.NET SDK

This guide explains how to develop applications using the .NET SDK.

Before You Begin

Install the following software on your local machine:

- Microsoft Visual Studio. The instructions in this topic are designed for Microsoft Visual Studio, but you can adapt them for other .NET IDEs.

In addition to the listed software, you need access to aDev Center Account that contains a provisioned sandbox.

Get Started

The following steps guide you through installing the .NET SDK, authenticating your project with your Kibo credentials, and making a call to the API. Specifically, the tutorial demonstrates how to create a console application that retrieves the number of customer accounts for a Kibo eCommerce site, teaching you the necessary concepts for then building a fully-fledged application of your own.

Create an application in Dev Center with the appropriate behaviors:

1. Log in to Dev Center.
2. Create a new application.
3. Add the `Customer Read` behavior to the application. This step is necessary to give your application the necessary permissions to read customer accounts. If you design additional functionality for your application, such as updating an order, add the appropriate behaviors to avoid a permissions error.
4. Install the application to the sandbox of your choice.
5. Enable the application in Admin. If you decide to add additional behaviors to your application after this step, you must reinstall the application to your sandbox and re-enable the application in Admin to apply the new behaviors.
6. Note the application key, shared secret, tenant ID, and site ID. You can obtain the application key and shared secret from the application details page. You can obtain the tenant ID and site ID by viewing your live site and looking at the URL, which has the pattern `t`*`TenantID`*`-s`*`SiteID`*`.sandbox.mozu.com`. You can obtain the master catalog ID through a GetTenant API call, which also returns the tenant ID and site ID, but the master catalog ID is not required for the API call used in this tutorial.

Create a .NET application that uses the .NET SDK:

1. Create a new project in Visual Studio.
2. Choose to develop a **Console Application** and click **OK**.

3. Open the NuGet packet manager (**Tools** > **Library Package Manager** > **Manage NuGet Packages for Solution**).
4. Search for *Mozu* in the online package search box.
5. Install the **Mozu.Api.SDK** and the **Mozu.Api.ToolKit** packages, and then close out of the NuGet packet manager. To learn about the available packages, refer to the About the Toolkits section.
6. Open the `App.config` file in your solution root directory for editing. This is the file where you specify the configuration data for your application.
7. Within `App.config`, specify your application configuration within an `appSettings` block inside of the `configuration` block, as shown in the following example, replacing the placeholder values with your application-specific values (leave the `startup` and `runtime` blocks as is). You always need to specify an application key, shared secret, and tenant ID for your application to make successful calls to the API. The site ID is required for most API calls, but not all, and the master catalog ID is required for some API calls, but not all.

```
<configuration>
   <startup>
      ...
   </startup>
   <appSettings>
   <add key="ApplicationId" value="yourApplicationKey" />
   <add key="SharedSecret" value="yourSharedSecret" />
   <add key="TenantId" value="yourTenantId" />
   <add key="SiteId" value="yourSiteId" />
   <add key="MasterCatalogId" value="yourMasterCatalogId"< /// not required for this tutori
al - shown for educational purposes
   </appSettings>
   <runtime>
      ...
   </runtime>
</configuration>
```

Add a class that inherits from AbstractBootstrapper.cs, which loads dependency injections and leverages the Autofac IoC container.

1. Right-click your project directory in Solution Explorer and select **Add** > **Class**.
2. Name the class *Bootstrapper.cs* and click **OK**.
3. Code the `Bootstrapper.cs` file so that it matches the following example, making sure to replace the namespace value with your application name:

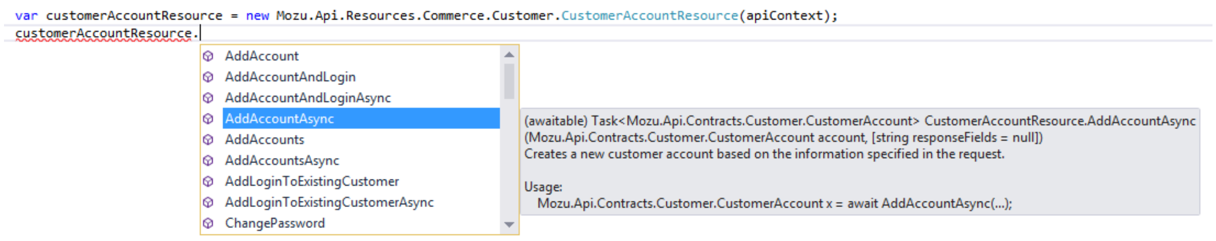
```
using Mozu.Api.ToolKit;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace yourAppName /// replace with your application name
{
   class Bootstrapper : AbstractBootstrapper
   {
   }
}
```

Make a call to the API in your main program file:

1. Code your main program file to match the following example, which obtains an API context, creates a customer account resource, and uses the `GetAccountsAsync` function to return the number of customer accounts on the tenant:

```
using System;
using System.Configuration;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Autofac;
using Mozu.Api;
using Mozu.Api.ToolKit.Config;

namespace yourAppName /// replace with your application name
{
    class Program
    {
        private static IApiContext _apiContext { get; set; }
        private static IContainer _container { get; set; }

        static void Main(string[] args)
        {
            // generate an API context based on config data
            var apiContext = Program.GenerateApicontext();

            // create a customer account resource
            var customerAccountResource = new Mozu.Api.Resources.Commerce.Customer.Custo
merAccountResource(apiContext);

            // get the collection of customer accounts
            var customerAccountCollection = customerAccountResource.GetAccountsAsync(page
Size: 200).Result; /// note the recommended use of the async versions of functions, rather tha
n the deprecated sync versions of functions

            // log the total number of customer accounts
            Console.WriteLine("Number of Customer Accounts:");
            Console.WriteLine(customerAccountCollection.TotalCount);

            Console.ReadLine();
        }

        private static IApiContext GenerateApicontext()
        {
            _container = new Bootstrapper().Bootstrap().Container;
            var appSetting = _container.Resolve();

            var tenantId = int.Parse(appSetting.Settings["TenantId"].ToString());
            var siteId = int.Parse(appSetting.Settings["SiteId"].ToString());
            _apiContext = new ApiContext(siteId: siteId, tenantId: tenantId);
            return _apiContext;
        }
    }
}
```

2. Click **Start** to run your application. After it builds, the application logs the number of customer accounts to the console.



```
var customerAccountResource = new Mozu.Api.Resources.Commerce.Customer.CustomerAccountResource(apiContext);
customerAccountResource.
```

```
(awaitable) Task<Mozu.Api.Contracts.Customer.CustomerAccount> CustomerAccountResource.AddAccountAsync
(Mozu.Api.Contracts.Customer.CustomerAccount account, [string responseFields = null])
Creates a new customer account based on the information specified in the request.

Usage:
  Mozu.Api.Contracts.Customer.CustomerAccount x = await AddAccountAsync(...);
```

About the Toolkits

NuGet Package Name	Description
Mozu.Api.SDK	The .NET SDK package. View the source on GitHub
Mozu.Api.ToolKit	Provides dependency injection containers and application eventing. View the source on GitHub
Mozu.Api.WebToolKit	Provides controllers and classes for building an MVC application. View the source on GitHub

SDK Function Reference

The Kibo .NET SDK provides dozens of built-in functions that help you interact with the API quickly and easily. To familiarize yourself with these functions:

- Use Visual Studio's autocomplete feature to discover the functions available to you. The majority of resources (that contain functions) are located under `Mozu.Api.Resources` and the majority of properties that you send to the API as JSON are located under `Mozu.Api.Contracts`. These objects mirror the structure of the API.
- View the SDK source on GitHub, and either dig through the source files or use the repository search box to find the function you are looking for.

Always use the asynchronous functions provided by the .NET SDK (denoted by the*Async* suffix). Unlike the deprecated synchronous functions, the asynchronous functions do not have to execute sequentially.