MYOB Advanced

Part 2- Web Services Advanced

Contents

Introduction

External systems can use MYOB Advanced integration interfaces to access the business functionality and data of MYOB Advanced. MYOB Advanced provides the following integration interfaces:

- The Open Data (OData) interface
- The contract-based REST API
- The contract-based SOAP API
- The screen-based SOAP API

This course shows advanced techniques of the data retrieval with the contract-based REST and SOAP APIs. One of the data retrieval scenarios is implemented with the screen-based SOAP API. This course does not cover the implementation of the integration scenarios with the OData interface. The basic information about the data retrieval through all these integration interfaces is available in the *I300 Web Services: Basic | Data Retrieval* training course, which is a prerequisite for this course.

The course is intended for developers who need to create applications that interact with MYOB Advanced.

The course is based on a set of examples of web integration scenarios that demonstrate the processes involved in developing a client application that uses MYOB Advanced integration interfaces. The course gives you ideas about how to develop your own applications by using the web services APIs. As you go through the course, you can complete the examples for a particular integration interface or for multiple integration interfaces.

After you complete all the lessons of the course, you will be familiar with the advanced techniques of data retrieval through the MYOB Advanced web services APIs.

How to Use This Course

To complete the course, you will complete the lessons from each part of the course in the order in which they are presented and pass the assessment test. More specifically, you will do the following:

1. Complete *Course Prerequisites*, and carefully read *Company Story and MyStoreIntegration Application*.
2. Complete the lessons in all parts of the training guide. In each lesson, you should review the description of the lesson, at least one of the examples for the integration interface that you are interested in, and the lesson summary. You can skip the other examples of the lesson, or you may prefer to review multiple examples in the lesson if you are looking for the best solution for your integration scenario.

What Is in a Part?

The first part of the course explains how to authorize a third-party application to work with the MYOB Advanced integration services.

The other parts of the course are dedicated to the implementation of particular web integration scenarios that you may need to implement in a third-party application that integrates an external system with MYOB Advanced.

Each part of the course consists of lessons you should complete.

What Is in a Lesson?

Each lesson is dedicated to a particular web integration scenario that you can implement by using the web services APIs. Each lesson consists of a brief description of the web integration scenario and examples of the implementation of this scenario.

The lesson may also include *Additional Information* topics, which are outside of the scope of this course but may be useful to some readers.

Each lesson ends with a *Lesson Summary* topic, which summarizes the possible options for the implementation of the web integration scenario with different integration interfaces.

What Are the Documentation Resources?

All the links listed in the *Related Links* sections refer to the documentation available on the *https://help.acumatica.com/* website. These and other topics are also included in the MYOB Advanced instance, and you can find them under the **Help** menu.

Course Prerequisites

To complete this course, you should be familiar with the basic principles of data retrieval with the MYOB Advanced integration services. We recommend that you complete the *I300 Web Services: Basic | Data Retrieval* training course before you begin this course.

You should complete this course on MYOB Advanced 2019. The following prerequisites should be met before you start with this course:

1. You need to deploy an instance of MYOB Advanced with the name *MyStoreInstance*.
2. The Postman application should be installed on your computer if you are going to complete the examples that illustrate the use of the REST API. To download and install Postman, follow the instructions on *https://www.getpostman.com/apps*.
3. Microsoft Visual Studio 2015 or later should be installed on your computer if you are going to complete the examples that illustrate the use of the SOAP API.

   The instructions in this guide are designed for Microsoft Visual Studio 2017. If you use a different version of Visual Studio, the menu paths and the user interface may differ.
4. You must have HTTP access from the computer where you work with the examples to the MYOB Advanced instance so that you can request the integration services.
5. Because in the examples of this course you will use the OAuth 2.0 authorization, you have to configure HTTPS on the MYOB Advanced website, as described in *Configuring a Website for HTTPS*. If you do not want to use OAuth 2.0 authorization, your application can use the API methods for the sign-in and sign-out (which were described in the *I300 Web Services: Basic | Data Retrieval* training course) and interact with MYOB Advanced via HTTP.

Deploying an MYOB Advanced Instance for the Training Course

You deploy an MYOB Advanced instance and configure it as follows:

1. Open the MYOB Advanced Configuration Wizard, and deploy a new application instance as follows:
    a. On the **Database Configuration** page of the MYOB Advanced Configuration Wizard, type the name of the database: `MyStoreInstance`.
    b. On the **Tenant Setup** page, set up one tenant:
        - **Login Tenant Name**: `MyStore`
        - **New**: Selected
        - **Parent Tenant ID**: 2
        - **Visible**: Selected

    The system creates a new MYOB Advanced instance, adds a new tenant, and loads the selected data.
2. Sign in to the new tenant by using the following credentials:
    - Login: `admin`
    - Password: `setup`

    Change the password when the system prompts you to do so.
3. Click the user name in the top right corner of the MYOB Advanced window and click **My Profile**. On the **General Info** tab of the User Profile (SM203010) form, which opens, select *MYSTORE* in the **Default Branch** box; then click **Save** on the form toolbar. In subsequent sign-ins to this account, you will be signed in to this branch.

Configuring a Website for HTTPS

In the examples of this guide, you will use the secure connection between the API client application and MYOB Advanced.

A secure connection between the client application and the MYOB Advanced website with a Secure Socket Layer (SSL) certificate is required for the authorization of the client application through OAuth 2.0.
Therefore, you have to set up the MYOB Advanced website for HTTPS, as described in this topic.

As the Microsoft IIS documentation states, the steps for configuring SSL for a site include the following:

1. You obtain an appropriate certificate. (For the purposes of completing the course, you can create a self-signed server certificate.)
2. You create an SSL binding on a site.
3. You test the website by making a request to the site.
4. Optional: You configure the SSL options.

To complete the examples of this guide, you should create a self-signed certificate and configure SSL binding as follows:

1. Create a self-signed certificate by doing the following:
    a. In the Control Panel, open **Administrative Tools** > **Internet Information Services (IIS) Manager**.
    b. In the **Features View**, double-click **Server Certificates**.
    c. Click **Create Self-Signed Certificate** in the **Actions** pane.
    d. Enter a name for the new certificate, and click **OK**.
2. Do the following to create an SSL binding:
    a. Select a site in the tree view, and click **Bindings** in the **Actions** pane.
    b. In the **Site Bindings** dialog box, click **Add** to add your new SSL binding to the site.
    c. In the **Type** drop-down list, select *https*.
    d. Select the self-signed certificate you created, and click **OK** to close the dialog box.
3. In the **Actions** pane, under **Browse Web Site**, click the link associated with the binding you just created (*Browse*:443(https)*). The browser will display an error page because the self-signed certificate was issued by your computer rather than by a trusted certificate authority.
4. Click the link to proceed with this website and disregard the error. The HTTPS website opens.

Company Story and MyStoreIntegration Application

In this course, you will simulate the integration of MYOB Advanced with the online store of a small retail company, MyStore. This company is a single business entity that has no branches or subsidiaries. MyStore uses MYOB Advanced for customer management, sales order processing, and payment collection.

MyStore plans to extend its business and start selling goods online. MyStore needs to investigate the options available in MYOB Advanced for integration with eCommerce applications. In the first stage of implementation, the integration application, which MyStore is developing, should retrieve information about stock items, sales orders, and payments from MYOB Advanced. In the second stage of implementation, the integration application should submit information about customers, sales
orders, and payments from the online store to MYOB Advanced. This course covers only the first stage of implementation. The second stage is covered in the *I320 Web Services: Data Manipulation* training course.

The MyStoreIntegration application, which you will build as you complete the course, will integrate MYOB Advanced with the online store of the MyStore company. For the implementation of the MyStoreIntegration application, the MyStore company can use one of the following interfaces:

- Contract-based REST API
- Contract-based SOAP API
- Screen-based SOAP API

The OData interface can be used only for the implementation of the data retrieval part of the MyStoreIntegration application, while the data submission should be performed by other integration interfaces because the data submission is not possible through OData. This implementation is outside of the scope of this course.

The examples of this course show the implementation of the MyStoreIntegration application with the contract-based REST and SOAP APIs. One of the integration scenarios (the retrieval of reports) is implemented with the screen-based SOAP API.

The following diagram shows how the MyStoreIntegration integration application fits in the integration of the MyStore online store with MYOB Advanced.

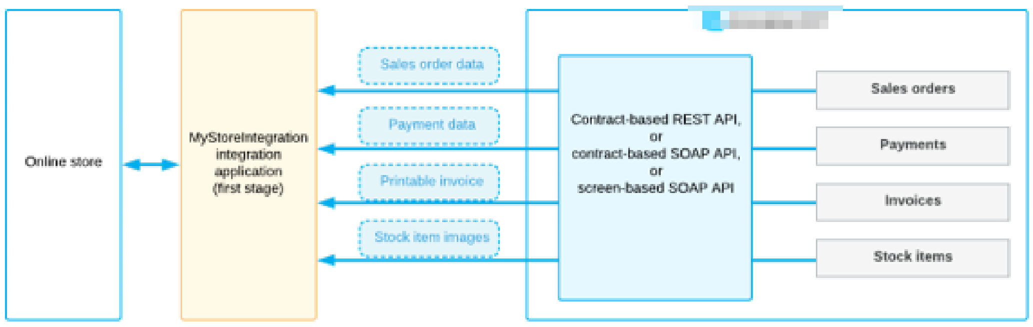

Online store
MyStoreIntegration integration application (first stage)
Sales order data
Payment data
Printable invoice
Stock item images
Contract-based REST API, or contract-based SOAP API, or screen-based SOAP API
Sales orders
Payments
Invoices
Stock items

Figure: Integration of the MyStore online store and MYOB Advanced

Integration Requirements

At the first stage of implementation, the MyStoreIntegration application should implement integration with MYOB Advanced to support the following usage scenarios in the online store:

- A registered customer should be able to view all of this customer's purchases.
- A registered customer should be able to view all of this customer's payments.
- A registered customer should be able to receive a printable version of the invoice for the ordered goods.
- A potential customer should be able to view the image of any selected item.

In this course, you do not implement the online store application itself; instead, you implement the integration part between the online store and MYOB Advanced. The integration part provides the
support for the listed scenarios in the online store application. (The implementation of the online store application is outside of the scope of this course.)

Part 1: Authorization of the Application to Work with the Web Service

Before an application can retrieve data from MYOB Advanced by using the integration interfaces provided by MYOB Advanced, the application needs to sign in to MYOB Advanced. You can implement the signing
in to and signing out from MYOB Advanced by using the authentication methods of the corresponding integration interfaces. (These methods are described in the *I300 Web Services: Basic | Data Retrieval* training course and in the documentation.)

Alternatively, you can authorize the application to work with the integration interfaces by using the OAuth 2.0 authorization mechanism. With OAuth 2.0, the client application will not operate with the MYOB Advanced credentials to sign a user in to MYOB Advanced; instead, the application will obtain an access token from MYOB Advanced and will use this token when it requests data from MYOB Advanced.

Authorization through OAuth 2.0 can be used for integrations through the contract-based APIs, but it cannot be used for the OData and screen-based API integrations.

In this part of the guide, you will register the MyStoreIntegration application in MYOB Advanced and configure the application to use the OAuth 2.0 mechanism of authorization. The MyStoreIntegration application will use the resource owner password credentials flow. With this flow, the credentials (username and password) of an MYOB Advanced user are provided directly to the client application, which uses the credentials to obtain the access token. For details about this flow, see *Resource Owner Password Credentials Flow* in the documentation.

According to the OAuth 2.0 specification, a secure connection between an OAuth 2.0 client application and the MYOB Advanced website with a Secure Socket Layer (SSL) certificate is required. You have set up the MYOB Advanced website for HTTPS before you started this course (as described in *Configuring a Website for HTTPS*).

Lesson 1.1: Registering the Application in MYOB Advanced

In this lesson, you will register the MyStoreIntegration application in MYOB Advanced as a connected application that uses the OAuth 2.0 authorization. To register this application, you will use the Connected Applications (SM303010) form. The application will use the resource owner password credentials flow.

When you are registering the client application, you have to be signed in to the tenant whose data the client application needs to access, because the client ID that is generated during the application registration includes the name of the tenant. In the instance that you use for the training course, there is only one tenant configured (with the name *MyStore*).

Lesson Objective

In this lesson, you will learn how to register a client application in MYOB Advanced.

Example 1.1.1: Registering the Application in MYOB Advanced

In this example, you will register the MyStoreIntegration application on the Connected Applications (SM303010) form.

Registering a Connected Application

Proceed as follows:

1. In the Summary area of the Connected Applications (SM303010) form, specify the following values:
   - **Client Name**: `MyStoreIntegration`
   - **OAuth 2.0 Flow**: *Resource Owner Password Credentials*
2. On the **Secrets** tab, click **Add Shared Secret**.
3. In the **Add Shared Secret** dialog box, which opens, do the following:
   a. In the **Description** box, type `MyStoreIntegration Secret`.
   b. Copy and save the value from the **Value** box.

      For security reasons, the value of the secret is displayed only once: when you create the secret by invoking this dialog box. Therefore, if you do not save the secret, you will not be able to obtain its value in the future.

   c. Click **OK**.
4. On the form toolbar, click **Save**. Notice that the client ID has been generated and inserted in the **Client ID** box. The name of the tenant to which you are signed in is appended to this client ID. The MyStoreIntegration application will use this client ID along with the client secret for authentication in MYOB Advanced.

Related Links

To Register a Client Application

Additional Information: OAuth 2.0 Authorization

The scenarios described in this topic are outside of the scope of this course but may be useful to some readers.

Use of the Authorization Code Flow and the Implicit Flow

Instead of the resource owner password credential flow, a client application that implements the OAuth 2.0 authorization mechanism can use one of the following OAuth 2.0 authorization flows supported by MYOB Advanced:

- Authorization code: With this authorization flow, the client application never gets the credentials of the applicable MYOB Advanced user. After the user is authenticated in MYOB Advanced, the client application receives an authorization code, exchanges it for an access token, and then uses the access token to work with data in MYOB Advanced. For details on the flow, see *Authorization Code Flow*.
- Implicit: With the implicit flow, the client application never gets the credentials of the applicable MYOB Advanced user. When the user is authenticated in MYOB Advanced, the client application does not receive an authorization code (as with the authorization code flow); instead, the client application directly receives an access token, and then uses the access token to work with data in MYOB Advanced. For more information about the flow, see *Implicit Flow*.

For a comparison of all supported flows, see *Comparison of the Flows*.

Revoking of the Access of a Connected Application

You can revoke the access of a connected application that you have registered on the Connected Applications (SM303010) form. For details about how to do this, see *To Revoke the Access of a Connected Application*.

Lesson Summary

In this lesson, you have learned how to register in MYOB Advanced a client application that uses the OAuth 2.0 authorization. During the registration, you have been signed in to the MyStore tenant, whose data the MyStoreIntegration application needs to access.

You have also reviewed the possible options of the OAuth 2.0 authorization.

The following table summarizes the availability of the OAuth 2.0 authorization method for each of the integration interfaces.

Integration Interface	OAuth 2.0 Authorization
OData interface	No
REST API	Yes
Contract-based SOAP API	Yes
Screen-based SOAP API	No

Lesson 1.2: Configuring the Application to Use OAuth 2.0

In this lesson, you will configure the MyStoreIntegration application to use the OAuth 2.0 authorization as follows:

- For the contract-based REST integration application, you will configure a Postman collection to use OAuth 2.0.
- For the contract-based SOAP integration application, you will add to the MyStoreIntegration project a C# code that obtains an access token from MYOB Advanced.

You will connect to the token endpoint, pass the client ID and client secret in the authorization header, and request access to the web services API (that is, you will request the `api` scope). You will receive the access token from MYOB Advanced to use it in subsequent requests to MYOB Advanced. You will not request the refresh token, which the client application can use to request a new access token when the access token has expired.

Lesson Objective

In this lesson, you will learn how to configure an integration application to use OAuth 2.0 for authorization in MYOB Advanced.

Example 1.2.1: Configuring the REST Application to Use OAuth 2.0

In this example, you will configure a Postman collection to use the OAuth 2.0 authorization for the requests to MYOB Advanced.

You will connect directly to the token endpoint. In Postman, you cannot use the discovery endpoint, which is *https://<MYOB Advanced instance URL>/identity/*, because Postman does not support OpenID Connect Discovery. (In the discover endpoint address, *<MYOB Advanced instance URL>* is the URL of the MYOB Advanced instance to which the client application is going to connect.) If the client application supports OpenID Connect Discovery, we recommend that the client application use the discovery endpoint address to obtain the token endpoint address. The use of the discovery endpoint eliminates the need to change the application if the address of the token endpoint changes. An example of the use of the discovery endpoint is provided for the contract-based SOAP API in *Example 1.2.2: Configuring the SOAP Application to Use OAuth 2.0*.

You will configure the Postman collection to request the `api` access scope, which provides access to the web services APIs. You will not use the `offline_access` scope, which requests that a refresh token be granted.

Configuring a Postman Collection

To configure a Postman collection to use the OAuth 2.0 authorization in MYOB Advanced, do the following:

1. If you use a self-signed certificate for HTTPS, in Postman settings, turn off SSL certificate verification.
2. In Postman, create a collection.

   Instead of creation of a collection, you can import to Postman the collection provided with this course (`MyStoreIntegration\REST.postman_collection.json`). This collection is configured for this course and already contains all the requests that are used in this course. You can use this collection for testing of the requests.
3. On the **Authorization** tab of the **Edit Collection** dialog box, do the following:
   a. Select the following values:
      - **Type**: *OAuth 2.0*
      - **Add auth data to**: *Request Headers*

   b. Click **Get New Access Token**.
4. In the **Get New Access Token** dialog box, which opens, specify the following values:
   - **Token Name**: `MyStoreIntegration`
   - **Grant Type**: *Password Credentials*
   - **Access Token URL**: `https://localhost/MyStoreInstance/identity/connect/token`
   - **Username**: `admin`
   - **Password**: The password for the `admin` user
   - **Client ID**: The client ID of the application, which you can copy from the **Client ID** box on the Connected Applications (SM303010) form for the MyStoreIntegration client (which you have created in *Example 1.1.1: Registering the Application in MYOB Advanced*)
   - **Client Secret**: The client secret that you have received and saved during the registration of the MyStoreIntegration client on the Connected Applications form
   - **Scope**: `api`

- **Client Authentication**: *Send as Basic Auth header*

5. Click **Request Token**. Once the token is received, the **Manage Access Tokens** dialog box opens.

   In certain versions of Postman, the approach described above does not work. Instead of this approach, you can send a direct `POST` request to the token endpoint. In this request, you should pass the client ID, client secret, MYOB Advanced username and password, the type of the authorization flow, and the requested scope in the body of the request. For an example of this
   request, see the Postman `MyStoreIntegration\REST.postman_collection.json` collection provided with this course. For details about the parameters passed in the request body, see *Resource Owner Password Credentials Flow* in the documentations.

6. In the **Manage Access Tokens** dialog box, click **Use Token**.
7. In the **Edit Collection** dialog box, click **Update**.

Related Links

Resource Owner Password Credentials Flow

Example 1.2.2: Configuring the SOAP Application to Use OAuth 2.0

In this example, you will configure the MyStoreIntegration contract-based SOAP application to use OAuth 2.0 for authorization in MYOB Advanced.

You will use the discovery endpoint address, which is *https://<MYOB Advanced instance URL>/identity/*. (In this address, *<MYOB Advanced instance URL>* is the URL of the MYOB Advanced instance to which the client application is going to connect.) The MyStoreIntegration application will use the discovery endpoint address to find out the token endpoint address. We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the token endpoint address changes.

You will configure the MyStoreIntegration application to request the `api` access scope, which provides access to the web services APIs. You will not request the `offline_access` scope, which requests that a refresh token be granted.

Configuring the Application

To configure the MyStoreIntegration application, do the following:

1. Open Visual Studio, and create a new Visual C# console application from the *Console App (.Net Framework)* template with the *MyStoreIntegration* name.

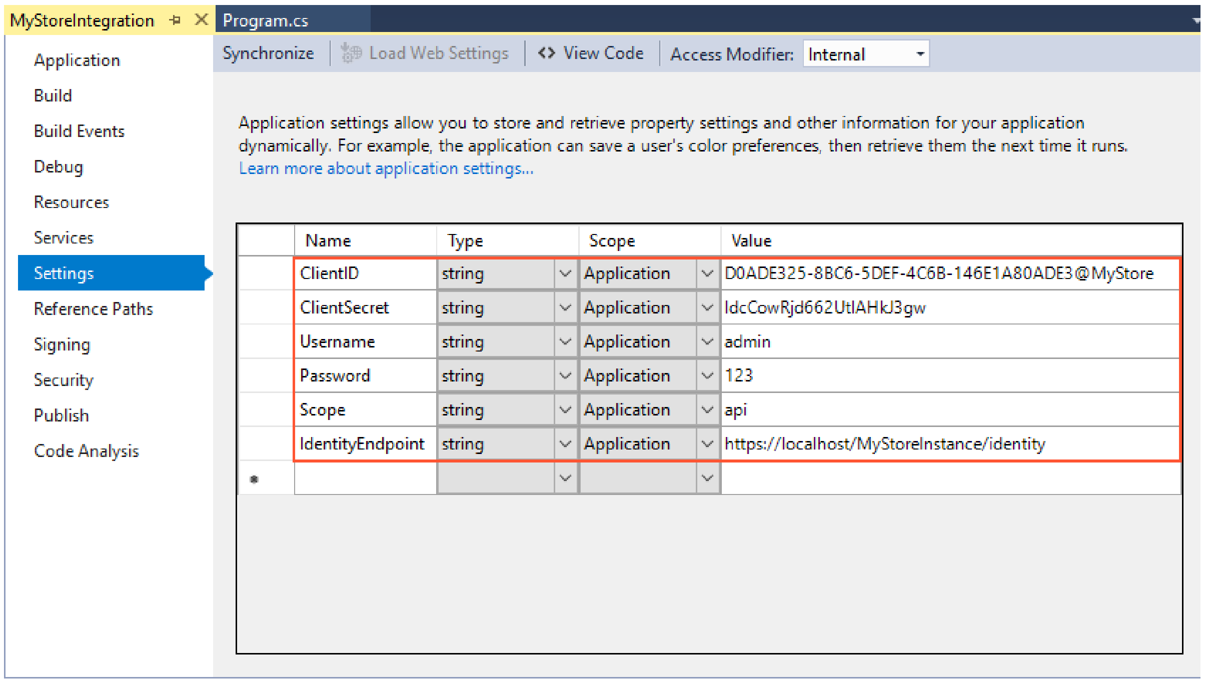
   Instead of creating a Visual Studio project, you can use the solution provided with this course (`MyStoreIntegration\CBAPI\MyStoreIntegration.sln`). This solution is configured for this course and already contains all the methods that are used in this course. You can use this solution for testing.

2. Configure the settings of the *MyStoreIntegration* project as follows:

   a. Right-click the **MyStoreIntegration** project folder in Solution Explorer and select **Properties** from the context menu. On the **MyStoreIntegration** tab, which opens, click **Settings** and then **This project does not contain a default settings file. Click here to create one.** Visual Studio creates the `Settings.settings` file in the `Properties` folder of the *MyStoreIntegration* project and opens the file.

   b. Add to the application settings the `IdentityEndpoint`, `ClientID`, `ClientSecret`, `Username`, `Password`, and `Scope` properties, which are shown in the following screenshot. Specify the values of the properties in the **Value** column as follows:

      - As the *IdentityEndpoint*, type `https://localhost/MyStoreInstance/identity`, which is the URL of the identity endpoint in this example.
      - As the *ClientID*, type the client ID that was generated in the **Client ID** box of the Connected Applications (SM303010) form when you registered the client application in *Example 1.1.1: Registering the Application in MYOB Advanced*.
      - As the *ClientSecret*, type the client secret that you saved when you registered the client application in *Example 1.1.1: Registering the Application in MYOB Advanced*.
      - As the *Username*, type the username (in this example, `admin`) that the application should use to sign in to MYOB Advanced.
      - As the *Password*, type the password for the username.
      - As the *Scope*, type `api`.



Access Modifier: Internal

Application settings allow you to store and retrieve property settings and other information for your application dynamically. For example, the application can save a user's color preferences, then retrieve them the next time it runs.
Learn more about application settings...

Name	Type	Scope	Value
ClientID	string	Application	D0ADE325-8BC6-5DEF-4C6B-146E1A80ADE3@MyStore
ClientSecret	string	Application	IdcCowRjd662UtIAHkJ3gw
Username	string	Application	admin
Password	string	Application	123
Scope	string	Application	api
IdentityEndpoint	string	Application	https://localhost/MyStoreInstance/identity

Figure: Application settings

3. Install the IdentityModel NuGet package.

   To install the package, do the following:

   1. Right-click the **MyStoreIntegration** project folder in Solution Explorer and click **Manage NuGet Packages**.
   2. On the **Browse** tab of the **NuGet Package Manager**, which opens, search for `IdentityModel`.
   3. Click the IdentityModel library in the list of found records and click **Install**.

4. In the `Main()` method of the `Program` class, add the following code, which obtains the access token.

```
using System;
using IdentityModel.Client;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
using System.ServiceModel.Channels;
using System.Net;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;

namespace MyStoreIntegration
{
    class Program
    {
        static void Main(string[] args)
        {
            //This code is necessary only if you connect to the website
            //through the HTTPS connection and
            //you need to use custom validation of an SSL certificate
            //(for example, if the website uses a self-signed certificate).
            ServicePointManager.ServerCertificateValidationCallback += new
            RemoteCertificateValidationCallback(ValidateRemoteCertificate);

            //Discover the token endpoint
            DiscoveryClient discoveryClient = new DiscoveryClient(
              Properties.Settings.Default.IdentityEndpoint);
            DiscoveryResponse discoveryResponse =
              discoveryClient.GetAsync().Result;

            //Obtain and use the access token
            using (TokenClient tokenClient =
                new TokenClient(discoveryResponse.TokenEndpoint,
                Properties.Settings.Default.ClientID,
                Properties.Settings.Default.ClientSecret))
            {
                tokenClient.BasicAuthenticationHeaderStyle =
                  BasicAuthenticationHeaderStyle.Rfc2617;
                var result = tokenClient
                .RequestResourceOwnerPasswordAsync(
                  Properties.Settings.Default.Username,
                  Properties.Settings.Default.Password,
                  Properties.Settings.Default.Scope
                ).Result;

                string accessToken = result.AccessToken;

                //You will add the integration code here
            }
        }

        //A callback, which is used to validate the certificate of
        //an MYOB Advanced website in an SSL conversation
        private static bool ValidateRemoteCertificate(object sender,
        X509Certificate cert, X509Chain chain, SslPolicyErrors policyErrors)
        {
            //For simplicity, this callback always returns true.
            //In a real integration application, you must check the SSL
            //certificate here.
            return true;
        }
    }
}
```

5. Rebuild the project.
6. Insert a breakpoint in the line `string accessToken = result.AccessToken;` and run the application. When the breakpoint is hit, make sure `result.AccessToken` is not `null`.

Related Links

Resource Owner Password Credentials Flow

Lesson Summary

In this lesson, you have learned how to configure an integration application to use OAuth 2.0 for authorization in MYOB Advanced. You have connected to the token endpoint, passed the client ID and client secret in the authorization header, and requested access to the web services API. You have received the access token from MYOB Advanced. You will use this token in subsequent requests to MYOB Advanced.

Lesson 1.3: Signing Out from MYOB Advanced

In this lesson, you will add to the MyStoreIntegration REST and SOAP applications the requests that sign out from MYOB Advanced.

If you have granted only the `api` scope to the application, the access token of the application expires in one hour and the session that was opened for this access token is closed automatically. However, if the application has been granted only the `api` scope, we recommend that you call the sign-out method after you have finished your work with MYOB Advanced, because the MYOB Advanced license includes a limit for the number of API users. If you have not signed out, you may have issues with subsequent authorization requests or sign-ins through APIs. For details about how to deal with the issues related to the limit for the number of API users during the authorization requests, see *Appendix: Troubleshooting*.

However, if you authorize your integration application to work with MYOB Advanced through OAuth 2.0, the sign-out is not always required after you have finished your work with MYOB Advanced. (This is opposed to the situation when your integration application uses the API methods for the sign-in in MYOB Advanced—that is, uses cookies to manage the application sessions. For these applications, the sign-out is required to close the session each time the work with MYOB Advanced is finished.) For details about when the sign-out is required, see *Additional Information: Session Management in the OAuth 2.0 Applications*.

Lesson Objective

In this lesson, you will learn how to sign out from MYOB Advanced in an OAuth 2.0 application.

Example 1.3.1: Using the REST API

In this example, through the REST API, you will sign out from MYOB Advanced.

To sign out from MYOB Advanced, you will use the `POST` HTTP method and the endpoint for signing out.

Signing Out from MYOB Advanced

To sign out from MYOB Advanced, do the following:

1. In the Postman collection, add a new request, and configure the settings of the request as follows:
   - HTTP method: `POST`
   - URL: *https://localhost/MyStoreInstance/entity/auth/logout*
   - Headers:

Key	Value
`Accept`	`application/json`
`Content-Type`	`application/json`

   By default, the request in the collection uses the authorization type specified for the collection—that is, the OAuth 2.0 type, which you have specified in *Example 1.2.1: Configuring the REST Application to Use OAuth 2.0*.

2. Send the request. The response of the successful request includes the `204 No Content` status code.
3. Save the request.

Related Links

Logout from the Service

Example 1.3.2: Using the Contract-Based SOAP API

In this example, you will import the WSDL description of the *Default/18.200.001* system endpoint into the *MyStoreIntegration* project and add to the project the code that attaches the access token to each request to MYOB Advanced and signs out from MYOB Advanced when the work is finished. To sign out from MYOB Advanced, you will use the `Logout()` method of a `DefaultSoapClient` object. We recommend that you call the `Logout()` method in the `finally` block.

Importing the WSDL File into a Visual Studio Project

To add a reference to the web service, proceed as follows:

1. Add to the *MyStoreIntegration* project the *Default* service reference to the *Default/18.200.001* system endpoint. (Use the HTTPS address of the endpoint.)

   For details about how to add the service reference to the project, see *To Configure the Client Application* in the documentation or the *I300 Web Services: Basic | Data Retrieval* training course.

2. Modify the `app.config` file of the project as follows. You use the `Transport` security mode for the HTTPS connection.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  ...
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="Acumatica"
         maxReceivedMessageSize="6553600">
          <security mode="Transport" />
        </binding>
        ...
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address=
       "https://localhost/MyStoreInstance/entity/Default/18.200.001"
       binding="basicHttpBinding" bindingConfiguration="Acumatica"
       contract="MyStore.DefaultSoap" name="DefaultSoap" />
    </client>
  </system.serviceModel>
</configuration>
```

3. Rebuild the project.

Signing Out from MYOB Advanced

To add the code for the sign-out from MYOB Advanced, proceed as follows:

1 In the `Program` class, add the code marked in bold type below.

```
...
using MyStoreIntegration.Default;

namespace MyStoreIntegration
{
    class Program
    {
        static void Main(string[] args)
        {
            ...
            //Obtain and use the access token
            using (TokenClient tokenClient =
                new TokenClient(discoveryResponse.TokenEndpoint,
                Properties.Settings.Default.ClientID,
                Properties.Settings.Default.ClientSecret))
            {
                tokenClient.BasicAuthenticationHeaderStyle =
                  BasicAuthenticationHeaderStyle.Rfc2617;
                var result = tokenClient
                .RequestResourceOwnerPasswordAsync(
                  Properties.Settings.Default.Username,
                  Properties.Settings.Default.Password,
                  Properties.Settings.Default.Scope
                ).Result;

                string accessToken = result.AccessToken;

                //Using the Default/18.200.001 endpoint
                using (DefaultSoapClient soapClient = new DefaultSoapClient())
                    try
                    {
                        soapClient.Endpoint.Behaviors.Add(
                          new AccessTokenAdderBehavior(accessToken));
                        //You will add the integration code here
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine(e);
                        Console.WriteLine();
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();
                    }
                    finally
                    {
                        //Sign out from MYOB Advanced
                        soapClient.Logout();
                    }
            }
        }

        //A supplementary class that adds the access token
        //to each request to the service
        private class AccessTokenAdderBehavior : IEndpointBehavior,
          IClientMessageInspector
        {
            private readonly string _accessToken;

            public AccessTokenAdderBehavior(string accessToken)
            {
                _accessToken = accessToken;
            }

            void IEndpointBehavior.Validate(ServiceEndpoint endpoint) { }

            void IEndpointBehavior.AddBindingParameters(
              ServiceEndpoint endpoint,
              BindingParameterCollection bindingParameters) { }

            void IEndpointBehavior.ApplyDispatchBehavior(
              ServiceEndpoint endpoint,
              EndpointDispatcher endpointDispatcher) { }

            void IEndpointBehavior.ApplyClientBehavior(
              ServiceEndpoint endpoint, ClientRuntime clientRuntime)
              => clientRuntime.ClientMessageInspectors.Add(this);

            object IClientMessageInspector.BeforeSendRequest(
              ref Message request, IClientChannel channel)
            {
                var httpRequestMessageProperty =
                  new HttpRequestMessageProperty();
                httpRequestMessageProperty.Headers.Add(
                  HttpRequestHeader.Authorization, "Bearer " + _accessToken);
                request.Properties.Remove(HttpRequestMessageProperty.Name);
                request.Properties.Add(
                  HttpRequestMessageProperty.Name,
                  httpRequestMessageProperty);
                return null;
            }

            void IClientMessageInspector.AfterReceiveReply(
              ref Message reply, object correlationState)
            {
            }
        }

        ...
    }
}
```

2 Rebuild the project.

Related Links

To Configure the Client Application

Logout() Method

Additional Information: Session Management in the OAuth 2.0 Applications

If you authorize your integration application to work with MYOB Advanced through OAuth 2.0, the sign-out is not always required after you have finished your work with MYOB Advanced.

Whether or not the sign-out is required depends on the access scope that the user has granted to the application as follows:

- If the user has granted only the `api` scope to the application, the sign-out is not required; however, we recommend that in this case you sign out after you have finished your work with MYOB Advanced. This scenario was described in the examples of this lesson.
- If the application has been granted the `api` and `offline_access` scopes (that is, the application has requested a refresh token along with an access token), when the access token has expired, the application can request a new access token by sending a request to the token endpoint and providing the refresh token. MYOB Advanced issues the first access token along with the session ID. If the client application requests a new access token by presenting a refresh token, MYOB Advanced reuses the session ID that was issued for the first access token issued with the refresh token. That is, the system uses a single session for each access granted to the client application. In this case, you do not need to sign out after you have finished your work with MYOB Advanced. This scenario is outside of the scope of this course.
- If the application has been granted the `api:concurrent_access` scope, MYOB Advanced can maintain multiple sessions for the application, managing session IDs through cookies. In this case, the application has to explicitly sign out from MYOB Advanced in each session to close the session. This scenario is outside of the scope of this course.

For details on the scopes that are available for each of the OAuth 2.0 flows, see the descriptions of the flows in the documentation (*Authorization Code Flow*, *Implicit Flow*, and *Resource Owner Password Credentials Flow*).

Lesson Summary

In this lesson, you have learned how to sign out from MYOB Advanced in an OAuth 2.0 application. You have also reviewed whether the sign-out is necessary for the OAuth 2.0 applications.

Part 2: Performance Optimization

In this part of the course, you will learn how to optimize the performance of an application that uses MYOB Advanced integration interfaces. You will perform the minimum number of requests that retrieve all necessary detail lines and find out how to deal with errors that may occur during the retrieval of the list of records through the contract-based APIs.

As a result of completing the lessons of this part, you will have experience in optimizing performance of an application that retrieves a list of sales orders from MYOB Advanced. You will also retrieve the list of payment records of a customer one by one, and will learn when you may need to use this approach.

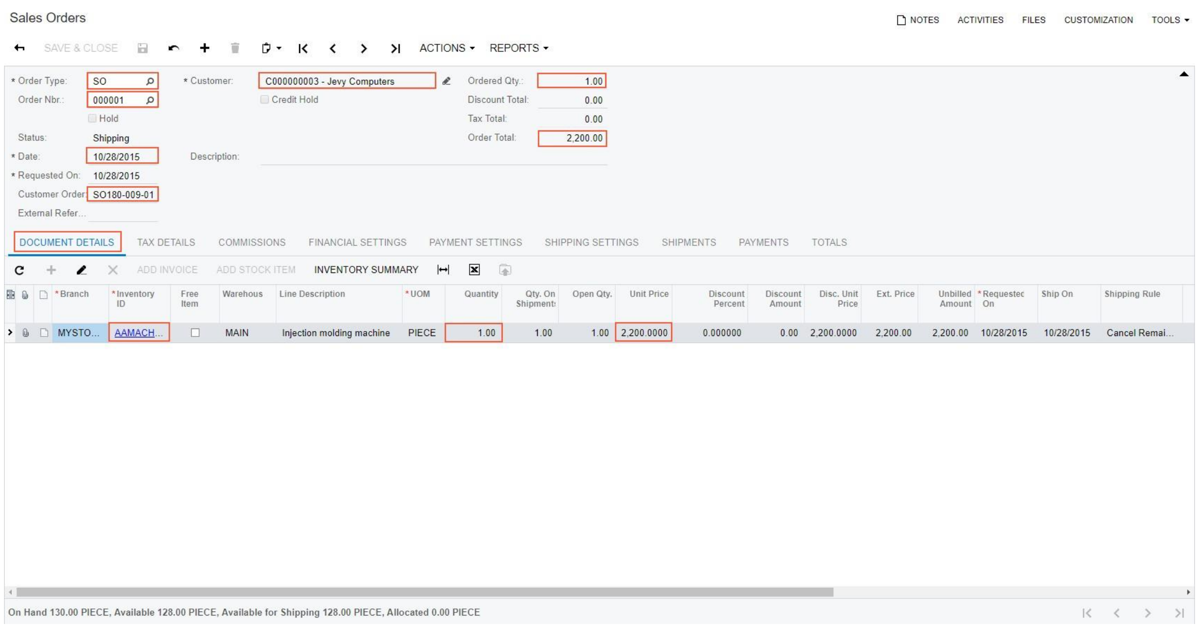
Lesson 2.1: Retrieving a List of Sales Orders with Details and Related Shipments

The online store of the MyStore company should display to a customer the list of sales orders of the customer with details and related shipments. The sales orders are created and maintained on the Sales Orders (SO301000) form in MYOB Advanced. That is, the MyStoreIntegration application should be able to export the list of sales orders with multiple kinds of details from MYOB Advanced.

In this lesson, you will retrieve the list of sales orders of the *C00000003* customer. For each sales order in the list, you need to retrieve the following values:

- The order type (**Order Type** of the Summary area of the Sales Orders form)
- The order number (**Order Nbr.** of the Summary area)
- The customer ID (**Customer ID** of the Summary area)
- The customer order number (**Customer Order** of the Summary area)
- The date when the sales order was created (**Date** of the Summary area)
- The ordered quantity (**Ordered Qty.** of the Summary area)
- The order total (**Order Total** of the Summary area)
- For each inventory item in the order (the **Document Details** tab):
    - The inventory ID (the **Inventory ID** column)
    - The quantity (the **Quantity** column)
    - The unit price (the **Unit Price** column)
- For each shipment in the order (the **Shipments** tab):
    - Shipment number (the **Document Nbr.** column)
    - Invoice number (the **Invoice Nbr.** column)

These elements are shown in the following screenshots.


Sales Orders
NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS
SAVE & CLOSE ACTIONS REPORTS

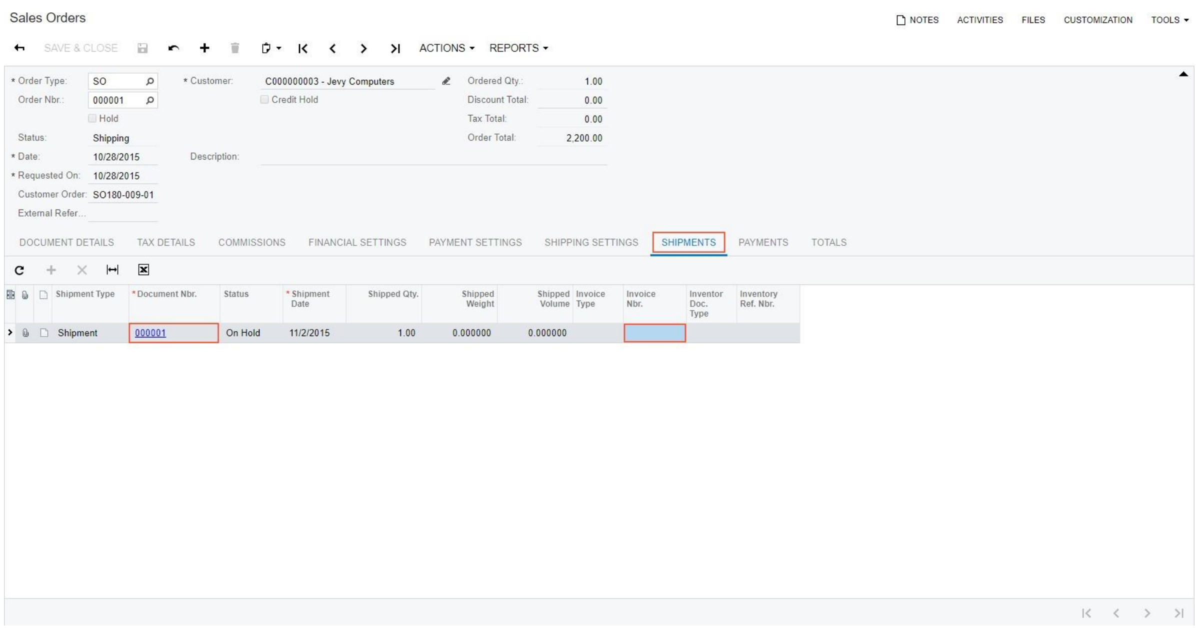
* Order Type:	SO	* Customer:	C000000003 - Jevy Computers	Ordered Qty.:	1.00
Order Nbr.:	000001		Credit Hold	Discount Total:	0.00
	Hold			Tax Total:	0.00
Status:	Shipping			Order Total:	2,200.00
* Date:	10/28/2015	Description:			
* Requested On:	10/28/2015				
Customer Order:	SO180-009-01				
External Refer...					

DOCUMENT DETAILS TAX DETAILS COMMISSIONS FINANCIAL SETTINGS PAYMENT SETTINGS SHIPPING SETTINGS SHIPMENTS PAYMENTS TOTALS
ADD INVOICE ADD STOCK ITEM INVENTORY SUMMARY

*Branch	*Inventory ID	Free Item	Warehous	Line Description	*UOM	Quantity	Qty. On Shipment:	Open Qty.	Unit Price	Discount Percent	Discount Amount	Disc. Unit Price	Ext. Price	Unbilled Amount	*Requestec On	Ship On	Shipping Rule
MYSTO...	AAMACH...	☐	MAIN	Injection molding machine	PIECE	1.00	1.00	1.00	2,200.0000	0.000000	0.00	2,200.0000	2,200.00	2,200.00	10/28/2015	10/28/2015	Cancel Remai...

On Hand 130.00 PIECE, Available 128.00 PIECE, Available for Shipping 128.00 PIECE, Allocated 0.00 PIECE

Figure: The Summary area and the Document Details tab



* Order Type:	SO	* Customer:	C000000003 - Jevy Computers	Ordered Qty.:	1.00
Order Nbr.:	000001		Credit Hold	Discount Total:	0.00
	Hold			Tax Total:	0.00
Status:	Shipping			Order Total:	2,200.00
* Date:	10/28/2015	Description:			
* Requested On:	10/28/2015				
Customer Order:	SO180-009-01				
External Refer...					

Shipment Type	* Document Nbr.	Status	* Shipment Date	Shipped Qty.	Shipped Weight	Shipped Volume	Invoice Type	Invoice Nbr.	Inventor Doc. Type	Inventory Ref. Nbr.
Shipment	000001	On Hold	11/2/2015	1.00	0.000000	0.000000				

Figure: The Shipments tab

You will use the `SalesOrder` entity of the *Default/18.200.001* endpoint to list the sales orders. The `SalesOrder` entity is mapped to the Sales Orders (SO301000) form. For the best performance of the application, it is important that you request only the values of the fields that you need (instead of requesting the values of all fields available in the entity).

To exclude the performance issues with the applications that work with MYOB Advanced through the contract-based APIs, multiple records cannot be retrieved from MYOB Advanced with multiple kinds of detail lines at a time. Therefore, you will perform the following requests, each of which retrieves one kind of detail lines:

- You will retrieve the list of sales orders with summary information and details about the inventory items included in the order (the data from the Summary area and the **Document Details** tab of the Sales Orders form).
- You will retrieve the list of sales orders with key fields and shipment details (the data from the **Shipments** tab of the Sales Orders form).

Instead of requesting each kind of detail lines separately (as described in this lesson), you could request records one by one with both kind of details at once. However, requesting records one by one significantly impairs the performance of the application. Therefore, we recommend that you use the approach described in this lesson.

This lesson shows how you can implement this scenario by using the contract-based REST API and contract-based SOAP API. You can also implement this scenario by using the OData interface and the screen-based SOAP API, but these implementations are outside of the scope of this course. For short information about these implementations, see *Additional Information: Retrieval of the List of Sales Orders Through Other Integration Interfaces*.

Lesson Objective

In this lesson, you will learn how to retrieve from MYOB Advanced records with multiple kinds of details.

Example 2.1.1: Using Two GET Requests (REST)

In this example, through the REST API, you will configure two HTTP requests that export the list of sales orders with multiple kinds of details.

To perform the requests, you will use the `GET` methods with the `$filter`, `$expand`, and `$select` parameters.

Retrieving the List of Sales Orders

To retrieve the list of sales orders with details and related shipments, do the following:

Because you have not requested the refresh token during the OAuth 2.0 authorization of the application, the access token, which you have received during authorization, expires in one hour and you need to request a new access token once the token has expired. In Postman, to receive a new access token, in the request that you configure in the collection, you can do the following:

1. Click the **Authorization** tab and click the link to the authorization settings of the collection.
2. In the **Edit Collection** dialog box, which opens, click **Get New Access Token**.
3. In the **Get New Access Token** dialog box, click **Request Token**.
4. In the **Manage Access Tokens** dialog box, click **Use Token**.
5. In the **Edit Collection** dialog box, click **Update**.

1. To retrieve the list of sales orders with summary information and document details, in Postman, configure the following settings of a contract-based REST API request:
   - HTTP method: `GET`
   - URL: *https://localhost/MyStoreInstance/entity/Default/18.200.001/SalesOrder*
   - Parameters of the request:

Parameter	Value
`$filter`	`CustomerID eq 'C000000003'`
`$expand`	`Details`
`$select`	`OrderNbr,OrderType,CustomerID,CustomerOrder,Details/InventoryID,Details/OrderQty,Details/UnitPrice,Date,OrderedQty,OrderTotal`

   - Headers:

Key	Value
`Accept`	`application/json`
`Content-Type`	`application/json`

2. Send the request. The response contains the `200 OK` status code. For each sales order of the *C00000003* customer, the body of the response includes the values of the `OrderType`, `OrderNbr`, `CustomerID`, `CustomerOrder`, `Date`, `OrderedQty`, and `OrderTotal` fields and the list of details with the values of `InventoryID`, `OrderQty`, and `UnitPrice`. The following code shows a fragment of the response body.

```
[
    {
        "id": "e6306119-9ce3-4903-b079-8cc22ab2b22b",
        "rowNumber": 1,
        "note": "",
        "CustomerID": {
            "value": "C000000003"
        },
        "CustomerOrder": {
            "value": "SO180-009-01"
        },
        "Date": {
            "value": "2015-10-28T00:00:00+03:00"
        },
        "Details": [
            {
                "id": "4e018049-a249-40cf-8b4f-d92e29d23c19",
                "rowNumber": 1,
                "note": "",
                "InventoryID": {
                    "value": "AAMACHINE1"
                },
                "OrderQty": {
                    "value": 1
                },
                "UnitPrice": {
                    "value": 2200
                },
                "custom": {},
                "files": []
            }
        ],
        "OrderedQty": {
            "value": 1
        },
        "OrderNbr": {
            "value": "000001"
        },
        "OrderTotal": {
            "value": 2200
        },
        "OrderType": {
            "value": "SO"
        },
        "custom": {},
        "files": []
    },
    ...
]
```

3. Save the request.
4. To retrieve the list of sales orders with shipment details, add a new request and configure the following settings of the request:
   - HTTP method: `GET`
   - URL: *https://localhost/MyStoreInstance/entity/Default/18.200.001/SalesOrder*
   - Parameters of the request:

Parameter	Value
`$filter`	`CustomerID eq 'C000000003'`
`$expand`	`Shipments`
`$select`	`OrderNbr,OrderType,Shipments/InvoiceNbr,Shipments/ShipmentNbr`

5. Send the request. The response contains the `200 OK`status code, and for each sales order of the *C00000003* customer, the body includes the values of the `OrderType`, `OrderNbr`, and

`CustomerID` fields and the list of shipments with the values of `InvoiceNbr` and `ShipmentNbr`. The following code shows a fragment of the response body.

```
[
    {
        "id": "e6306119-9ce3-4903-b079-8cc22ab2b22b",
        "rowNumber": 1,
        "note": "",
        "CustomerID": {
            "value": "C000000003"
        },
        "OrderNbr": {
            "value": "000001"
        },
        "OrderType": {
            "value": "SO"
        },
        "Shipments": [
            {
                "id": "7ece755e-6746-4068-9728-c06893fd7709",
                "rowNumber": 1,
                "note": "",
                "InvoiceNbr": {},
                "ShipmentNbr": {
                    "value": "000001"
                },
                "custom": {},
                "files": []
            }
        ],
        "custom": {},
        "files": []
    },
    ...
]
```

6. Save the request.

Related Links

Retrieval of Records by Conditions
Parameters for Retrieving Records

Example 2.1.2: Using Two GetList() Calls (SOAP)

In this example, you will add to the MyStoreIntegration application the method that retrieves the list of sales orders with document details and shipment details from MYOB Advanced.

You will use a `SalesOrder` class and the `GetList()` method of an instance of the `DefaultSoapClient` class available in the *Default* service reference. The `SalesOrder` class corresponds to the `SalesOrder` entity of the *Default/18.200.001* endpoint, which is mapped to the Sales Orders (SO301000) form.

You will use `ReturnBehavior.OnlySpecified` to make the system return only the values of the fields that are specified in the request.

Retrieving the List of Sales Orders

To export the list of sales orders with document details and related shipments, proceed as follows:

1. In the *MyStoreIntegration* project, add the `Integration` folder and the `PerformanceOptimization` C# class.
2. In the `PerformanceOptimization.cs` file, add the following `using` directives.

```
using MyStoreIntegration.Default;
using System.IO;
```

3. In the `PerformanceOptimization` class, add the `ExportSalesOrders()` method, as shown in the following code.

```
//Retrieving the list of sales orders of a customer
public static void ExportSalesOrders(DefaultSoapClient soapClient)
{
    Console.WriteLine("Getting the list of sales orders of a customer...");

    //Customer data
    string customerID = "C000000003";

    //Specify the customer ID of a customer whose sales orders should be
    //exported and the fields to be returned
    SalesOrder ordersToBeFound = new SalesOrder
    {
        //Return only the specified values
        ReturnBehavior = ReturnBehavior.OnlySpecified,

        //Specify the customer whose sales order should be returned
        CustomerID = new StringSearch { Value = customerID },

        //Specify the fields of the entity and details to be returned
        OrderType = new StringReturn(),
        OrderNbr = new StringReturn(),
        CustomerOrder = new StringReturn(),
        Date = new DateTimeReturn(),
        OrderedQty = new DecimalReturn(),
        OrderTotal = new DecimalReturn(),
        Details = new SalesOrderDetail[]
        {
            new SalesOrderDetail
            {
                InventoryID = new StringReturn(),
                OrderQty = new DecimalReturn(),
                UnitPrice = new DecimalReturn()
            }
        }
    };

    //Get the list of sales orders with details
    Entity[] soList = soapClient.GetList(ordersToBeFound);

    //Save the results to a CSV file
```

```
using (StreamWriter file = new StreamWriter(string.Format(
  @"SalesOrderDetails_Customer_{0}.csv", customerID)))
{
    //Add headers to the file
    file.WriteLine("OrderType;OrderNbr;CustomerID;CustomerOrder;Date;" +
      "OrderedQty;OrderTotal;InventoryID;OrderQty;UnitPrice;");

    //Write the values for each sales order
    foreach (SalesOrder salesOrder in soList)
    {
        foreach (SalesOrderDetail detail in salesOrder.Details)
        {
            file.WriteLine(string.Format(
                "{0};{1};{2};{3};{4};{5};{6};{7};{8};{9};",
                //Document summary
                salesOrder.OrderType.Value,
                salesOrder.OrderNbr.Value,
                salesOrder.CustomerID.Value,
                salesOrder.CustomerOrder.Value,
                salesOrder.Date.Value,
                salesOrder.OrderedQty.Value,
                salesOrder.OrderTotal.Value,
                detail.InventoryID.Value,
                detail.OpenQty.Value,
                detail.UnitCost.Value));
        }
    }
}

//Specify the fields in shipments to be returned
ordersToBeFound = new SalesOrder
{
    ReturnBehavior = ReturnBehavior.OnlySpecified,

    CustomerID = new StringSearch { Value = customerID },

    OrderType = new StringReturn(),
    OrderNbr = new StringReturn(),
    Shipments = new SalesOrderShipment[]
    {
        new SalesOrderShipment
        {
            ShipmentNbr = new StringReturn(),
            InvoiceNbr = new StringReturn()
        }
    }
};

//Get the list of sales orders of the customer
soList = soapClient.GetList(ordersToBeFound);

//Save results to a CSV file
using (StreamWriter file = new StreamWriter(string.Format(
  @"SalesOrderShipments_Customer_{0}.csv", customerID)))
{
    //Add headers to the file
    file.WriteLine("OrderType;OrderNbr;ShipmentNbr;InvoiceNbr;");

    //Write the values for each sales order
    foreach (SalesOrder salesOrder in soList)
    {
        foreach (SalesOrderShipment shipment in salesOrder.Shipments)
        {
            file.WriteLine(string.Format("{0};{1};{2};{3};",
                //Document summary
                salesOrder.OrderType.Value,
                salesOrder.OrderNbr.Value,
                shipment.ShipmentNbr.Value,
                shipment.InvoiceNbr.Value));
        }
```

```
            }
        }
    }
```

4. In the `Program.cs` file, add the `MyStoreIntegration.Integration` `using` directive.
5. In the `try` block of the `Main()` method of the `Program` class, add the code that calls the `ExportSalesOrders()` method, as shown below.

   ```
   PerformanceOptimization.ExportSalesOrders(soapClient);
   ```

6. Rebuild the program, and run the application. If you run the project in Debug mode, you can find two files (`SalesOrderDetails_Customer_C000000003.csv` and `SalesOrderShipments_Customer_C000000003.csv`) in the `\bin\Debug` folder of the MyStoreIntegration project.

Related Links

GetList() Method (Contract Version 3)
ReturnBehavior Property (Contract Version 3)

Additional Information: Retrieval of the List of Sales Orders Through Other Integration Interfaces

You can implement the integration scenario described in this lesson by using the OData interface or the screen-based SOAP API.

Implementation with the OData Interface

To export records with multiple kinds of detail lines from MYOB Advanced by using the OData interface, you need to configure multiple generic inquiries on the Generic Inquiry (SM208000) form that export all necessary data (one generic inquiry for each kind of detail lines that you need to export), expose these generic inquiries via OData (by clicking the **Expose via OData** check box on the form), and execute the OData requests. For the best performance of your application, we recommend that you create a separate generic inquiry for each kind of detail lines.

For details about the execution of OData requests, see *OData Support* in the documentation or the *I300 Web Services: Basic | Data Retrieval* training course.

Implementation with the Screen-Based SOAP API

To retrieve the records with multiple kinds of detail lines from MYOB Advanced by using the screen-based SOAP API, you export the records from an MYOB Advanced form by using the `Export()` method that corresponds to this form. In the first parameter, you specify the needed fields as an array of `Command` objects. This method requests the records one by one; therefore, it has slow performance. You can compose multiple generic inquiries on the Generic Inquiry (SM208000) form that retrieve the necessary data (one generic inquiry for each kind of detail lines that you need to export) and use them for the data retrieval to optimize the performance of your application.

For details about the data retrieval with the screen-based API, see *Export() Method* and *Commands for Retrieving the Values of Elements* in the documentation, or see the *I300 Web Services: Basic | Data Retrieval* training course.

Lesson Summary

In this lesson, you have added to the MyStoreIntegration REST and SOAP applications the requests that retrieve the list of sales orders with details and related shipments from MYOB Advanced. You have used one request for each kind of detail that you need to retrieve, which optimizes the performance of the application. You have also reviewed how this scenario can be implemented with the OData interface and the screen-based SOAP API.

The following table summarizes the availability of the performance optimization options for different integration interfaces.

Integration Interface	**Optimized Retrieval of Multiple Kinds of Detail Lines**
OData interface	Yes, if you create multiple custom generic inquiries (one inquiry for each kind of detail line)
REST API	Yes (you need to perform multiple requests: one request for each kind of detail line)
Contract-based SOAP API	Yes (you need to perform multiple requests: one request for each kind of detail line)
Screen-based SOAP API	Yes, if you create multiple custom generic inquiries (one inquiry for each kind of detail line)

Lesson 2.2: Retrieving the List of Payments One by One

The online store of the MyStore company should display to a customer the list of payments of the customer. The payments are created on the Payments and Applications (AR302000) form in MYOB Advanced. That is, the MyStoreIntegration application should be able to export the list of payments from MYOB Advanced.

This lesson shows how to export payments with details if it is impossible to request the list of payments with details in one request.

In this lesson, you will retrieve the list of payments of the *C00000003* customer. For each payment in the list, you need to retrieve the following values (with the corresponding locations on the Payments and Applications form):

- The reference number (the **Reference Nbr.** box in the Summary area)
- The type (the **Type** box in the Summary area)
- The status (the **Status** box in the Summary area)
- The application date (the **Application Date** box in the Summary area)
- The type and reference number of the document to which the payment is applied (the **Doc. Type** and **Reference Nbr.** columns on the **Application History** tab)

These elements correspond to the following fields of the `Payment` entity of the *Default/18.200.001* endpoint:

- `ReferenceNbr`
- `Type`
- `Status`
- `ApplicationDate`
- `DisplayDocType` and `DisplayRefNbr` of the `ApplicationHistory` detail entity

When multiple records are retrieved from MYOB Advanced through an endpoint with Contract Version 3 (which is the *Default/18.200.001* endpoint has), the system tries to optimize the retrieval of the records and obtain all needed records in one request to the database (instead of requesting the records one by one). However, in the examples of this lesson, the optimization fails for the `ApplicationDate` field and the fields of the `ApplicationHistory` detail entity, and the system returns an error. To fix this error, you have the following options:

- If you do not need to retrieve the `ApplicationDate` field and the fields of the `ApplicationHistory` detail entity, you can exclude these fields and the `ApplicationHistory` entity from the request.
- If you need to retrieve the `ApplicationDate` field and the fields of the `ApplicationHistory` detail entity, you can retrieve the needed records one by one by the key fields.

In this lesson, you will use the latter option. That is, you will do the following:

1. Retrieve the list of key fields of the payments of the *C00000003* customer
2. Retrieve the payments with the needed details one by one by using the key fields

To achieve the best performance of the retrieval of the list of payments, you can create a custom generic inquiry, add it to a custom endpoint or an endpoint extension, and use this generic inquiry for the data retrieval. This scenario is outside of the scope of this course. For details about retrieval of the data from a generic inquiry, see *Retrieval of Data from an Inquiry Form* (for the REST API) and *Put() Method* (for the SOAP API) in the documentation, or see the *I300 Web Services: Basic* | *Data Retrieval* training course.

Lesson Objective

In this lesson, you will learn how to deal with the errors that can occur if the performance optimization fails.

Example 2.2.1: Using GET with Key Field Values (REST)

In this example, you will do the following:

1. Try to retrieve the list of payments with `ApplicationHistory` details in one request, which fails.
2. Modify the request to retrieve only the key fields of the needed payments.
3. Configure the request that retrieves the payments one by one by the key fields. To retrieve the record by the key fields, you will specify the values of the key fields in the URL of the request.

If you specify the key field values in the `$filter` parameter instead of passing the key fields in the URL of the request, MYOB Advanced treats this request as a request for multiple records and performs additional optimizations, which are not necessary when you request one record. Therefore, we recommend that you specify the key field values in the URL of the request if you want to retrieve one record.

Retrieving the List of Payments

To retrieve the list of payments, do the following:

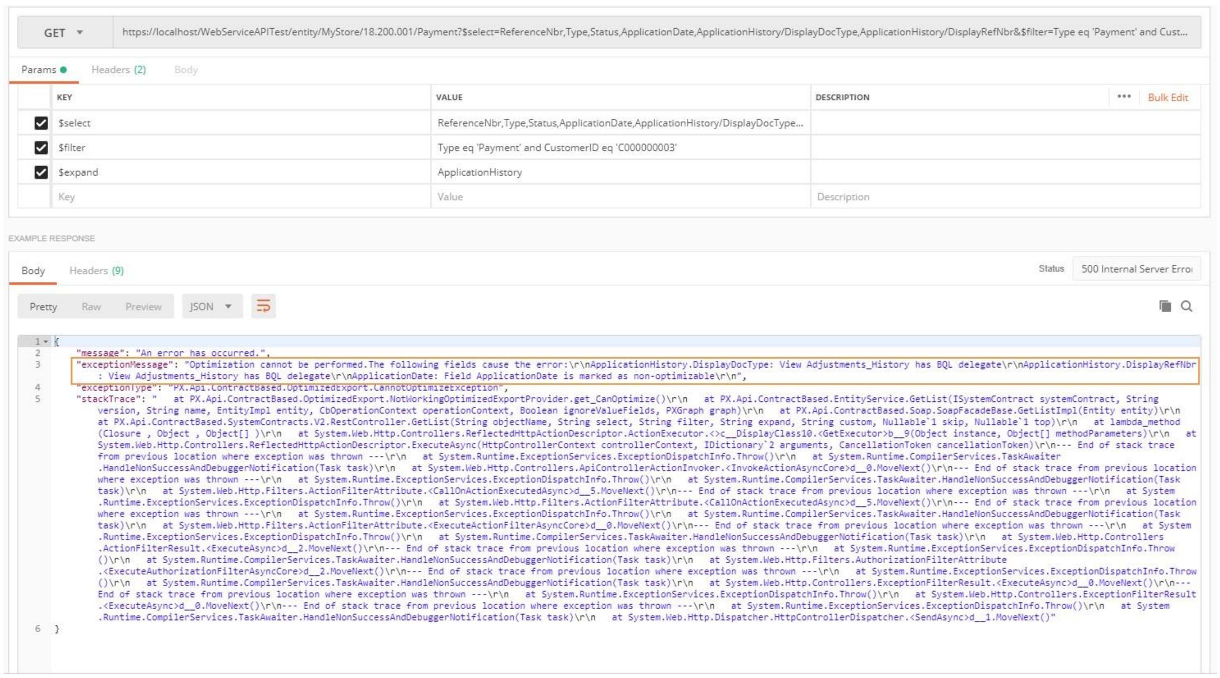
1. In Postman, configure the following settings of the contract-based REST API request:
    - HTTP method: `GET`
    - URL: *https://localhost/MyStoreInstance/entity/Default/18.200.001/Payment*
    - Parameters of the request:

Parameter	Value
`$filter`	`Type eq 'Payment' and CustomerID eq 'C000000003'`
`$expand`	`ApplicationHistory`
`$select`	`ReferenceNbr,Type,Status,ApplicationHistory/DisplayDocType,ApplicationHistory/DisplayRefNbr,ApplicationDate`

    - Headers:

Key	Value
`Accept`	`application/json`
`Content-Type`	`application/json`

2. Send the request. The request fails with the error that is shown in the following screenshot.



https://localhost/WebServiceAPITest/entity/MyStore/18.200.001/Payment?$select=ReferenceNbr,Type,Status,ApplicationDate,ApplicationHistory/DisplayDocType,ApplicationHistory/DisplayRefNbr&$filter=Type eq 'Payment' and Cust...

	KEY	VALUE	DESCRIPTION
✓	$select	ReferenceNbr,Type,Status,ApplicationDate,ApplicationHistory/DisplayDocType...	
✓	$filter	Type eq 'Payment' and CustomerID eq 'C000000003'	
✓	$expand	ApplicationHistory	
	Key	Value	Description

Status 500 Internal Server Error

```
{
    "message": "An error has occurred.",
    "exceptionMessage": "Optimization cannot be performed.The following fields cause the error:\r\nApplicationHistory.DisplayDocType: View Adjustments_History has BQL delegate\r\nApplicationHistory.DisplayRefNbr
        : View Adjustments_History has BQL delegate\r\nApplicationDate: Field ApplicationDate is marked as non-optimizable\r\n",
    "exceptionType": "PX.Api.ContractBased.OptimizedExport.CannotOptimizeException",
    "stackTrace": "   at PX.Api.ContractBased.OptimizedExport.NotWorkingOptimizedExportProvider.get_CanOptimize()\r\n   at PX.Api.ContractBased.EntityService.GetList(ISystemContract systemContract, String
        version, String name, EntityImpl entity, CbOperationContext operationContext, Boolean ignoreValueFields, PXGraph graph)\r\n   at PX.Api.ContractBased.Soap.SoapFacadeBase.GetListImpl(Entity entity)\r\n
        at PX.Api.ContractBased.SystemContracts.V2.RestController.GetList(String objectName, String select, String filter, String expand, String custom, Nullable`1 skip, Nullable`1 top)\r\n   at lambda_method
        (Closure , Object , Object[] )\r\n   at System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.<>c__DisplayClass10.<GetExecutor>b__9(Object instance, Object[] methodParameters)\r\n   at
        System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ExecuteAsync(HttpControllerContext controllerContext, IDictionary`2 arguments, CancellationToken cancellationToken)\r\n--- End of stack trace
        from previous location where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Runtime.CompilerServices.TaskAwaiter
        .HandleNonSuccessAndDebuggerNotification(Task task)\r\n   at System.Web.Http.Controllers.ApiControllerActionInvoker.<InvokeActionAsyncCore>d__0.MoveNext()\r\n--- End of stack trace from previous location
        where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
        task)\r\n   at System.Web.Http.Filters.ActionFilterAttribute.<CallOnActionExecutedAsync>d__5.MoveNext()\r\n--- End of stack trace from previous location where exception was thrown ---\r\n   at System
        .Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Web.Http.Filters.ActionFilterAttribute.<CallOnActionExecutedAsync>d__5.MoveNext()\r\n--- End of stack trace from previous location
        where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
        task)\r\n   at System.Web.Http.Filters.ActionFilterAttribute.<ExecuteActionFilterAsyncCore>d__0.MoveNext()\r\n--- End of stack trace from previous location where exception was thrown ---\r\n   at System
        .Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)\r\n   at System.Web.Http.Controllers
        .ActionFilterResult.<ExecuteAsync>d__2.MoveNext()\r\n--- End of stack trace from previous location where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw
        ()\r\n   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)\r\n   at System.Web.Http.Filters.AuthorizationFilterAttribute
        .<ExecuteAuthorizationFilterAsyncCore>d__2.MoveNext()\r\n--- End of stack trace from previous location where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw
        ()\r\n   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)\r\n   at System.Web.Http.Controllers.ExceptionFilterResult.<ExecuteAsync>d__0.MoveNext()\r\n---
        End of stack trace from previous location where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System.Web.Http.Controllers.ExceptionFilterResult
        .<ExecuteAsync>d__0.MoveNext()\r\n--- End of stack trace from previous location where exception was thrown ---\r\n   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()\r\n   at System
        .Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)\r\n   at System.Web.Http.Dispatcher.HttpControllerDispatcher.<SendAsync>d__1.MoveNext()"
}
```

Figure: The returned error

3. Modify the parameters of the request to return only the key fields of the payments as follows:

Parameter	Value
`$filter`	`Type eq 'Payment' and CustomerID eq 'C000000003'`
`$select`	`ReferenceNbr,Type`

4. Send the request. The response contains the `200 OK` status code, and the body includes the values of the `CustomerID`, `ReferenceNbr`, and `Type` fields.
5. Save the request.
6. Configure the request that retrieves the details of the payments by the values of the key fields (`ReferenceNbr` and `Type`) as follows:
   - HTTP method: `GET`
   - URL: *https://localhost/MyStoreInstance/entity/Default/18.200.001/Payment/Payment/000001*
   - Parameters of the request:

Parameter	Value
`$select`	`ReferenceNbr,Type,Status,ApplicationHistory/DisplayDocType,ApplicationHistory/DisplayRefNbr,ApplicationDate`
`$expand`	`ApplicationHistory`

   - Headers:

Key	Value
`Accept`	`application/json`
`Content-Type`	`application/json`

7. Send the request. Make sure the HTTP status code is `200 OK`, and review the returned data, of which an example is shown below.

```
{
    "id": "525ec24d-e03b-4628-a430-712163515bc2",
    "rowNumber": 1,
    "note": "",
    "ApplicationDate": {
        "value": "2018-10-12T00:00:00+03:00"
    },
    "ApplicationHistory": [
        {
            "id": "ad31f8d4-04a0-45b8-9cf5-efd1b20cb664",
            "rowNumber": 1,
            "note": "",
            "DisplayDocType": {
                "value": "Invoice"
            },
            "DisplayRefNbr": {
                "value": "INV000045"
            },
            "custom": {},
            "files": []
        }
    ],
    "ReferenceNbr": {
        "value": "000001"
    },
    "Status": {
        "value": "Closed"
    },
    "Type": {
        "value": "Payment"
    },
    "custom": {},
    "files": []
}
```

8. Save the request.

9. Repeat the request for each pair of payment key fields in the list.

If you performed only the examples described in this guide, the *C00000003* customer has only one payment; therefore, you do not need to perform other requests to retrieve the details of all payments of the customer.

Related Links

Retrieval of Records by Conditions
Retrieval of a Record by Key Fields

Example 2.2.2: Using the Get() Method and the Key Field Values (SOAP)

In this example, you will export the list of payments of the *C000000003* customer one by one.

Retrieving the List of Payments

To retrieve the list of payments, do the following:

1. In the `PerformanceOptimization` class, add the following code:

```
//Retrieving the list of payments of a customer
public static void ExportPayments(DefaultSoapClient soapClient)
{
    Console.WriteLine("Retrieving the list of payments of a customer...");

    //Input data
    string customerID = "C000000003";
    string docType = "Payment";

    //Select the payments that should be exported
    Payment soPaymentsToBeFound = new Payment
    {
        ReturnBehavior = ReturnBehavior.OnlySpecified,

        Type = new StringSearch { Value = docType },
        CustomerID = new StringSearch { Value = customerID },

        ReferenceNbr = new StringReturn()
    };
    Entity[] payments = soapClient.GetList(soPaymentsToBeFound);

    //Retrieve the payments one by one
    foreach (Payment payment in payments)
    {
        Payment soPaymentToBeRetrieved = new Payment
        {
            ReturnBehavior = ReturnBehavior.OnlySpecified,

            Type = new StringSearch
            {
                Value = payment.Type.Value,
            },
            ReferenceNbr = new StringSearch
            {
                Value = payment.ReferenceNbr.Value,
            },

            ApplicationDate = new DateTimeReturn(),
            Status = new StringReturn(),
            ApplicationHistory = new[]
            {
                new PaymentApplicationHistoryDetail
                {
                    DisplayDocType = new StringReturn(),
                    DisplayRefNbr = new StringReturn()
                }
            }
        };
        Payment result = (Payment)soapClient.Get(soPaymentToBeRetrieved);

        //Save the results to a CSV file
        using (StreamWriter file = new StreamWriter(
            string.Format(@"Payment_{0}.csv", payment.ReferenceNbr.Value)))
        {
            file.Write(string.Format("{0};{1};{2};{3};",
                    //Document summary
                    result.Type.Value,
                    result.ReferenceNbr.Value,
                    result.ApplicationDate.Value,
                    result.Status.Value));
            foreach (PaymentApplicationHistoryDetail detail
              in result.ApplicationHistory)
            {
                file.Write(string.Format("{0};{1};",

                    //Application details
                    detail.DisplayDocType.Value,
                    detail.DisplayRefNbr.Value));
            }
        }
    }
}
```

2. In the `try` block of the `Main()` method of the `Program` class, add the code that calls the `ExportPayments()` method, as shown below.

   You can comment the code of the previous example in the `Program.Main()` method.

```
PerformanceOptimization.ExportPayments(soapClient);
```

3. Rebuild the program and run the application. If you run the project in Debug mode, you can find the resulting files in the `\bin\Debug` folder of the *MyStoreIntegration* project.

Related Links

Get() Method

ReturnBehavior Property (Contract Version 3)

Lesson Summary

In this lesson, you have added to the MyStoreIntegration REST and SOAP applications the methods that retrieve the payments of a customer one by one. You have used this approach because the retrieval of the payments with the specified fields could not be optimized for performance.

Part 3: Retrieval of Attachments

In this part of the guide, you will use the web services API to retrieve the attachments of the stock item records from MYOB Advanced. You will retrieve the attachments by using the contract-based REST and SOAP APIs.

Attachments cannot be retrieved from MYOB Advanced by using the OData requests. You can obtain the attachments of a record in MYOB Advanced by using the screen-based SOAP API. However, if you do not know the name of the file that you need to obtain, the retrieval of the attachment requires preparation of a special generic inquiry. For details, see *Additional Information: Retrieval of the Attachments with the Screen-Based SOAP API*.

Lesson 3.1: Retrieving the Attachments of a Stock Item

In this lesson, you will add to the MyStoreIntegration application a method that retrieves the files that are attached to a stock item record.

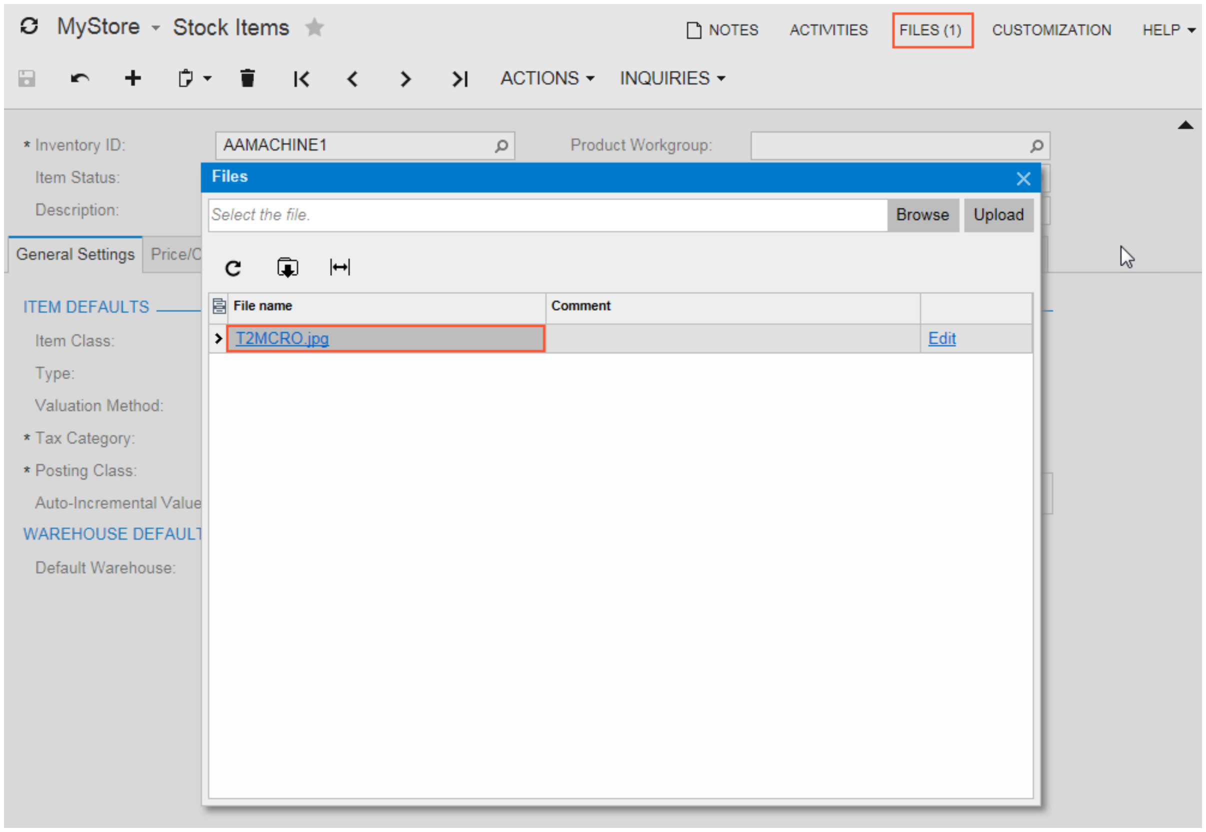
The MyStore company needs to display an image of each item that is sold in the online store. Images for the items can be stored in MYOB Advanced as attachments to the Stock Items (IN202500) form. To display an image of a stock item in the online store, the MyStoreIntegration application should export the images that are attached to the stock item. The name of the image is not known when the online store requests the file from MYOB Advanced.

Lesson Objective

In this lesson, you will learn how to retrieve the files that are attached to a stock item by using the contract-based APIs.

Prerequisites

On the Stock Items (IN202500) form, select the record with the *AAMACHINE1* inventory ID. On the title toolbar, click **Files** and notice that one file with the name `T2MCRO.jpg` is attached to the record, as shown in the following screenshot. In the examples of this lesson, you will export this file from MYOB Advanced.



* Inventory ID:	AAMACHINE1
Product Workgroup:	
Item Status:	
Description:	
Item Class:	
Type:	
Valuation Method:	
* Tax Category:	
* Posting Class:	
Auto-Incremental Value	
Default Warehouse:	

Select the file.

File name	Comment	
T2MCRO.jpg		Edit

Figure: Attached file

Example 3.1.1: Using the GET Method (REST)

In this example, you will configure an HTTP request that exports the files attached to a stock item record by using the contract-based REST API.

You will obtain the needed stock item record by using the values of its key fields. You will specify the key fields in the URL of the `GET` request. You will retrieve only the ID of the stock item and the values of the system fields, which are always returned. Then you will retrieve the attached file by using the `GET` method and the link that will have been returned in the `href` field of an entity in the `files` array.

Retrieving the Attachments of a Stock Item

To retrieve the attachments of a stock item, do the following:

1. In Postman, configure the following settings of the request that retrieves the stock item with the *AAMACHINE1* ID:
    - HTTP method: `GET`
    - URL: *https://localhost/MyStoreInstance/entity/Default/18.200.001/StockItem/AAMACHINE1*
    - Parameters: `$select=InventoryID`
    - Headers:

Key	**Value**
`Accept`	`application/json`
`Content-Type`	`application/json`

2. Send the request. The response of the successful request contains the `200 OK` status code and includes in the body the links to the files attached to the stock item record, as the following code example shows.

```
{
    "id": "2537440a-52c5-4c79-9382-5a78ec581f1e",
    "rowNumber": 1,
    "note": "",
    "InventoryID": {
        "value": "AAMACHINE1"
    },
    "custom": {},
    "files": [
        {
            "id": "9be45eb7-f97d-400b-96a5-1c4cf82faa96",
            "filename": "Stock Items (AAMACHINE1)\\T2MCRO.jpg",
            "href": "/MyStoreInstance/entity/Default/18.200.001/
files/9be45eb7-f97d-400b-96a5-1c4cf82faa96"
        }
    ]
}
```

3. Send the `GET` request to the URL returned for the attached file (*https://localhost/MyStoreInstance/entity/Default/18.200.001/files/9be45eb7-f97d-400b-96a5-1c4cf82faa96*). The response of a successful request contains the file in the response body.

Related Links

Retrieval of a Record by Key Fields
Retrieval of a File Attached to a Record

Example 3.1.2: Using the GetFiles() Method (SOAP)

In this example, you will add to the MyStoreIntegration application a method that exports all files attached to a stock item on the Stock Items (IN202500) form by using the contract-based SOAP API.

You will use the `Get()` method of a `DefaultSoapClient` object to retrieve a stock item record with the specified inventory ID. You will use `ReturnBehavior.OnlySpecified` to return only the specified fields and the `ImageUrl` field to be returned. Then you will pass the retrieved `StockItem` entity to the `GetFiles()` method of the `DefaultSoapClient` object to obtain the attached files.

Retrieving the Files That Are Attached to a Stock Item

To export the files that are attached to a stock item, do the following:

1. In the `Integration`folder of the *MyStoreIntegration* project, add a new C# class with the name `Attachments`.

   Visual Studio creates a `Attachments.cs` file in the *MyStoreIntegration* project.

2. In the `Attachments.cs` file, type the `using` directives as shown in the following code to make the `Attachments` class use the *Default* service reference and the `System.IO` classes.

```
using MyStoreIntegration.Default;
using System.IO;
```

3. In the `Attachments` class of the *MyStoreIntegration* project, add the `ExportStockItemFiles()` method, which is shown in the following code.

```
//Retrieving the files that are attached to a stock item
public static void ExportStockItemFiles(DefaultSoapClient soapClient)
{
    Console.WriteLine(
        "Retrieving the files that are attached to a stock item...");

    //Parameters of filtering
    string inventoryID = "AAMACHINE1";

    //Filter the items by inventory ID
    StockItem stockItemToBeFound = new StockItem
    {
        InventoryID = new StringSearch { Value = inventoryID },
        ImageUrl = new StringReturn(),
        ReturnBehavior = ReturnBehavior.OnlySpecified
    };

    //Get the stock item record
    StockItem stockItem = (StockItem)soapClient.Get(stockItemToBeFound);

    //Get the files that are attached to the stock item and
    //save them on a local disc
    if (stockItem != null && stockItem.ImageUrl != null)
    {
        //Get the attached files
        Default.File[] files = soapClient.GetFiles(stockItem);

        //Save the files on disc
        foreach (Default.File file in files)
        {
            //The file name obtained from MYOB Advanced has the following
            //format: Stock Items (<Inventory ID>)\<File Name>
            string fileName = Path.GetFileName(file.Name);
            System.IO.File.WriteAllBytes(fileName, file.Content);
        }
    }
}
```

4. In the `try` block of the `Main()` method of the `Program` class, call the `ExportStockItemFiles()` method of the `Attachments` class, as the following code shows.

   ```
   Attachments.ExportStockItemFiles(soapClient);
   ```

5. Rebuild the project, and run the application. If you run the project in Debug mode, you can find the exported `T2MCRO.jpg` file in the `\bin\Debug` folder of the *MyStoreIntegration* project.

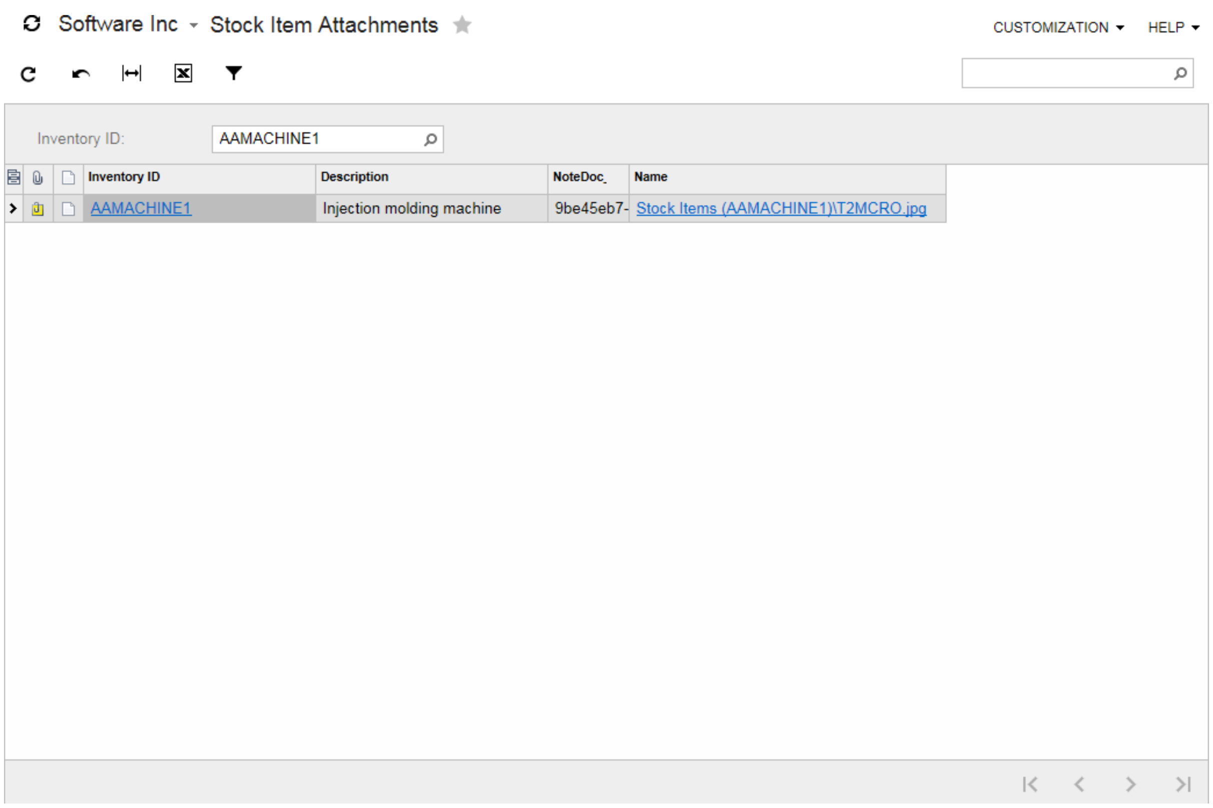
Related Links

Get() Method
GetFiles() Method

Additional Information: Retrieval of the Attachments with the Screen-Based SOAP API

The scenario described in this topic is outside of the scope of this course but may be useful to some readers.

To retrieve the file attached to a record, you use the `Attachment` service command of the object that corresponds to the Summary object of the form. To use this service command, you need to know the name of the attached file, but this is generally not the case. For the retrieval of the file name, you need to create a generic inquiry that retrieves the file names for the records. For example, the Stock Item Attachments (INGI0005) generic inquiry, which has been preconfigured in the instance that you are using for this training course, obtains the names of the files that are attached to a stock item record. You can view the generic inquiry on the Generic Inquiry (SM208000) form by selecting the inquiry with the title *Stock Item Attachments* and clicking **View Inquiry**. The Stock Item Attachments generic inquiry is shown in the following screenshot.



Inventory ID: AAMACHINE1

Inventory ID	Description	NoteDoc_	Name
AAMACHINE1	Injection molding machine	9be45eb7-	Stock Items (AAMACHINE1)\T2MCRO.jpg

Figure: Stock Item Attachments generic inquiry

That is, to retrieve the files that are attached to a record, you do the following:

1. Retrieve the names of the files by using the `Export()` or `Submit()` method of the `Content` object that corresponds to the generic inquiry.
2. Compose a command with the retrieved name and the `Attachment` service command as the linked command, and pass this command to the `Export()` method of the `Content` object that corresponds to the form from which you need to retrieve the attachments.

For details about the retrieval of the attachments, see *Commands for Working with Attachments*.

Lesson Summary

In this lesson, you have learned how to export the files that are attached to a record.

The following table summarizes whether and how the attachment can be retrieved from MYOB Advanced through the use of the integration interfaces.

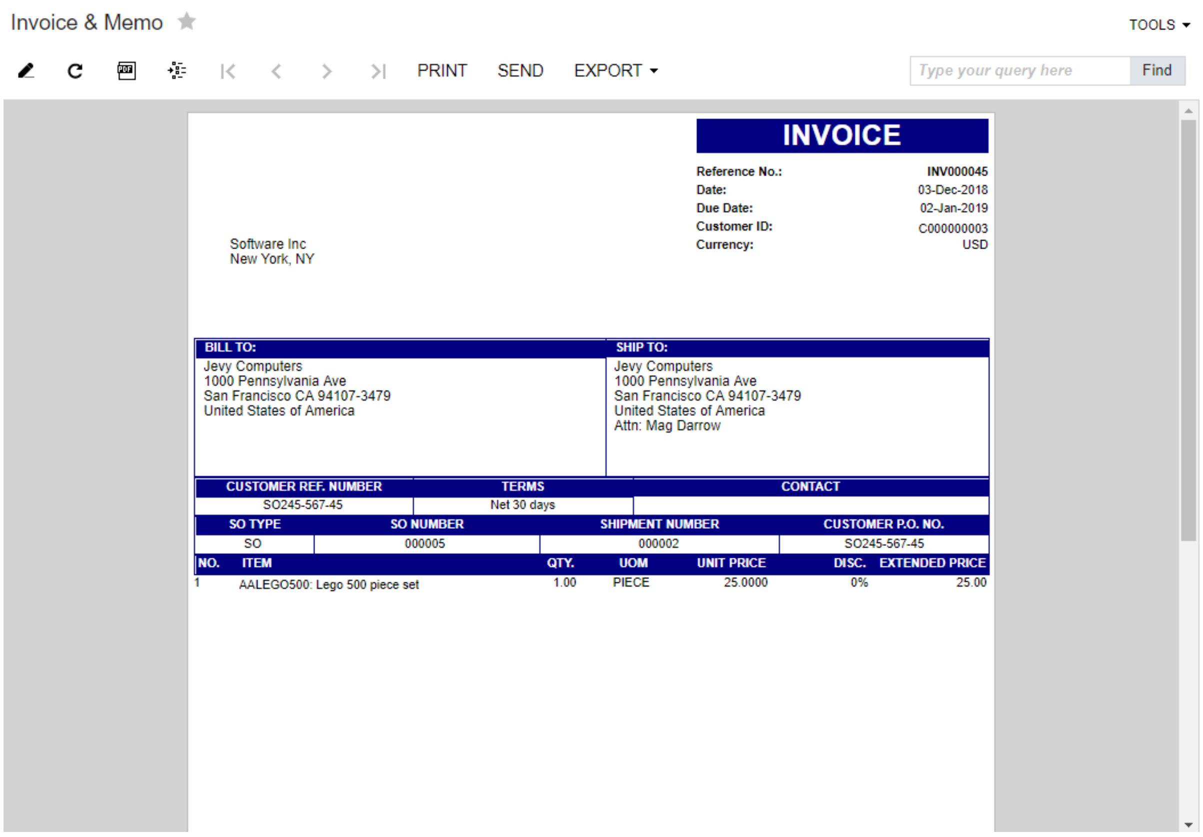
Integration Interface	**Retrieval of Attachments**
OData interface	No
REST API	Yes
Contract-based SOAP API	Yes
Screen-based SOAP API	Yes (if the names of the attached files are known)

Part 4: Retrieval of Reports

In this part of the guide, you will use the web services API to retrieve a printable report from MYOB Advanced. The only web services API that you can use to obtain a printable document is the screen-based SOAP API. OData and the contract-based APIs do not provide the functionality for the retrieval of printable documents.

Lesson 4.1: Generating the Printable Version of an Invoice

The MyStore company needs to provide to a customer a printable version of an invoice. You can view the printable version of the invoice on the Invoice & Memo (S0643000) report, which is shown in the following screenshot.



INVOICE

Reference No.:	INV000045
Date:	03-Dec-2018
Due Date:	02-Jan-2019
Customer ID:	C000000003
Currency:	USD

Software Inc
New York, NY

BILL TO:	SHIP TO:
Jevy Computers 1000 Pennsylvania Ave San Francisco CA 94107-3479 United States of America	Jevy Computers 1000 Pennsylvania Ave San Francisco CA 94107-3479 United States of America Attn: Mag Darrow

CUSTOMER REF. NUMBER	TERMS	CONTACT
SO245-567-45	Net 30 days	

SO TYPE	SO NUMBER	SHIPMENT NUMBER	CUSTOMER P.O. NO.
SO	000005	000002	SO245-567-45

NO.	ITEM	QTY.	UOM	UNIT PRICE	DISC.	EXTENDED PRICE
1	AALEGO500: Lego 500 piece set	1.00	PIECE	25.0000	0%	25.00

Figure: Invoice & Memo form

This lesson shows how you can implement this integration scenario by using the screen-based SOAP API.

Lesson Objective

In this lesson, you will learn how to retrieve a PDF report from MYOB Advanced by using the screen-based SOAP API.

Example 4.1.1: Using the Screen-Based SOAP API

In this example, you will add to the MyStoreIntegration application a method that prints a PDF version of an invoice that exists in MYOB Advanced.

You will add to the project the reference to the web service that is provided by the Invoice & Memo (SO643000) report form. You will use the screen-based API wrapper to prevent application failures due to possible UI changes on the Invoice & Memo report form in later versions of MYOB Advanced.

To get the printable version of an invoice, you will use the `PdfContent` command of the `ReportResults` subobject of the `Content` object that corresponds to the Invoice & Memo (SO643000) report form.

Prerequisites

In this example, you will get a PDF version of the invoice with reference number *INV000045*. On the Invoices (SO303000) form, verify that an invoice with this reference number exists and that it has the *Closed* status.

Instructions for Generating the Printable Version of an Invoice

To get a PDF version of the invoice by using the web services API, do the following:

1. In Visual Studio, create a Visual C# console application with the *MyStoreIntegrationSBAPI* name and the *.NET Framework 4.7.1* target framework.

   Instead of creating a Visual Studio project, you can use the solution provided with this course (`MyStoreIntegration\SBAPI\MyStoreIntegrationSBAPI.sln`). This solution is configured for this course and already contains the method that is described in this example. You can use this solution for testing.

2. In the project, add the reference to the screen-based web service provided by the Invoice & Memo (SO643000) report form. Use the `SO643000` web reference name.

   To get the link to the service, on the Invoice & Memo report form, click **Tools** > **Web Service** and copy the URL from the address line in the browser for the page that opens. For details about how to add the web service reference to the project, see *To Import the WSDL File Into the Development Environment* in the documentation.

3. Add the `Integration` folder to the *MyStoreIntegrationSBAPI* project, and add a new C# class to it with the name `Reports`.

4. To use the screen-based API wrapper, add a reference to the `PX.Soap.dll` library to the *MyStoreIntegrationSBAPI* project. `PX.Soap.dll` is located in the `ScreenBasedAPIWrapper` folder in the MYOB Advanced installation folder. (By default, it is `C:\Program Files\MYOB Advanced \ScreenBasedAPIWrapper`).

5. In the `Reports` class, add the `MyStoreIntegrationSBAPI.SO643000`, `PX.Soap`, and `System.IO` `using` directives.

6. In the `Reports` class, define the `GetPrintableInvoice()` method as follows.

```
//Getting the printable version of an invoice
//on the Invoice & Memo (SO643000) report form
public static void GetPrintableInvoice(Screen context)
{
    //Invoice data
    string docType = "Invoice";
    string invoiceNbr = "INV000045";

    Console.WriteLine("Generating the printable version of an invoice...");

    //Get the schema of the Invoice & Memo form (SO643000)
    //by using the screen-based API wrapper
    Content invoiceFormSchema =
        PX.Soap.Helper.GetSchema<Content>(context);
    //Specify the needed invoice and get a PDF version of it
    var commands = new Command[]
    {
        new Value
        {
            Value = docType,
            LinkedCommand = invoiceFormSchema.Parameters.DocumentType
        },
        new Value
        {
            Value = invoiceNbr,
            LinkedCommand = invoiceFormSchema.Parameters.ReferenceNumber
        },
        invoiceFormSchema.ReportResults.PdfContent
    };

    //Submit the commands to the form
    Content[] pdfInvoice = context.Submit(commands);

    //Save the result in a PDF file
    if (pdfInvoice != null && pdfInvoice.Length > 0)
    {
        File.WriteAllBytes(string.Format(@"Invoice_{0}.pdf", invoiceNbr),

 Convert.FromBase64String(pdfInvoice[0].ReportResults.PdfContent.Value));
    }
}
```

7. In the `Main()` method of the `Program` class, call the `GetPrintableInvoice()` method of the `Reports` class, as the following code shows. In this code, modify the reference to the web service in the `Url` property of the `Screen` object and the username and password that are passed to the `Login()` method, if necessary.

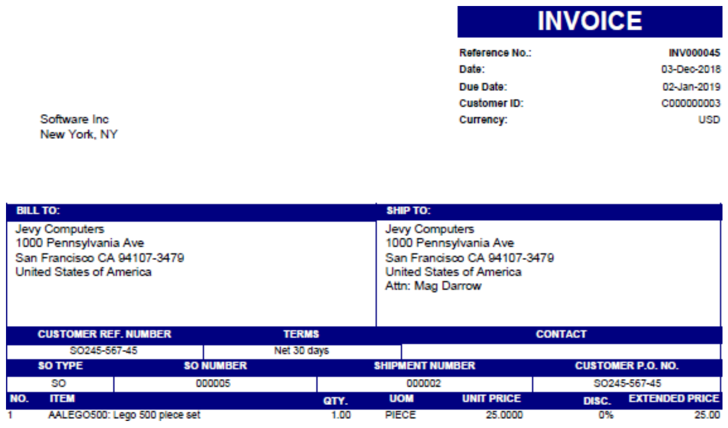
```
using System;
using MyStoreIntegrationSBAPI.SO643000;
using MyStoreIntegrationSBAPI.Integration;

namespace MyStoreIntegrationSBAPI
{
    class Program
    {
        static void Main(string[] args)
        {
            using (Screen screen = new Screen())
            {
                //Specify the connection parameters
                screen.CookieContainer = new System.Net.CookieContainer();
                screen.Url = Properties.Settings.
                  Default.MyStoreIntegrationSBAPI_SO643000_Screen;

                //Sign in to MYOB Advanced
                screen.Login
                (
                    "admin",
                    "123"
                );

                try
                {
                    Reports.GetPrintableInvoice(screen);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                    Console.WriteLine();
                    Console.WriteLine("Press any key to continue");
                    Console.ReadLine();
                }
                finally
                {
                    //Sign out from MYOB Advanced
                    screen.Logout();
                }
            }
        }
    }
}
```

8. Rebuild the project, and run the application. The invoice is saved in the `Invoice_INV000045.pdf` file in `\bin\Debug` folder of the *MyStoreIntegration* project if you run the project in the Debug mode. A fragment of the `Invoice_INV000045.pdf` file is shown in the following screenshot.



INVOICE

Reference No.:	INV000045
Date:	03-Dec-2018
Due Date:	02-Jan-2019
Customer ID:	C000000003
Currency:	USD

Software Inc
New York, NY

BILL TO:	SHIP TO:
Jevy Computers 1000 Pennsylvania Ave San Francisco CA 94107-3479 United States of America	Jevy Computers 1000 Pennsylvania Ave San Francisco CA 94107-3479 United States of America Attn: Mag Darrow

CUSTOMER REF. NUMBER	TERMS	CONTACT
SO245-567-45	Net 30 days	

SO TYPE	SO NUMBER	SHIPMENT NUMBER	CUSTOMER P.O. NO.
SO	000005	000002	SO245-567-45

NO.	ITEM	QTY.	UOM	UNIT PRICE	DISC.	EXTENDED PRICE
1	AALEGO500: Lego 500 piece set	1.00	PIECE	25.0000	0%	25.00

Figure: Printable version of the invoice

Lesson Summary

In this lesson, you have added to the MyStoreIntegration application the method that obtains a printable version of an invoice. To get the PDF version of an invoice, you have used the `PdfContent` command of the `ReportResults` subobject of the `Content` object that corresponds to the Invoice & Memo (SO643000) report form.

The following table summarizes whether PDF reports can be retrieved from MYOB Advanced by using the integration interfaces.

Integration Interface	**Retrieval of PDF Reports**
OData interface	No
REST API	No
Contract-based SOAP API	No
Screen-based SOAP API	Yes

Appendix: Comparison of the Integration Interfaces

The following table summarizes the differences between the integration interfaces described in this training guide. For each integration interface, the table shows whether the scenario mentioned in the header cell can be implemented with this integration interface.

OAuth 2.0 Authorization

Integration Interface	OAuth 2.0 Authorization
OData interface	No
REST API	Yes
Contract-based SOAP API	Yes
Screen-based SOAP API	No

Optimized Retrieval of Multiple Kinds of Detail Lines

Integration Interface	Optimized Retrieval of Multiple Kinds of Detail Lines
OData interface	Yes, if you create multiple custom generic inquiries (one inquiry for each kind of detail line)
REST API	Yes (you need to perform multiple requests: one request for each kind of detail line)
Contract-based SOAP API	Yes (you need to perform multiple requests: one request for each kind of detail line)
Screen-based SOAP API	Yes, if you create multiple custom generic inquiries (one inquiry for each kind of detail line)

Retrieval of Attachments

Integration Interface	Retrieval of Attachments
OData interface	No
REST API	Yes
Contract-based SOAP API	Yes
Screen-based SOAP API	Yes (if the names of the attached files are known)

Retrieval of Reports

Integration Interface	Retrieval of PDF Reports
OData interface	No
REST API	No
Contract-based SOAP API	No
Screen-based SOAP API	Yes

Appendix: Web Integration Scenario Reference

In this topic, you can find reference links to the topics of this training course that describe how to implement the following web integration scenarios:

- **Authorizing the application to work with MYOB Advanced**: *Lesson 1.1: Registering the Application in MYOB Advanced*, *Lesson 1.2: Configuring the Application to Use OAuth 2.0*
- **Signing out from MYOB Advanced in an OAuth 2.0 application**: *Lesson 1.3: Signing Out from MYOB Advanced*
- **Listing the sales orders of a customer**: *Lesson 2.1: Retrieving a List of Sales Orders with Details and Related Shipments*
- **Listing the payments of a customer**: *Lesson 2.2: Retrieving the List of Payments One by One*
- **Retrieving the images of a stock item**: *Lesson 3.1: Retrieving the Attachments of a Stock Item*
- **Generating a printable invoice by invoice ID**: *Lesson 4.1: Generating the Printable Version of an Invoice*

Appendix: Troubleshooting

I get the error *API Login Limit* when my application requests access to MYOB Advanced web services APIs through OAuth 2.0. What should I do?

For an application that uses OAuth 2.0 for authorization in MYOB Advanced, this error appears if all of the following are true:

- The API login limit is specified in the MYOB Advanced license. The license restriction for the API users is shown in the **Maximum Number of Web Services API Users** box on the **License** tab of the License Monitoring Console (SM604000) form.
- The number of unclosed sessions (that is, the sessions in which you have signed in to MYOB Advanced through one of the web services APIs or obtained access to MYOB Advanced web services APIs through OAuth 2.0 and have not signed out from MYOB Advanced) equals the API login limit in the license.
- You try to request access to web services APIs through OAuth 2.0 once more.

You can deal with this error as follows:

1. Modify the code of your application so that it signs out from MYOB Advanced each time the work with MYOB Advanced is finished. For details, see *Lesson 1.3: Signing Out from MYOB Advanced*.
2. If the integration application has not closed the session, you can do one of the following:
    - Pass the access token that was used during the previous session and sign out from MYOB Advanced.
    - Wait for one hour until the session that has been opened through OAuth 2.0 expires.
    - Restart the site in the Internet Information Services (IIS) Manager or by clicking the **Restart Application** button on the toolbar of the Apply Updates (SM203510) form.

When I retrieve the list of records from MYOB Advanced through the contract-based API, I get the error *Optimization cannot be performed. The following fields cause the error*. What should I do?

When multiple records are retrieved from MYOB Advanced through an endpoint with Contract Version 3, the system tries to optimize the retrieval of the records and obtain all needed records in one request to the database (instead of requesting the records one by one). If the optimization fails, the system returns an error, which specifies the entities or fields that caused the failure of the optimized request. To prevent the error from occurring, you can do one of the following:

- If you do not need to retrieve the entities or fields that caused the failure, you can exclude these entities or fields from the request as follows:
    - For the REST API:
        - Exclude the entities from the entities specified in the `$expand` parameter.
        - Explicitly specify the other fields to be returned (while excluding the fields that caused the failure) by using the `$select` parameter.
    - For the SOAP API:
        - Exclude the entities by setting the `ReturnBehavior` property to `None` for the entities
        - Explicitly specify the other fields to be returned (while excluding the fields that caused the failure) by using the `Return` and `Skip` classes of the needed value type (such as `StringReturn` and `StringSkip`) for the fields.

- If you need to retrieve the entities or fields that caused the failure, you can retrieve the needed records one by one either by key fields or by IDs.

For an example of how to deal with the optimization errors, see *Lesson 2.2: Retrieving the List of Payments One by One*.

When I retrieve the list of records from MYOB Advanced through the contract-based API, I get the error *More than one detail properties have been used in the request*. What should I do?

Multiple records with multiple kinds of detail lines cannot be retrieved from MYOB Advanced in one request. Instead, you should request each kind of detail lines in a separate request. For an example of how to request multiple kinds of detail lines, see *Lesson 2.1: Retrieving a List of Sales Orders with Details and Related Shipments*.