Partner Center Application Onboarding

Hands-on Labs

# Hands-on Lab 1: Introduction to the Partner Center API/SDK

The Partner Center SDK provides a superset of the earlier CREST API functionality. The SDK enables partners to connect sales and billing systems to Microsoft, so that the partners may sell Microsoft Commercial Cloud software. This Visual Studio-based lab uses the Partner Center SDK to demonstrate a simple partner-based application: creating a customer and selling that customer a new subscription;

## Exercise 1: On-board applications in Partner Center

In this exercise, you will on-board a Native Azure Active Directory application from Partner Center to access the Partner Center APIs.

## Exercise 2: Configure your First Application

In this exercise, you will configure a C# console application with the on-boarded Active Directory application credentials, and then pull the Partner Center SDK from Nuget.

## Exercise 3: Create and Query Paged Customers

In this exercise, you will extend the console application with the Partner Center SDK to create a customer. You will also add code to have the console query for customers. Finally, you will use Fiddler to inspect the raw HTTP requests made by the SDK.

## Exercise 4: Browse and Select Offers

In this exercise you will extend the console application to list the available Offers.

## Exercise 5: Place an Order for an Azure Subscription

In this exercise, you will extend the console application to place an order for a selected Azure subscription.

## Prerequisites

* Access to the Partner Center website (<https://partnercenter.microsoft.com>).
* The login information for your CSP Integration Sandbox account. Check the link to understand how to [Set up API access in Partner Center](https://docs.microsoft.com/en-us/partner-center/develop/set-up-api-access-in-partner-center)
* Download and install Visual Studio 2015/2017, [click here to download](https://www.visualstudio.com/en-us/downloads/).
* If you run automated build verification tests, conduct testing in production, or perform manual testing in the integration sandbox, you may reach the maximum limits for the integration sandbox. These limits are 75 customers, 5 subscriptions per customer, 25 seats per subscription, and $200 of Azure usage per month. [Click here for details](https://docs.microsoft.com/en-us/partner-center/develop/test-and-debug).
* To clean up at the end of your test run so there's space for the next round of testing, see the following topics:
  + [Delete a customer account from the integration sandbox](https://docs.microsoft.com/en-us/partner-center/develop/delete-a-customer-account-from-the-integration-sandbox)
  + [Decrease the quantity of a subscription](https://docs.microsoft.com/en-us/partner-center/develop/change-the-quantity-of-a-subscription)
  + [Suspend a subscription](https://docs.microsoft.com/en-us/partner-center/develop/suspend-a-subscription) so that you can remove it.

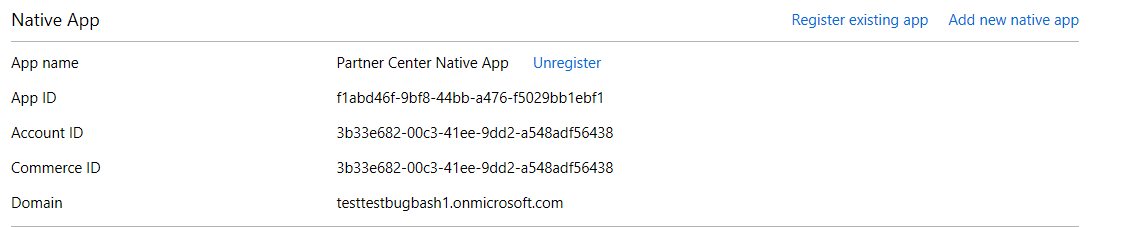
## Exercise 1: On-board applications in Partner Center

In this exercise, you will on-board an Azure Active directory application to Partner Center. This process enables your applications to access Partner Center APIs.

1. Log in to Partner Center (<https://partnercenter.microsoft.com>), using your integration sandbox account credentials.
2. Navigate to the dashboard, and then select **Application Settings** from the left side menu.
3. From the left-side menu, navigate to **App Management**.
4. Click on the **add new native app** link to create and onboard a native Azure Active Directory application.

In this tutorial, you will be using a user + app login model; therefore, you need perform this step in order to register and have your console application interact with Partner Center.

1. Observe the **App ID** property, which you will use in the upcoming exercises to identify your application.



**Important Note**: If this native app does not work well, you can recreate a new native app to instead

Congratulations! In this exercise you created a new Azure AD application, and then granted your new application access to Partner Center and customer’s subscription.

## Exercise 2: Configure your first Application

In this exercise, you will create and configure a C# application that will use the Partner Center SDK to invoke various Partner Center APIs.

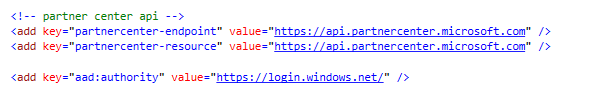
### Configure Partner Center credentials

1. From the provided resources folder, locate and open the Demo1 with Visual Studio.
2. Open the App.config file.

This file contains all the configuration settings needed to successfully authenticate and issue calls to the Partner Center API service.

1. Identify the preset settings for your application.

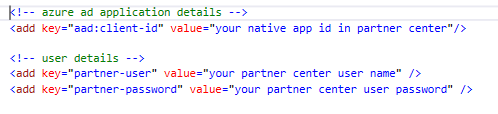
The App.config file includes a number of preset settings your application uses to issue calls to the Partner Center API service. such as those as shown below:



These settings will be used later in the tutorial to issue an Azure Active Directory token. Your console application will then use that token to access the Partner Center APIs.

1. Fill in the following key values in the App.config file:

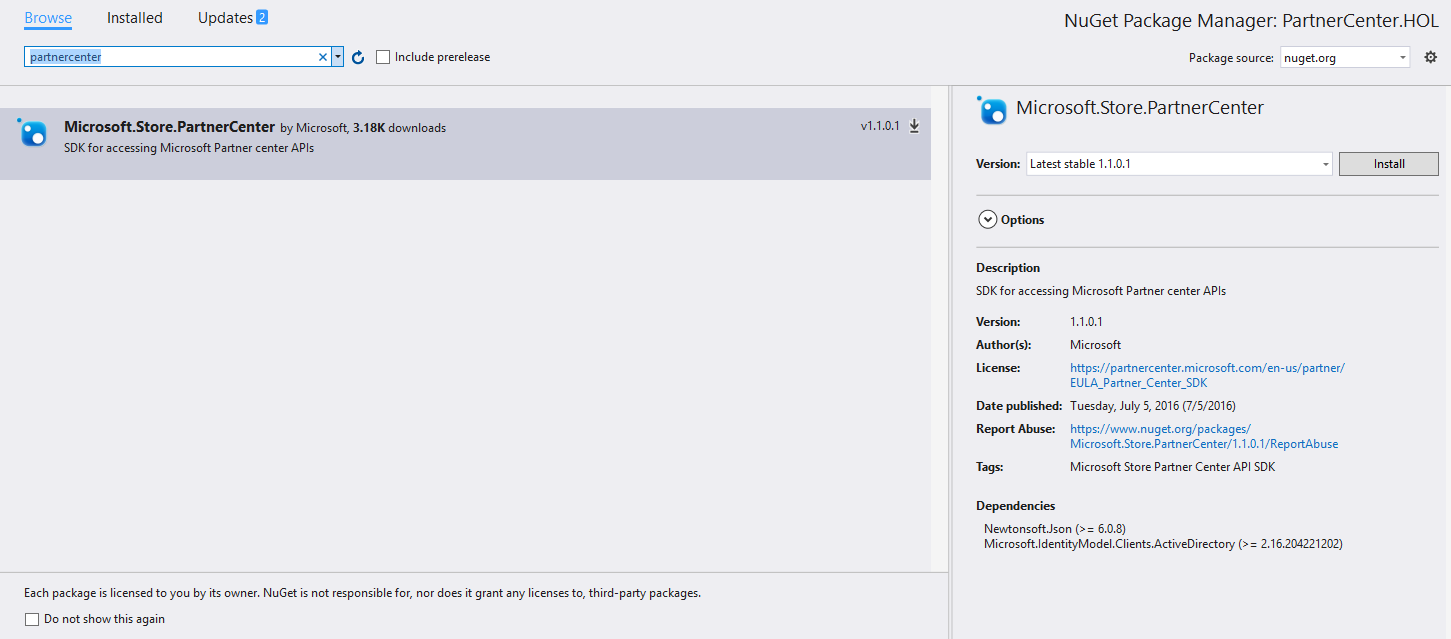
* **aad:client-id** – the application ID of the application from step 1.
* **partner-user** – the name of the admin agent user.
* **partner-password** – the password of the admin agent user.



You will need to provide the application ID you onboarded in step1 to identify the application to Partner Center. You will use the partner’s admin user name and password to access Partner Center. Later in the lab, you will use the Configuration class to simplify access to these values.

Import the Partner Center SDK from Nuget

1. Right-click the project in the **Solution** view, and then click **Manage Nuget packages**.
2. Search for: *Microsoft.Store.PartnerCenter*, select the version and install.



Here are all the packages and versions:

* *Microsoft.IdentityModel.Clients.ActiveDirectory 2.22.302111727*
* *Newtonsoft.Json 6.0.8*
* *Microsoft.Store.PartnerCenter 1.7.0*

**Important Note**: Due to the large difference between 2.\* and 3.\* for Microsoft.IdentityModel.Clients.ActiveDirectory , please ensure the 2.\* be installed

Obtain an Azure Active Directory User + App token

1. Open the Program.cs file, if it is not already open.
2. Add the following using statements to the Program.cs file:

using Microsoft.IdentityModel.Clients.ActiveDirectory;  
  
These statements will allow you to access objects more easily later in the code.

1. Add the LoginToAad method to the Program.cs file, as a peer of the Main method.

This code uses the user name, password, and the Partner Center application ID to obtain a token from Azure Active Directory.

/// <summary>

/// Login in azure ad and get the credential

/// </summary>

/// <param name="tenantId">the domain</param>

/// <param name="resource">the resouce to get access token</param>

/// <returns></returns>

private static async Task<AuthenticationResult> LoginToAad(string tenantId, string resource)

{

// auth from azure ad

var addAuthority = new UriBuilder(SettingsHelper.AadAuthority + tenantId);

UserCredential userCredentials = new UserCredential(SettingsHelper.UserId,

SettingsHelper.UserPassword);

AuthenticationContext authContext = new AuthenticationContext(addAuthority.Uri.AbsoluteUri);

return await authContext.AcquireTokenAsync(resource,

SettingsHelper.ClientId,

userCredentials);

}

1. Call the LoginToAad method and assign the token to a variable:

Get access token for partner center APIs:

var aadAuthenticationResult = Program.LoginToAad("common",SettingsHelper.PartnerCenterApiResourceId).Result;

Authenticate with the Partner Center SDK

1. Open the Program.cs file, if it is not already open.
2. Add the following using statements to the Program.cs file:

using Microsoft.Store.PartnerCenter;

using Microsoft.Store.PartnerCenter.Extensions;

These namespaces contain the classes and interfaces needed to authenticate with the Partner Center API.

1. In the Main method, create a partner credentials instance using the user credentials token you received from Azure Active Directory:

//login Azure AD, for partner center sdk, you can use the "common" as the tenant id

var aadAuthenticationResult = Program.LoginToAad("common", SettingsHelper.PartnerCenterApiResourceId).Result;

//set to use the public partner center api

PartnerService.Instance.ApiRootUrl = SettingsHelper.PartnerCenterApiEndpoint;

//get the credential

var authToken = new AuthenticationToken(aadAuthenticationResult.AccessToken, aadAuthenticationResult.ExpiresOn);

IPartnerCredentials credentials = PartnerCredentials.Instance.GenerateByUserCredentials(SettingsHelper.ClientId, authToken);

1. Using the generated credentials, create an instance of the *IPartner*:

IPartner partner = PartnerService.Instance.CreatePartnerOperations(credentials);

You can now use the *IPartner* reference to access all the Partner SDK functionality.

## Exercise 3: Create and query paged Customers

In this exercise, you will use the C# console application configured in exercise3 to create new customers. You will then issue queries to retrieve these customers. Finally, you will use Fiddler to inspect the raw HTTP requests made by the SDK.

### Create a Customer Using the Partner Center SDK

1. Open the Program.cs file, if it is not already open.
2. Add the following using statements to the Program.cs file:

using Microsoft.Store.PartnerCenter.Models;

using Microsoft.Store.PartnerCenter.Models.Customers;

These statements will allow you to access objects more easily later in the code.

1. Create a new customer by adding the CreateCustomer method to the Program class as a sibling to the Main method.

/// <summary>

/// Create new customer

/// </summary>

/// <param name="partnerOperations"></param>

/// <returns></returns>

private static Customer CreateCustomer(IPartner partnerOperations)

{

//set the customer's information

var domainPrefix = "EMD18031402";

string strCompnayName = "EMD18031402";

Customer newCustomer = new Customer()

{

// set the company profile

CompanyProfile = new CustomerCompanyProfile()

{

// set the domain

Domain = string.Format(CultureInfo.InvariantCulture, "{0}.onmicrosoft.com", domainPrefix),

// set company name

CompanyName = strCompnayName

},

//set the billing profile

BillingProfile = new CustomerBillingProfile

{

Culture = "en-US",

Language = "en",

FirstName = "Test",

LastName = "EMD",

Email = string.Format("daniel@{0}.onmicrosoft.com", domainPrefix),

CompanyName = strCompnayName,

DefaultAddress = new Address()

{

FirstName = "Test",

LastName = "EMD",

AddressLine1 = "1 Microsoft Way",

AddressLine2 = "Building 1",

City = "Redmond",

State = "WA",

Country = "US",

PostalCode = "98052",

PhoneNumber = "415-555-1212"

}

}

};

return partnerOperations.Customers.Create(newCustomer);

}

You can change the customer information to your own values. The code will create the customer, and then will use a helper method to show the newly created customer to the screen.

**Important Note**: The domain for the customer must be globally unique, rather than simply unique to your CSP Integration Sandbox. Therefore, when implementing the code above, be sure to update the **domain** prefix to a unique value.

1. Update the Main method to call the CreateCustomer function:

var customer = CreateCustomer(partner);

1. Save your changes, and then press **F5** to test the application.

Then you can use the QueryCustomers below to print the customer information.

### Query paged Customers Using the Partner Center SDK

1. Open the Program.cs file, if it is not already open.
2. Add the following using statement to the Program.cs file:

using Microsoft.Store.PartnerCenter.Models.Query;

This statement will allow you to access objects more easily later in the code.

1. Add the QueryCustomer method to the Program class as a sibling to the Main method.

QueryCustomer builds an indexed query that is configured to return paged customers. QueryCustomer then writes the customer list to the screen

/// <summary>

/// Query paged customers via keyword in company name

/// </summary>

/// <param name="partner"></param>

/// <param name="keywords">the keyword in company name</param>

/// <param name="pageSize">page size</param>

private static void QueryCustomers(IPartner partner, string keywords, int pageSize)

{

// query the customer via keyword in company name

var fieldFilter = new SimpleFieldFilter(CustomerSearchField.CompanyName.ToString(), FieldFilterOperation.StartsWith, keywords);

var myQuery = QueryFactory.Instance.BuildIndexedQuery(pageSize, 0, fieldFilter);

var customersPage = partner.Customers.Query(myQuery);

var customersEnumerator = partner.Enumerators.Customers.Create(customersPage);

int i = 0, pageNumber = 0;

while (customersEnumerator.HasValue)

{

++pageNumber;

var customers = customersEnumerator.Current;

StringBuilder report = new StringBuilder();

report.AppendFormat("The #{0} page exists {1} customers\n", pageNumber, customers.TotalCount);

report.AppendFormat("Index,Customer ID,Name,Domain:\n");

foreach (var customer in customers.Items)

{

++i;

try

{

report.AppendFormat("\n#{0},{1},{2},{3}\n", i, customer.Id, customer.CompanyProfile.CompanyName, customer.CompanyProfile.Domain);

// get the customer's subcriptions

var subscriptions = partner.Customers.ById(customer.Id).Subscriptions.Get();

if (subscriptions.TotalCount == 0)

{

report.AppendFormat("\nNo subscription exists\n");

}

else

{

report.AppendFormat("\nSubscription ID,Subscription Name,Subscription Quantity\n");

foreach (var subscription in subscriptions.Items)

{

//add the subscription

report.AppendFormat("{0},{1},{2}\n", subscription.Id, subscription.FriendlyName, subscription.Quantity);

}

}

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Succeed to get the #{0} customer at the #{1} page!", i, pageNumber);

}

catch (PartnerException)

{

//ignore some 403 errors related to Commerce and BEC

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Fail to get the #{0} customer at the #{1} page!", i, pageNumber);

}

Console.ForegroundColor = ConsoleColor.White;

}

//print the report

Console.WriteLine(report);

customersEnumerator.Next();

}

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Press any key to continue...");

Console.ReadLine();

}

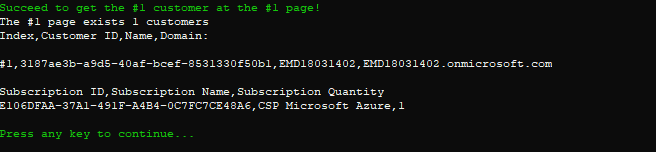
You can change the query to include a filter.

1. Update the Main method to call QueryCustomers.

QueryCustomers(partner, "your company name", 100);

1. Save your changes, and press **F5** to test the application.

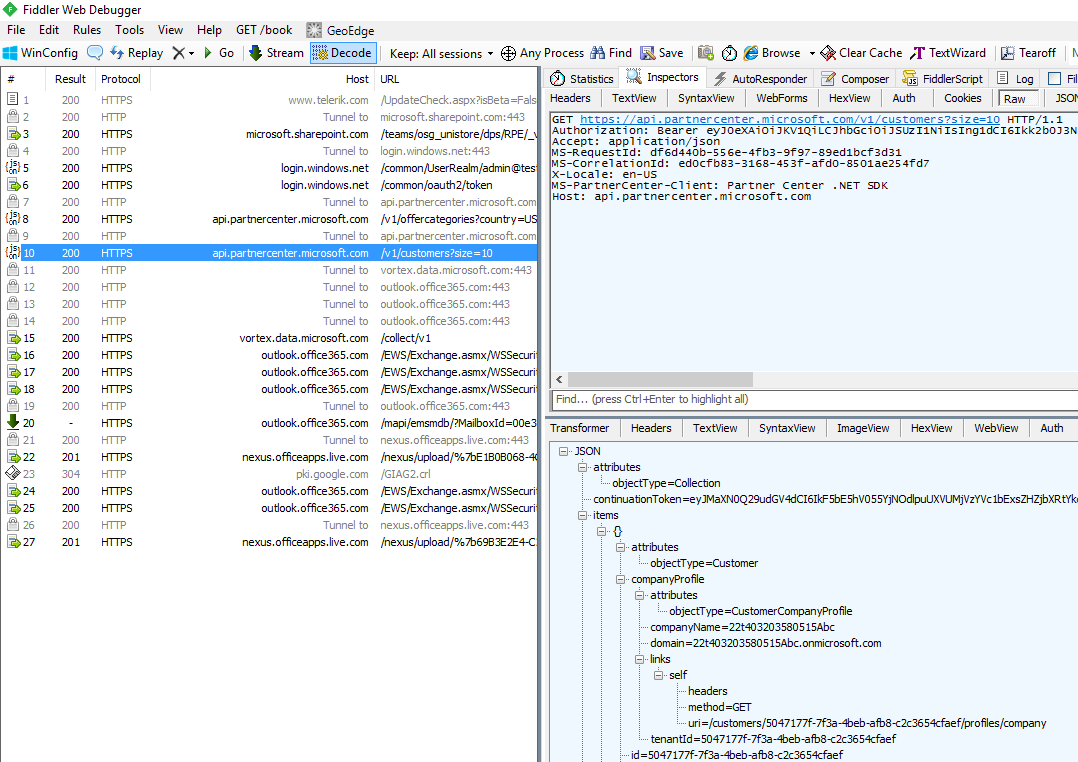
You should see the customer list in the console.



### Inspecting HTTP calls in Fiddler (optional)

1. Start Fiddler and run the console application that you have been coding.

Fiddler will show all the network requests made from your computer. You can find the ones made by the Partner Center SDK by locating requests made to the: *api.partnercenter.microsoft.com* host, and inspecting them to understand the payloads better.



Congratulations! In this exercise, you made your first Partner Center API calls using the Partner Center SDK. The API calls created a customer and retrieved the customer list. You also used Fiddler to inspect the raw HTTP requests made by the SDK.

## Exercise 4: Browse and Select Offers

In this exercise, you will extend the console application from the previous exercise to list the available offers.

### Adding Browse and Select Offers

1. Open the Program.cs file if it is not already open.
2. Add the following using statement to the Program.cs file:  
   using Microsoft.Store.PartnerCenter.Models.Offers;   
     
   These statements will allow you to access objects more easily later in the code.
3. Create the GetOffers method, parallel to the Main method already in the Program class.

/// <summary>

/// get the available offers by country

/// </summary>

/// <param name="partnerOperations"></param>

/// <param name="size"></param>

private static void GetOffers(IPartner partnerOperations, Int32 size)

{

var offers = partnerOperations.Offers.ByCountry("US").Get(0, size);

StringBuilder report = new StringBuilder();

report.Append("Offer ID, Category Name, SalesGroupId, Offer Name\n");

foreach (Offer offer in offers.Items)

{

report.AppendFormat(" {0}, {1}, {2},{3}\n", offer.Id, offer.Category.Name, offer.SalesGroupId, offer.Name);

}

// list offers

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("The available offers...");

Console.WriteLine(report);

}

This code retrieves the offers available in the US for the EN-US locale. The code then writes the offers out to the console.

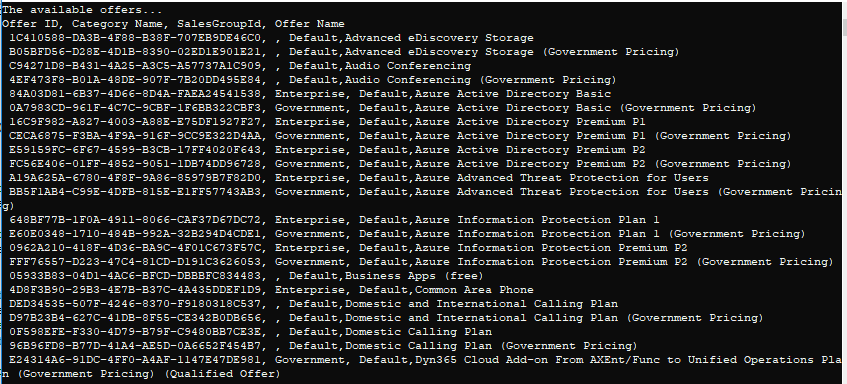
1. Update the Main method to call GetOffers:

GetOffers(partner, 500);

1. Save your changes, and press **F5** to test the application.

The console should display all the offers available to you for the specified region and locale.

Congratulations! In this exercise, you extended the console application created in the previous exercise to list the available Offers.



## Exercise 5: Place an Order for an Azure Subscription

In this exercise, you will extend the console application to place an order for a selected Azure subscription.

1. Open the Program.cs file, if it is not already open.
2. Add the following using statements to the Program.cs file:

using Microsoft.Store.PartnerCenter.Models.Orders;  
  
These statements will allow you to access objects more easily later in the code.

1. Add the PlaceOrder method to the Program class as a sibling to the Main method.

This method will create place a new order for the specified offer and customer:

**Important Note**: If you are using Production environment replace MS-AZR-0146P Offer ID for Azure (this is good for Sandbox) with the MS-AZR-0145P for production Azure subscription.

/// <summary>

/// Create the subscription/order for customer,like azure ,powerbi ,office365 etc

/// </summary>

/// <param name="partnerOperations"></param>

/// <param name="customerId">the customer id</param>

public static void PlaceOrder(IPartner partnerOperations, string customerId)

{

//you can add multi order items in same order

//but only offers with same salesgroupid can put into same order.

//for example , if you try to put the powerbi pro and azure subscritpipon into same order, error will occur

var lineItems = new List<OrderLineItem>();

/\*

\*

lineItems.Add(new OrderLineItem

{

LineItemNumber = 0,

OfferId = "800F4F3B-CFE1-42C1-9CEA-675512810488",

FriendlyName = "Power BI Pro - MSFT",

Quantity = 1

});

lineItems.Add(new OrderLineItem

{

LineItemNumber = 1,

OfferId = "A6ACBC1C-9D2A-482A-ABDA-DFB9285E301E",

FriendlyName = "Power BI Pro - Government",

Quantity = 1

});

\*/

//create the azure subscription

lineItems.Add(new OrderLineItem

{

LineItemNumber = 0,

OfferId = "MS-AZR-0146P",

FriendlyName = "CSP Microsoft Azure",

Quantity = 1

});

// create order

var order = new Order

{

ReferenceCustomerId = customerId,

LineItems = lineItems

};

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("create order...");

try

{

partnerOperations.Customers.ById(customerId).Orders.Create(order);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("order created successfully!");

}

catch (Exception ex)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("error occurs", ex.Message);

}

}

1. Update the Main method to call PlaceOrder, using a valid customer ID:

PlaceOrder(partner, "the customer ID");

**Important Note**: You can find a valid customer ID by looking at the console window from previous tests.

1. Save your changes, and press **F5** to test the application.



Congratulations! In this exercise you took the console application and extended it to place an order for a selected PowerBI offer.

**Important Note**: Only offers with same salesGroupID can be putted into same order, otherwise error will occur.