例如

Bot: How many people would attend this meeting in person

User: 10

而不是用下面的語句

Bot: How many people would attend this meeting in person

User: There’re 10 people would attend this meeting

第二種對話方式雖然比較貼近自然的對話方式，但是Open Question的處理方式會造成程式處理方式變得複雜，並且有極大的機率變得無法良好的控制使用者的輸入；例如使用者也可能說David, Mary, Tom and I will be attending this meeting. 我們會需要花費額外的時間來處理這類的回應。

一般來說，在Bot發布的第一個階段，我們會著重在功能的實現；上線之後透過收集使用者真正的對話，在第二、第三階段讓對話更貼近自然。

* FormFlow or LuisDialog

Bot Framework提供了許多種收集資訊的方式；大多數的情況我們會選擇使用LUIS(Language Understanding Intelligence Service)或是FormFlow，或是兩者混用。

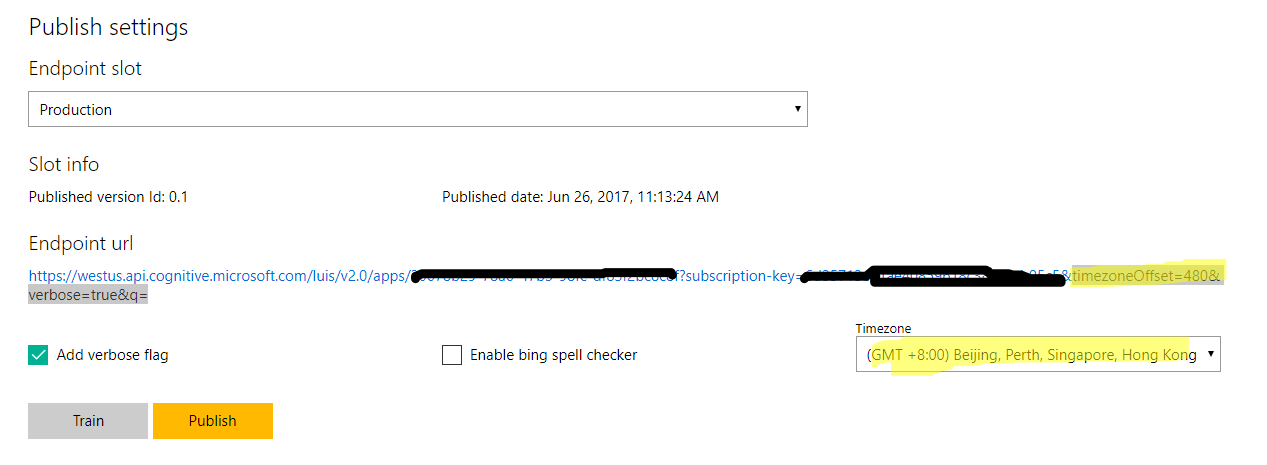
考慮上面的對話流程，MainDialog作為收集使用者最初的輸入，我們會在這裡使用LuisDialog來分析輸入資訊。

DetailDialog會根據使用者尚未輸入的資訊，產生需要的欄位要求使用者輸入。這裡可以考慮使用FormFlow或是LuisDialog；在本例，我們使用LuisDialog。

ConfirmDialog理論上只負責資料的呈現(會議室ID、時間等等)

* Bot Dialog Control
  + 在Bot Framework中，我們可以透過IDialogContext::Forward()將目前使用者的對話”Forward”到下一個Dialog；或是透過IDialogContext::Call()來”呼叫”某一個Dialog
    - Forward會將目前使用者的對話轉送給指定的Dialog；該Dialog初始化之後，會在MessageReceived()中取得使用者輸入的對話
    - Call則僅僅只將目前的對話控制權交給指定的Dialog，該Dialog被初始化之後會等待使用者輸入。
  + 由於一個DialogBox可以喚起下一個Dialog，下一個Dialog又可以繼續喚起其他的Dialog；這一連串的呼叫，會形成Dialog Stack；我們可以透過DialogStack來控制DialogBox的狀態；例如當使用者輸入Cancel時清空整個Stack，重新等待使用者輸入。
* 身分驗證
  + 在這個範例中，我們透過Azure AD整合的方式來驗證使用者身分。如果公司有使用Office365，並且有作Hybrid整合，則同樣可以用此範例提供的方式驗證身分。
  + 在這個範例中，我們使用微軟TED團隊所開發的AuthBot來幫助我們作Azure AD身分驗證：<https://github.com/MicrosoftDX/AuthBot>
* 時區
  + 使用者所使用的IM App的時區，應該都是當地時區，例如UTC+8。當在台灣的使用者與在美國的使用者說：I want to book a room tomorrow at 10 am時，系統必須能夠將他轉換為正確的時間。
  + LUIS
    - LUIS與Azure服務的預設時區都是GMT時區；LUIS可以透過發布時的設定修改預設時區。假設我們的使用者都在同一個時區，那麼可以將LUIS設定固定時區即可。否則，可以透過修改呼叫的REST API URL來指定時區。例如：

[https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/{model-id}?subscription-key={subscription-key}&timezoneOffset=480&verbose=true&q=book%20a%20room%20tomorrow%2010%20am](https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/%7bmodel-id%7d?subscription-key=%7bsubscription-key%7d&timezoneOffset=480&verbose=true&q=book%20a%20room%20tomorrow%2010%20am)



* + - Bot Framework的LuisDialog可以透過override ModifyLuisRequest來修改；要注意的是這裡的時區是以分鐘的形式傳遞(如timezoneOffset=-120)

protected override LuisRequest ModifyLuisRequest(LuisRequest request)

{

request.ExtraParameters =

"timezoneOffset=" +

this.TimezoneOffset?.TotalMinutes.ToString();

return request;

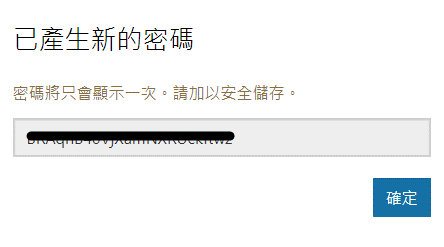
}

* + Bot Builder SDK
    - 3.8.2版本以後的Bot Builder SDK提供了一個額外的Activity Property: localtimestamp；當App支援時，這個Property會呈現使用者設備上所指定的當地時間。
    - 要注意的是，並不是每一種IM App都支援這個Property；我們仍然可能會需要透過機器人問答的方式取得使用者所在的時區。或者是修改我們的機器人，讓使用者提供正確的時間，而非以tomorrow這樣的說法指定時間。
    - 如果使用者使用的Channel可以提供Location(例如Cortana)；另一個方式是透過經緯度換算時區。
  + 在這個範例中，考慮到跨時區使用者，我們不修改LUIS的時區設定，而依賴前端App來提供時區。當前端App沒有提供時間時，我們會讀取設定檔指定一個時區。
* 使用者對話紀錄
  + 我們可以透過實作IActivityLog來記錄所有使用者的對話。一般來說，我們會根據ConversationId來那些語句屬於同一次對話；但在某些IM App中，ConversationId可能不會改變。此時也許我們會需要透過其他的方式將對話依照對話時間分群。
* 使用者取消
  + 想像使用者在訂會議室的過程當中，也許突然改變主意了。我們需要一個機制讓使用者可以隨時”中斷”對話。在Bot Framework中，我們透過實作IScorable以及Global Message Handler來達到這個目的。
* 系統監控
  + Bot實作完成後，會透過REST API的形式與Bot Connector連接；其實也就是一個API App。我們可以透過Application Insights來取得API App運作的紀錄；也可以透過Application Insights API發送自訂的Telemetry到Azure上，並透過Power BI來製作Dashboard。

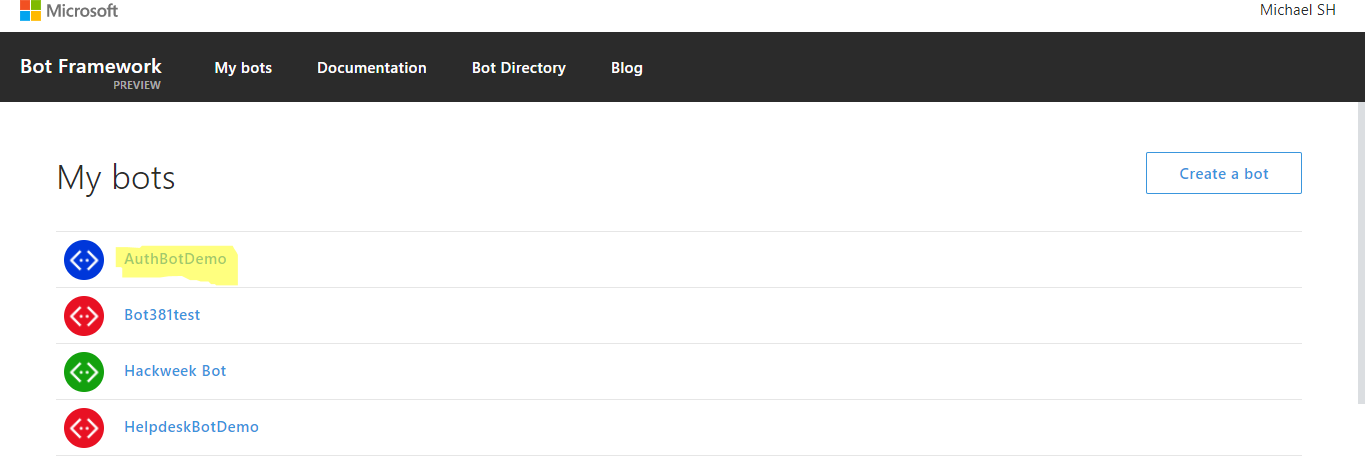
## 實作

### 建立並註冊Bot

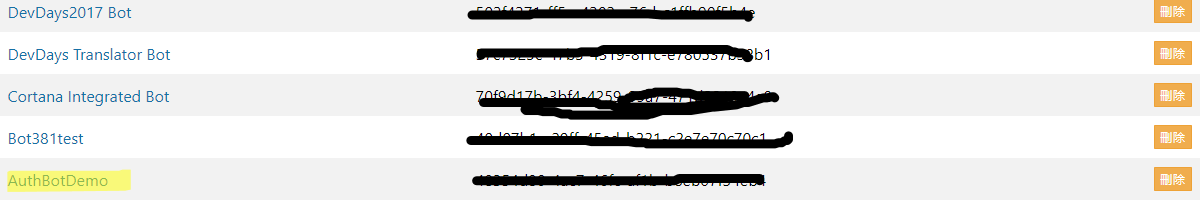
* 以**MSA**登入<https://dev.botframework.com>
* 註冊並建立一個Bot
  + 測試開發階段，Bot Handler的位址可以使用<https://localhost:3990/api/messages>
  + 正式發布後，這個位址要改為實際發布地位址，例如<https://demo.com/api/messages>
* 在過程中會在App Portal建立一個App，請把App的金鑰記錄下來。



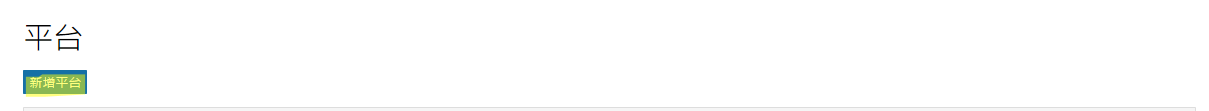
* 在<https://dev.botframework.com> 建立完成後，應該會看到如下的畫面



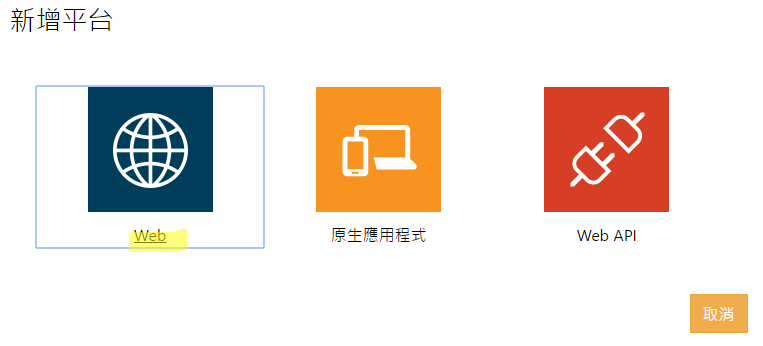
* 登入<https://apps.dev.microsoft.com/#/appList>
* 找到先前登錄的Bot，例如AuthBotDemo



* 點擊AuthBotDemo，進入設定頁面，並新增一個平台



* 選擇Web



* 設定Redirect URL為<https://www.bing.com/agents/auth>



* 如果我們的會議室系統已經與Office 365整合，則我們可以透過Microsoce Graph API來幫我們尋找以及訂會議室；如果需要這個功能，需要設定Microsoft Graph權限如下：



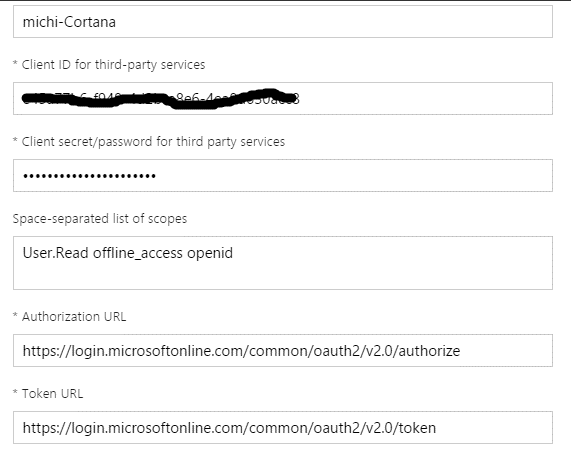
* 儲存

### Cortana Channel設定

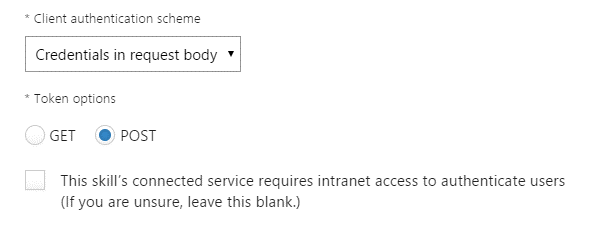
* 以Microsoft Account(非AD Account)登入<https://dev.botframework.com>
* 啟用Cortana Channel
* 其中，Client ID與Client Secret為剛剛註冊Bot時所產生的ID/Password。

Authorization Url與Token Url如下。

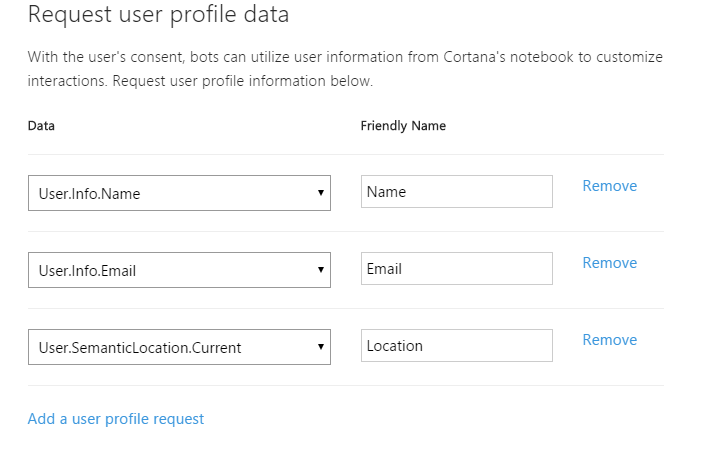
* + 如果剛剛沒有另外設定Microsoft Graph權限，則Scope為：User.Read offline\_access openid。



* 將Credential設定如下



* 同時要求以下權限



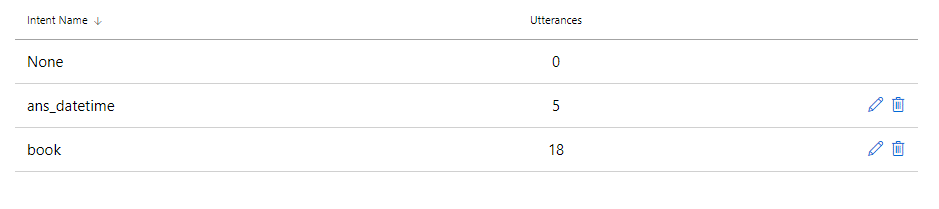
* 儲存設定

### 設計語意處理

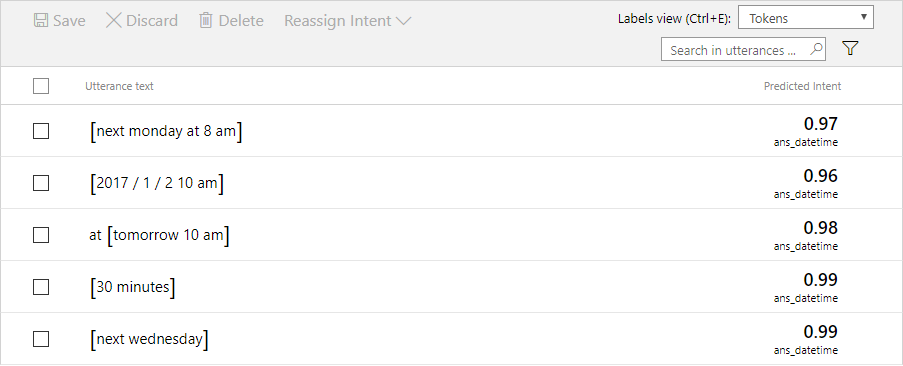
* 我們會使用LUIS來處理語意辨識。除了基本的會議室資訊收集之外，我會在這裡透過LUIS內建的Entity來處理類似Next Monday、Tomorrow之類的時間指定；因此我們會建立兩個Intent：ans\_datetime(處理時間分析)以及book(會議室資訊)
* 在LUIS建立以下的Entity
  + datetimeV2用以分析時間日期，例如Tomorrow at 10 AM、Next Friday at 2 PM或是July 10
  + roomid室會議室代號，如果有清單，可以把roomid設為”List”類型並輸入所有會議室代號；在這裡我使用Simple，讓LUIS來幫我們判斷
  + size是參加會議的人數，Meeting Room API需要這個參數來找到適合的會議室



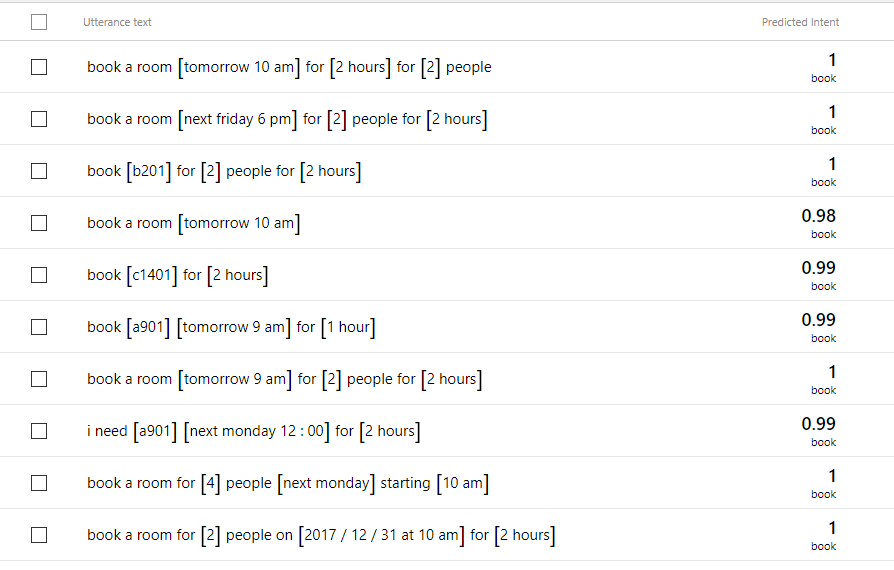
* 建立以下的Intent



* + ans\_date



* + Book

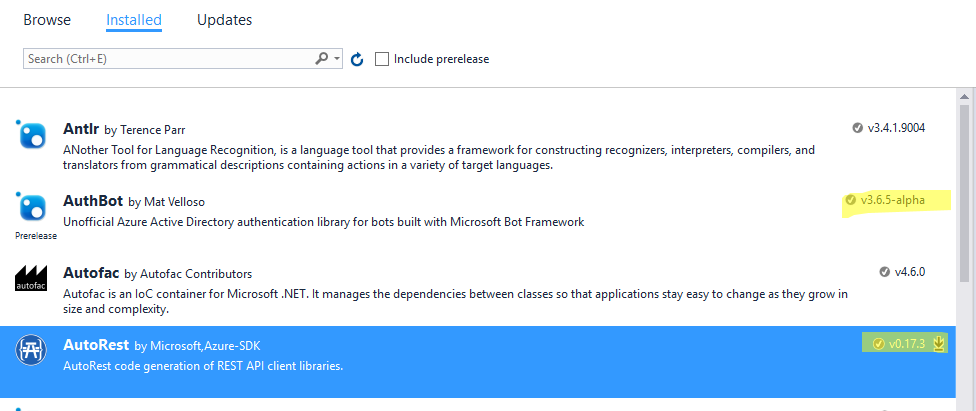


### 實作Bot Application

完整的Source Code在：<https://github.com/michael-chi/meetingroombotdemo>

這裡僅針對較重要的部分說明。

* 在Meeting Room Bot專案加入Autorest的Nuget套件
* 在Meeting Room Bot專案加入Authbot套件及其相依賴的套件



* Cortana Channel支援透過Connected Service來做使用者身分驗證；當使用Connected Service時，每個從Cortana發送上來的訊息本身都會帶一個AuthenticationToken的Property其內容為JWT Token。

透過解析這個JWT Token便可以取的使用者身分等相關資料。

* 將AuthBot source code加入Solution中；並且移除Meeting Room Bot的Authbot.dll reference然後加入Authbot Project的Reference。
* 我們需要在使用者發送訊息時檢查使用者是否已經登入；方法是在每次使用者發送消息時檢查在BotState中登入的狀態是否為true。AuthBot提供了透過Sign-in Card的方式要求使用者登入Azure AD；但是他目前並未支援Cortana本身提供的Connected Service方式；因此我們需要修改增加這一部分的處理。
* 打開Authbot的ContextExtension.cs，加入GetCortanaAccessToken()

private static string GetCortanaAccessToken(IBotContext context)

{

string token = string.Empty;

Activity activity = context.Activity as Activity;

if (activity.ChannelId.Equals("cortana",

StringComparison.InvariantCultureIgnoreCase))

{

var msg = activity as IMessageActivity;

{

var tokenEntity = msg.AsMessageActivity().Entities.Where(e => e.Type.Equals("AuthorizationToken")).SingleOrDefault();

if (tokenEntity == null)

{

//Debug mode

token = string.Empty;

}

else

{

token = tokenEntity?.Properties.Value<string>("token");

}

}

}

else

{

token = string.Empty;

}

return token;

}

* 修改原本的兩個GetAccessToken()，加入Cortana相關的判斷。

if (context.Activity.ChannelId.Equals("cortana",

StringComparison.InvariantCultureIgnoreCase))

{

string token = null;

if (context.UserData.TryGetValue(ContextConstants.AuthResultKey,

out authResult))

{

//we have credential

}

else

{

token = GetCortanaAccessToken(context);

var jwt = new JwtSecurityToken(token);

authResult.AccessToken = token;

long tick = long.MinValue;

long.TryParse(jwt.Payload.Claims.Where(

c => c.Type.Equals("exp",

StringComparison.InvariantCultureIgnoreCase))

.SingleOrDefault()?.Value, out tick);

authResult.ExpiresOnUtcTicks = tick;

InMemoryTokenCacheMSAL tokenCache =

new InMemoryTokenCacheMSAL(Encoding.ASCII.GetBytes(token));

authResult.TokenCache = tokenCache.Serialize();

context.StoreAuthResult(authResult);

}

return authResult.AccessToken;

}

* 當使用非Cortana Channel時，我們會使用AuthBot的Sign-in Card要求使用者登入。使用者登入成功後，AAD會Callback我們在註冊Bot時指定的OAuthCallback handler。這個Handler已經包含在AuthBot中。
* 在Visual Studio的Meeting Room Bot專案中，打開settings.config如下：

<appSettings>

<!-- update these with your BotId, Microsoft App Id and your Microsoft App Password-->

<add key="BotId" value="YourBotId" />

<add key="MicrosoftAppId" value="{your app id}" />

<add key="MicrosoftAppPassword" value="your password" />

<add key="StorageName" value="storage account"/>

<add key="StorageKey" value="storage key"/>

<add key="MeetingRoomApiUrl" value="meeting api url"/>

<add key="LuisModel" value="luis model id"/>

<add key="LuisKey" value="luis key"/>

<add key="ActiveDirectory.Mode" value="v2" />

<add key="ActiveDirectory.Scopes" value="User.Read Calendars.Read Calendars.Read.Shared Calendars.ReadWrite Calendars.ReadWrite.Shared" />

<add key="ActiveDirectory.EndpointUrl" value="https://login.microsoftonline.com" />

<add key="ActiveDirectory.Tenant" value="common" />

<add key="ActiveDirectory.ClientId" value="{your app id}" />

<add key="ActiveDirectory.ClientSecret" value=" your password " />

<add key="ActiveDirectory.RedirectUrl" value="https://localhost:3990/api/OAuthCallback" />

</appSettings>

* + 修改MicrosoftAppId、ActiveDirectory.ClientId為剛剛所註冊的Bot Id
  + 修改MicrosoftAppPassword、ActiveDirectory.ClientSecret為先前步驟註冊的Bot的Password
  + ActiveDirectory.RedurectUrl在測試階段可以設定為<https://localhost:3990/api/OAuthCallback>；佈署後則需要修改為https://{your FQDN}/api/OAuthCallback
  + 修改LuisModel與LuisKey為剛剛建立的Luis的App Id與App Key
  + 修改Storage Account與Storage Key為Azure Storage Account的相關資料；我們需要這個Storage Account提供ICS檔案讓使用者下載。
  + 其他的設定都可以不需要修改。
* 打開Global.asax.cs，透過以下程式碼讀取設定檔。

AuthBot.Models.AuthSettings.Mode = ConfigurationManager.AppSettings["ActiveDirectory.Mode"];

AuthBot.Models.AuthSettings.EndpointUrl = ConfigurationManager.AppSettings["ActiveDirectory.EndpointUrl"];

AuthBot.Models.AuthSettings.Tenant = ConfigurationManager.AppSettings["ActiveDirectory.Tenant"];

AuthBot.Models.AuthSettings.RedirectUrl = ConfigurationManager.AppSettings["ActiveDirectory.RedirectUrl"];

AuthBot.Models.AuthSettings.ClientId = ConfigurationManager.AppSettings["ActiveDirectory.ClientId"];

AuthBot.Models.AuthSettings.ClientSecret = ConfigurationManager.AppSettings["ActiveDirectory.ClientSecret"];

AuthBot.Models.AuthSettings.Scopes = ConfigurationManager.AppSettings["ActiveDirectory.Scopes"].Split(' ');

* 如果使用Cortana Channel，我們還可以要求Cortana將使用者所在的位置座標發送到後端來；其座標位置會被放在訊息的UserInfo這個Entity中。

{

"type": "UserInfo",

"Name": {

"GivenName": "GivenName",

"FamilyName": "FamileName"

},

"Email": "xxxx@msn.com",

"Location": {

"StartTime": "2017-06-18T05:37:05Z",

"EndTime": "2017-06-18T13:51:57Z",

"Hub": {

"Id": "xxxx-xxxx-4404-9652-77b4e95a2ac3",

"Type": "Home",

"Name": "home",

"Latitude": 225.0225,

"Longitude": 221.0221,

"Address": "Some Addresses"

},

"VenueName": "",

"Away": false

}

},

* 同樣在Global.aspx.cs，我們透過下列代碼註冊ActiveLog與Global Message Handler

var builder = new ContainerBuilder(); builder.RegisterType<CancelScorable>().AsImplementedInterfaces().InstancePerLifetimeScope(); builder.RegisterType<ActivityLogger>().AsImplementedInterfaces().InstancePerDependency();

builder.Update(Conversation.Container);

* 實作ActivityLogger.cs作為對話紀錄。在這裡我們透過Application Insights把對話紀錄上傳到Application Insights中。

public async Task LogAsync(IActivity activity)

{

if (activity.ChannelData != default(dynamic))

{

Trace.TraceInformation("[Channel Data]" + JsonConvert.SerializeObject(activity).Replace(Environment.NewLine, ""));

}

try

{

var tokenEntity = activity.AsMessageActivity().Entities.Where(e => e.Type.Equals("AuthorizationToken")).SingleOrDefault();

var token = tokenEntity?.Properties?.Value<string>("token");

if (token != null)

{

var jwt = new JwtSecurityToken(token);

var log = string.Join(" | ", jwt.Payload.Claims.ToArray().Select(c => c.Type + ":" + c.Value).ToArray());

Trace.TraceInformation("[Token]" + log);

}

}

catch(Exception exp)

{

Trace.TraceError($"[Exception]{exp.ToString()}");

}

* 實作CancelScorable.cs作為中斷對話的Global Handler。

protected override async Task<string> PrepareAsync(IActivity item, CancellationToken token)

{

var message = item as IMessageActivity;

if (message != null && !string.IsNullOrWhiteSpace(message.Text))

{

if (message.Text.Trim().Equals("cancel", StringComparison.InvariantCultureIgnoreCase))

{

return "cancel";

}

}

return null;

}

* 在MainDialog.cs中，我們建立幾個必要的欄位

public readonly string[] RequiredParameters = new string[]

{

"size","startDate","startTime","duration"

};

* 在book的Intent Hander中，我們需要判斷現在使用者提供了哪些資料。如果使用者有提供roomId，則不需要size。

var duration = TryFindEntity(result, "builtin.datetimeV2.duration",

"duration")?.SingleOrDefault();

var startDateTime = TryFindEntity(result, "builtin.datetimeV2.datetime",

"datetime")?.ToArray().FirstOrDefault();

var startDate = TryFindEntity(result, "builtin.datetimeV2.date",

"date")?.ToArray().FirstOrDefault();

var startTime = TryFindEntity(result, "builtin.datetimeV2.time",

"time")?.ToArray().FirstOrDefault();

var roomId = TryFindEntity(result, "roomid")?.SingleOrDefault();

var size = TryFindEntity(result, "size")?.SingleOrDefault();

bool requireMoreInfo = false;

requireMoreInfo = (startDateTime == null && startTime == null) ||

(size == null && roomId == null) ||

(duration == null);

* 找出所有額外需要使用者提供的資料後，透過IDialogContext::Forward()將使用者輸入的語句轉送給DetailDialog

await context.Forward<object>(new DetailInfoDialog()

{

RequiredInformation = queue,

StartDate = startDate?.ToDateTime(TimezoneOffset),

StartTime = startTime == null ?

startDateTime?.ToDateTime(TimezoneOffset) :

startTime?.ToDateTime(TimezoneOffset),

Duration = duration?.ToTimeSpan(),

Size = size?.ToInt64(),

RoomId = roomId?.Entity,

Offset = TimezoneOffset

},

ResumedFromDetailForm,

(Microsoft.Bot.Connector.Activity)context.Activity);

* 當所有資料都收集完畢時，將資料轉送到ConfirmDialog

private async Task Book(IDialogContext context, DetailInfoDialog detail)

{

var confirmDialog = new BookingConfirmDialog(

new MeetingRoomSuggestionConstrain

{

Attendees = new List<string> { Requestor },

Location = detail.RoomId,

Start = detail.StartTime.Value.DateTime,

End =

detail.StartTime.Value.DateTime

.AddMinutes(detail.Duration.Value.TotalMinutes),

Size = (int)(detail.Size.HasValue ? detail.Size.Value : 0),

MeetingOrganizer = Requestor,

MeetingRoomIds = string.IsNullOrEmpty(detail.RoomId) ? null :   
new string[] { detail.RoomId },

LocationDisplayNames = null,

TimezoneOffset = TimezoneOffset

});

await context.Forward<object>(confirmDialog,

ResumedFromConfirmationDialog,

context.Activity.AsMessageActivity());

}

* 在這裡我們使用Forward而不使用Call的原因是，對使用者而言，他已經輸入了問句；我們希望DetailDialog及ConfirmDialog直接接手處理，而不需要等待使用者輸入。
* 在DetailDialog中，當所有的資料都收集完畢後，DetailDialog需要把資料傳回給MainDialog，由MainDialog接手處理。

if (StartDate.HasValue && StartTime.HasValue)

{

StartTime = new DateTimeOffset(StartDate.Value.Year,

StartDate.Value.Month,

StartDate.Value.Day,

StartTime.Value.Hour,

StartTime.Value.Minute,

StartTime.Value.Second,

StartTime.Value.Offset);

}

ctx.Done<object>(this);

* 當Bot需要日期時間資訊時，我們會透過Prompt Dialog來取得資訊。如先前提到的，LUIS提供了datetimeV2這個預設的entity幫我們解析時間日期相關資訊；因此在這裡，我們透過LuisService指定Luis Model呼叫。

TimeRelatedPromptDialog.cs

LuisService luis = new LuisService(new LuisModelAttribute(

modelID: ConfigurationManager.AppSettings["LuisModel"],

subscriptionKey: ConfigurationManager.AppSettings["LuisKey"],

apiVersion: LuisApiVersion.V2

));

* 透過override TryParse()來判斷輸入的日期時間

protected override bool TryParse(IMessageActivity message, out string result)

{

result = ((Activity)message).Text;

T item = Task.Run(() => GetResultDate(((Activity)message).Text)).Result;

bool hasResult = false;

if (EqualityComparer<T>.Default.Equals(item, default(T)))

{

result = null;

hasResult = false;

}

else

{

Result = item;

OnCompleted?.Invoke(Result);

hasResult = true;

}

return hasResult;

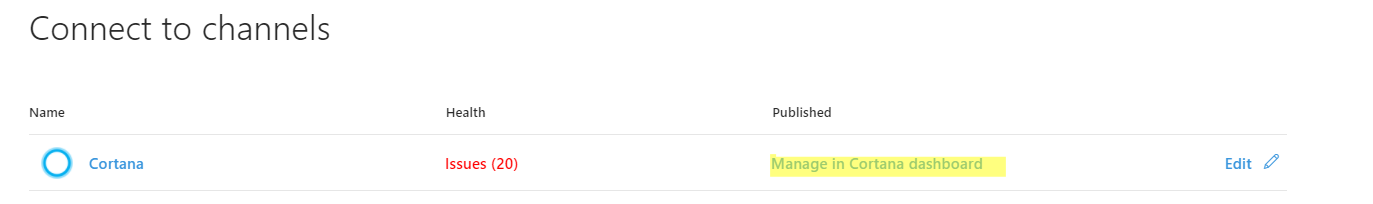
}

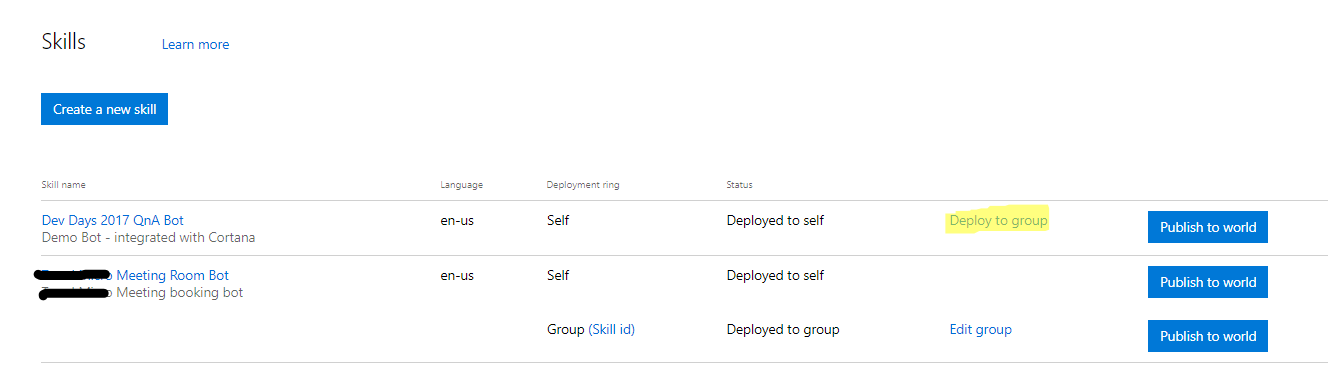
* 最後，當我們確定Book到會議室了，機器人會產生一個ICS檔案讓使用者下載。我們透過ICSGenerator.cs來產生這個檔案；並且，我們透過HeroCard產生一個會議的Summary呈現給使用者。

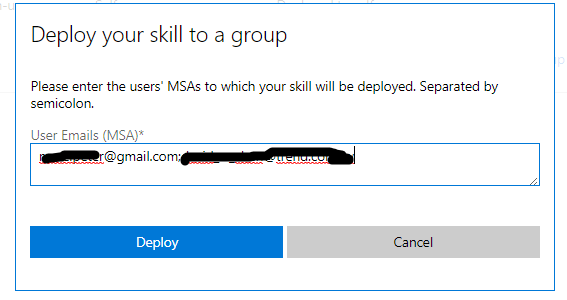
## **發佈**

### Cortana Channel

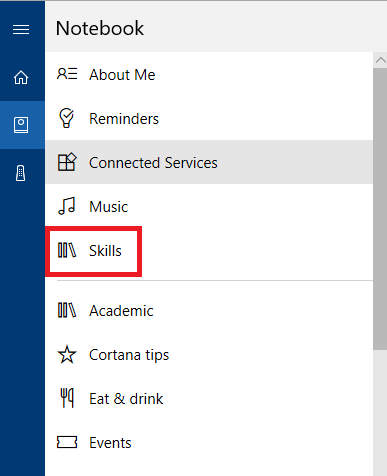
* 測試階段可以在Cortana Management Dashboard上將此Skill發佈給測試者

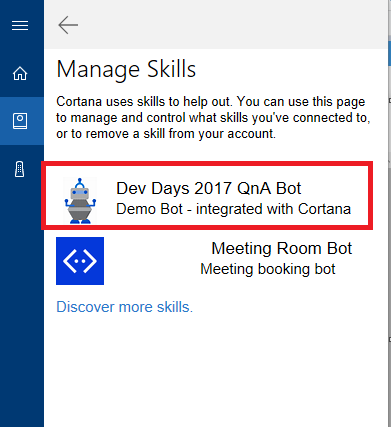






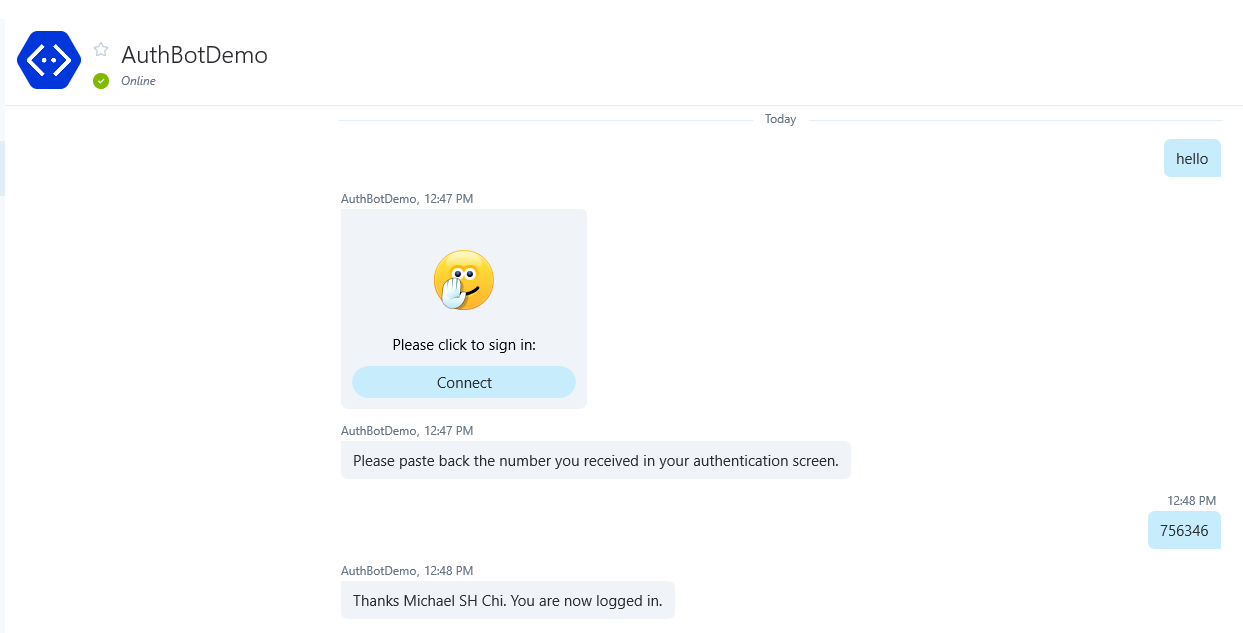
* 使用者會收到一封邀請信件，使用者點擊連結後，便可將她的MSA與Cortana Skill連結起來。
* 使用者可以在Cortana的Notebook -> Skills中確認已經連結到Cortana Skill

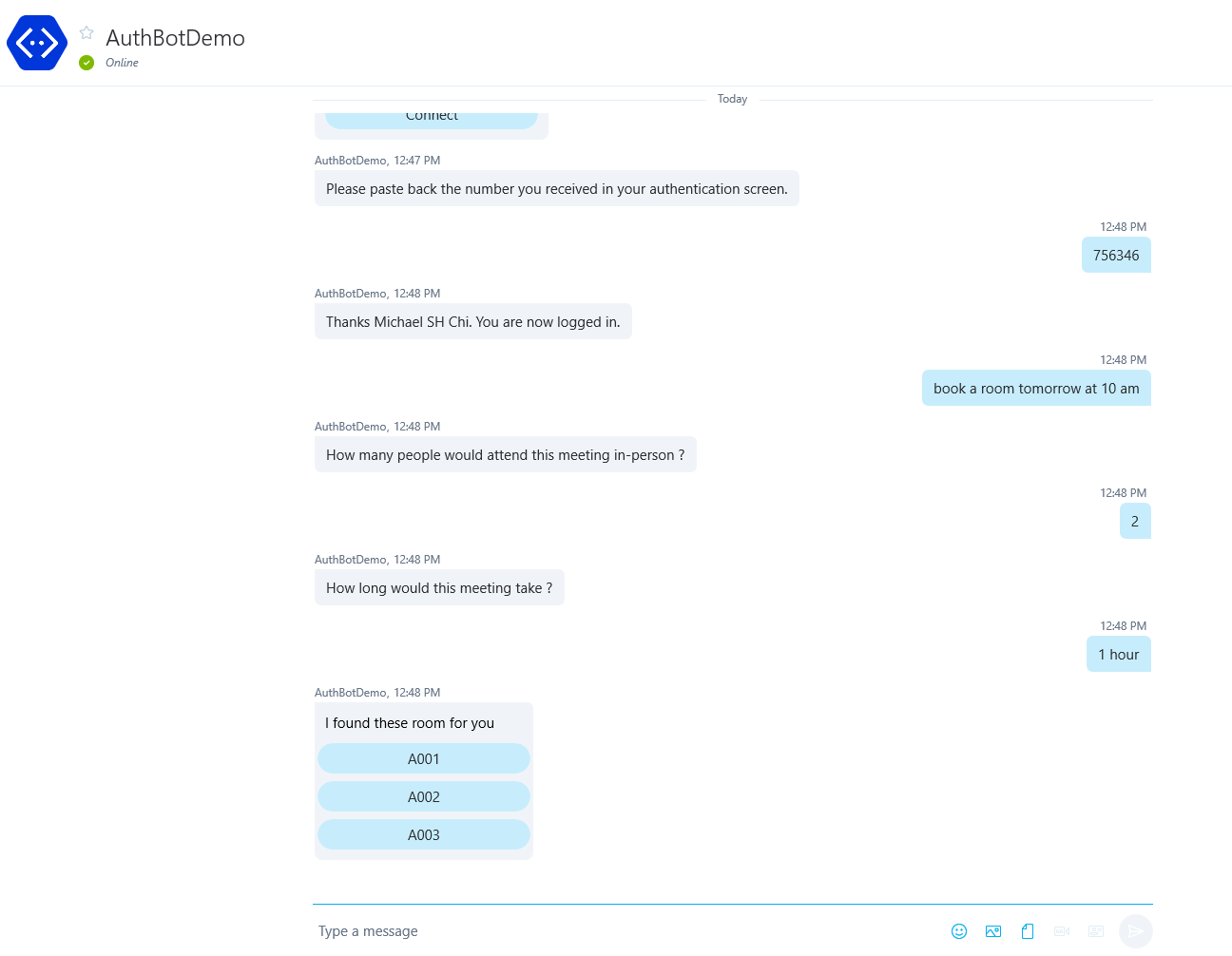


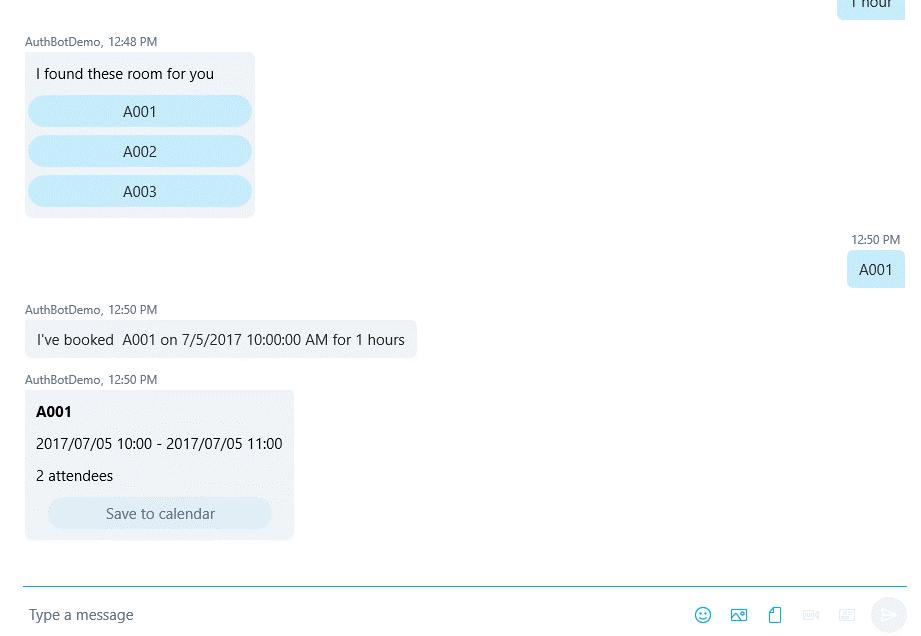


* 當管理者將Cortana Skill發佈到Production後，使用者可以在<https://www.microsoft.com/en-us/windows/cortana/cortana-skills/> 找到發佈的Skill

### **Skype Channel**







## **使用Cortana Skill**

* 打開Cortana後，可以使用語音輸入Tell Meeting Room Hi，或是Ask Meeting Room book a room tomorrow開始對話
* 第一次使用時，系統會帶出登入畫面；請輸入Azure AD或是Office 365上的帳號密碼登入