Developer Course

Web Services

I330 Data Manipulation with REST API 2022 R1

Revision: 3/31/2022

Acumatica
The Cloud ERP

Contents

Copyright

3933 Lake Washington Blvd NE, # 350, Kirkland, WA 98033

Restricted Rights

The product is provided with restricted rights. Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in the applicable License and Services Agreement and in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (c)(2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable.

Disclaimer

Acumatica, Inc. makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Acumatica, Inc. reserves the right to revise this document and make changes in its content at any time, without obligation to notify any person or entity of such revisions or changes.

Trademarks

Acumatica is a registered trademark of Acumatica, Inc. HubSpot is a registered trademark of HubSpot, Inc. Microsoft Exchange and Microsoft Exchange Server are registered trademarks of Microsoft Corporation. All other product names and services herein are trademarks or service marks of their respective companies.

Software Version: 2022 R1

Last Updated: 03/31/2022

Introduction

External systems can use Acumatica ERP integration interfaces to access the business functionality and data of Acumatica ERP. Acumatica ERP 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 how you can submit data to Acumatica ERP and process data in Acumatica ERP with the contract-based REST API. The data submission and data processing with the contract-based and screen-based SOAP APIs is outside of the scope of this course. The OData interface cannot be used for this type of integration.

Information about data retrieval through the OData and contract-based REST integration interfaces is available in the *I300 Data Retrieval with OData*, *I310 Data Retrieval with REST API Basics*, and *I320 Advanced Data Retrieval with REST API* training courses, which are prerequisites for this course.

This course is intended for developers who need to create applications that interact with Acumatica ERP.

The course is based on a set of examples of web integration scenarios that demonstrate the process of developing a client application that uses the contract-based REST API. The course gives you ideas about how to develop your own applications by using the contract-based REST API.

After you complete all the lessons of the course, you will be familiar with the advanced techniques of data submission and data processing through the Acumatica ERP contract-based REST API.

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, the example for the integration interface, and the lesson summary.
3. In Partner University, take *I330 Certification Test: Data Manipulation*.

After you pass the certification test, you will be given the Partner University certificate of course completion.

What Is in a Part?

All 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 Acumatica ERP.

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 contract-based REST API. Each lesson consists of a brief description of the web integration scenario and an example 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.

What Are the Documentation Resources?

The complete Acumatica ERP and Acumatica Framework documentation is available on *https://help.acumatica.com/* and is included in the Acumatica ERP instance. While viewing any form used in the course, you can click the **Open Help** button in the top pane of the Acumatica ERP screen to bring up a form-specific Help menu; you can use the links on this menu to quickly access form-related information and activities and to open a reference topic with detailed descriptions of the form elements.

Licensing Information

For the educational purposes of this course, you use Acumatica ERP under the trial license, which does not require activation and provides all available features. For the production use of the Acumatica ERP functionality, an administrator has to activate the license the organization has purchased. Each particular feature may be subject to additional licensing; please consult the Acumatica ERP sales policy for details.

Course Prerequisites

To complete this course, you should be familiar with the financial and distribution functionality of Acumatica ERP, the basic principles of the system, and the principles of data retrieval with the Acumatica ERP contract-based REST API. We recommend that you complete the *I310 Data Retrieval with REST API Basics* and *I320 Advanced Data Retrieval with REST API* training courses before you go through this course.

You need to perform the prerequisite actions described in this part before you start to complete the course.

1. Make sure the environment that you are going to use for the training course conforms to the *System Requirements for Acumatica ERP 2022 R1*.
2. Make sure that the Web Server (IIS) features that are listed in *Configuring Web Server (IIS) Features* are turned on.
3. Deploy an instance of Acumatica ERP2022 R1 with the name *MyStoreInstance* and a tenant that contains the *I100* data. If you have completed the *I310 Data Retrieval with REST API Basics* or *I320 Advanced Data Retrieval with REST API* training course, you can use the instance that you have deployed for this course. For information on how to deploy the instance for the training course, see *Deploying an Acumatica ERP Instance for the Training Course*.
4. Make sure the following conditions are met:
   - The Postman application should be installed on your computer. To download and install Postman, follow the instructions on *https://www.postman.com/downloads/*.
   - A Postman collection should be configured to use the OAuth 2.0 authorization, or the requests for signing in to and signing our from Acumatica ERP should be included in the collection. You can use the Postman collection that you have created in the *I320 Advanced Data Retrieval with REST API* training course. For details about how to configure OAuth 2.0 authorization, see Part 1 in the *I320 Advanced Data Retrieval with REST API* training course. For details about the sign-in and sign-out methods, see *Sign In to the Service* and *Sign Out from the Service* in the documentation.
5. Make sure you have HTTP access from the computer where you work with the examples to the Acumatica ERP instance.
6. If you want to work with the examples that describe how to process credit card payments through the Acumatica ERP website or if you use the OAuth 2.0 authorization, configure HTTPS on the Acumatica ERP website, as described in *Configuring a Website for HTTPS*. If you have configured HTTPS on the Acumatica ERP website for the *I320 Advanced Data Retrieval with REST API* training course and you use the same site for the current course, you do not need to configure HTTPS. If you are not going to process credit card payments and you do not use the OAuth 2.0 authorization, you can call API methods via HTTP.

Deploying an Acumatica ERP Instance for the Training Course

Instead of deploying a new instance, you can use the Acumatica ERP instance that you have deployed for the *I310 Data Retrieval with REST API Basics* or *I320 Advanced Data Retrieval with REST API* training course.

You deploy an Acumatica ERP instance and configure it as follows:

1. Open the Acumatica ERP Configuration Wizard, and deploy a new application instance as follows:
   a. On the **Database Configuration** page of the Acumatica ERP Configuration Wizard, type the name of the database: `MyStoreInstance`.
   b. On the **Tenant Setup** page, set up one tenant with the *I100* data inserted by specifying the following settings:

- **Login Tenant Name**: `MyStore`
- **New**: Selected
- **Insert Data**: *I100*
- **Parent Tenant ID**: 1
- **Visible**: Selected

The system creates a new Acumatica ERP 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 Acumatica ERP 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, Acumatica ERP, and the Authorize.Net payment gateway for making transactions to the Authorize.Net payment gateway through the Acumatica ERP website, as shown in the following diagram. (Acumatica ERP uses the Authorize.Net payment gateway for processing credit card payments.)

You can make API calls to Acumatica ERP (item 1 in the diagram) through HTTP; however, we recommend that you use a secure connection between the API client application and Acumatica ERP for credit card processing.
A secure connection between the Authorize.Net payment gateway and the Acumatica ERP website (item 2 in the diagram) with a Secure Socket Layer (SSL) certificate is required for making transactions. Therefore, you have to set up the Acumatica ERP website for HTTPS to be able to process credit card payments, as described in this topic.

Figure: API interaction with Acumatica ERP and Authorize.Net

A secure connection between the client application and the Acumatica ERP website with an SSL certificate is also required for the authorization of the client application through OAuth 2.0.

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**.

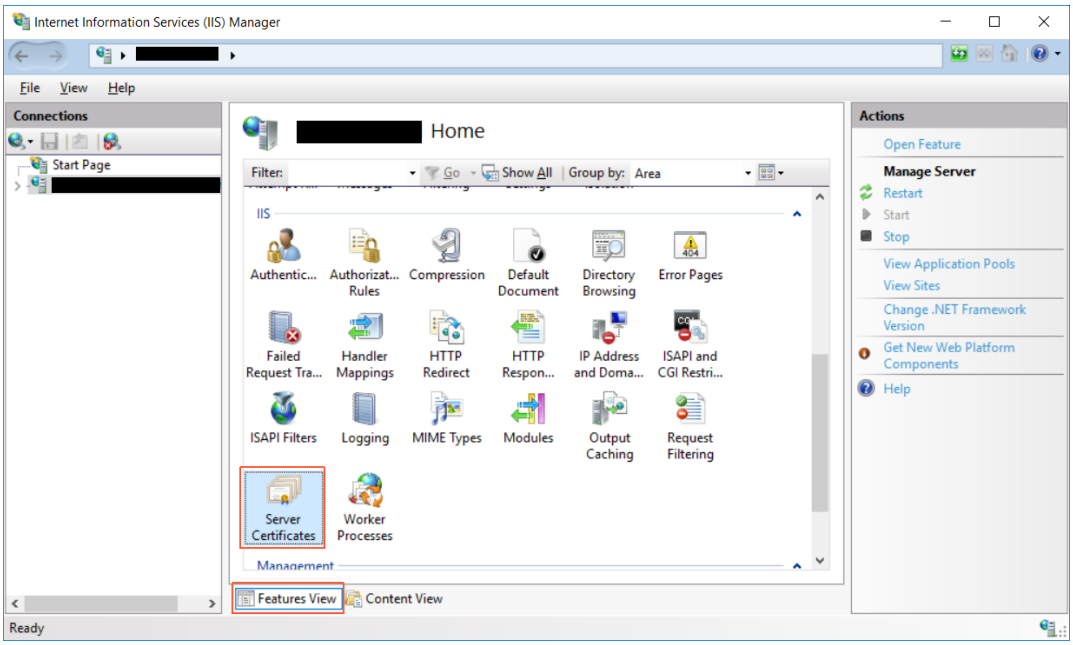

Figure: IIS Manager. Server Certificates icon

   c. Click **Create Self-Signed Certificate** in the **Actions** pane.

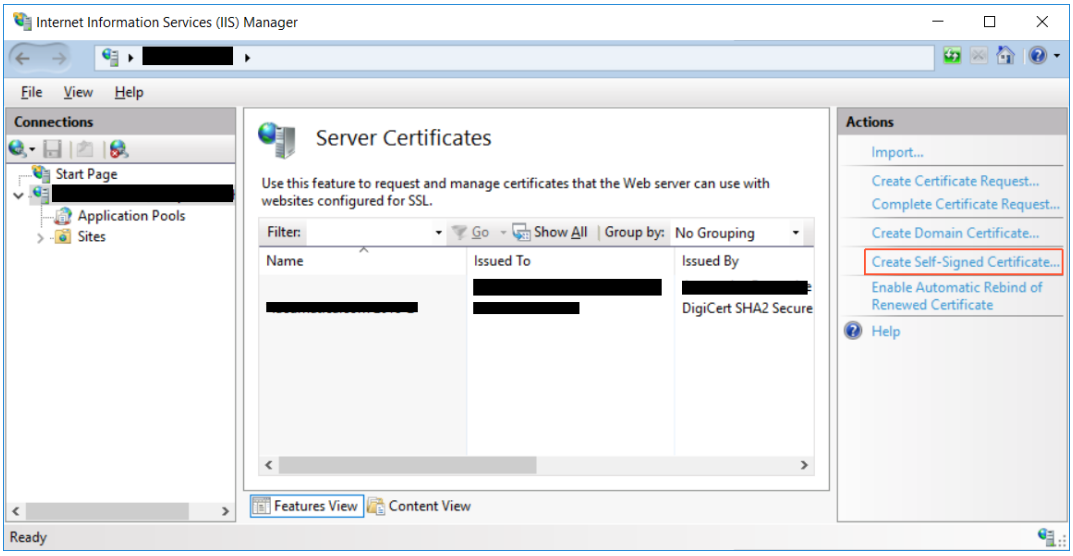

Figure: IIS Manager. Create Self-Signed Certificate link

   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.

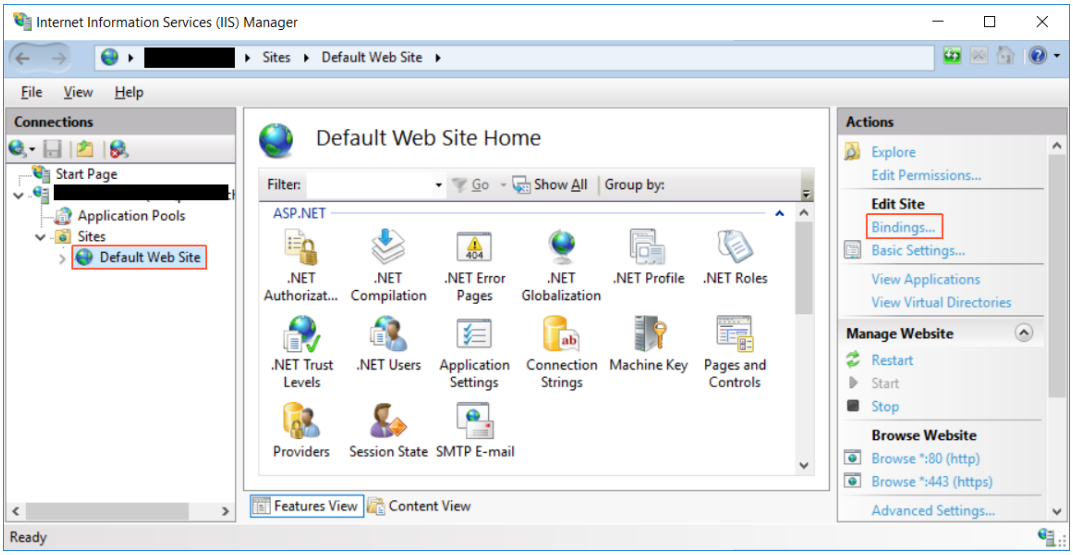

Figure: IIS Manager. Bindings link

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)*).

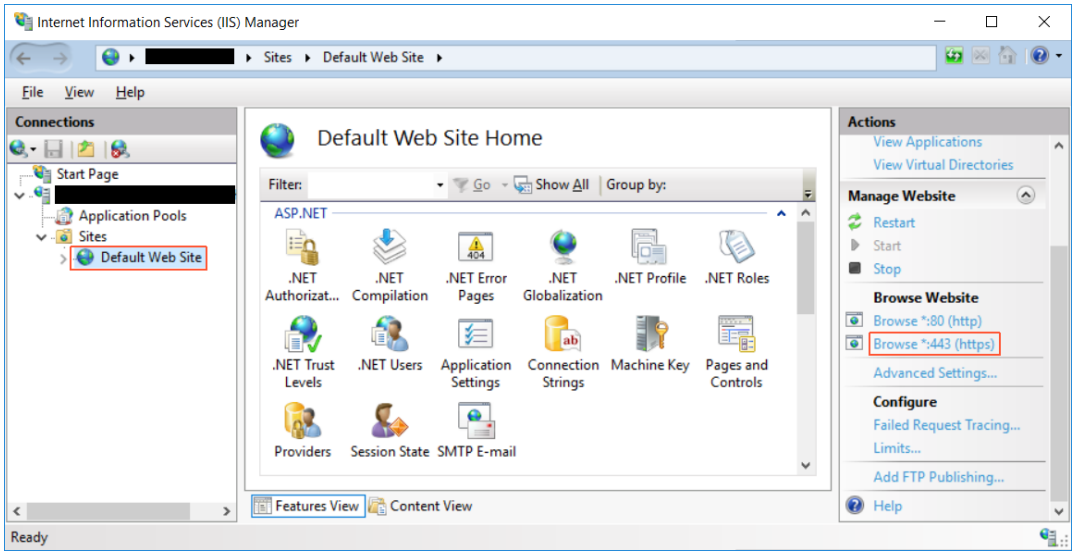

Figure: Opening the HTTPS website

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 Acumatica ERP 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 Acumatica ERP for customer management, sales order processing, and payment collection.
MyStore has been acting upon plans to extend its business and start selling goods online. To do this, MyStore has been investigating the options available in Acumatica ERP for integration with eCommerce applications.

In the first stage of the implementation, which is covered in the *I320 Advanced Data Retrieval with REST API* training course, MyStore has developed the MyStoreIntegration integration application. Due to the development during this stage, MyStoreIntegration retrieves information about stock items, sales orders, and payments from Acumatica ERP.

In the second stage of the implementation, which this course covers, MyStoreIntegration will be expanded to submit information about customers, sales orders, and payments from the online store to Acumatica ERP. For the implementation of the MyStoreIntegration application, the MyStore company can use the contract-based REST API.

The OData interface can be used only for the implementation of the data retrieval part of the MyStoreIntegration application; data submission should be performed by other integration interfaces because data submission is not possible through OData. The examples of this course show the implementation of the MyStoreIntegration application with the contract-based REST API.

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

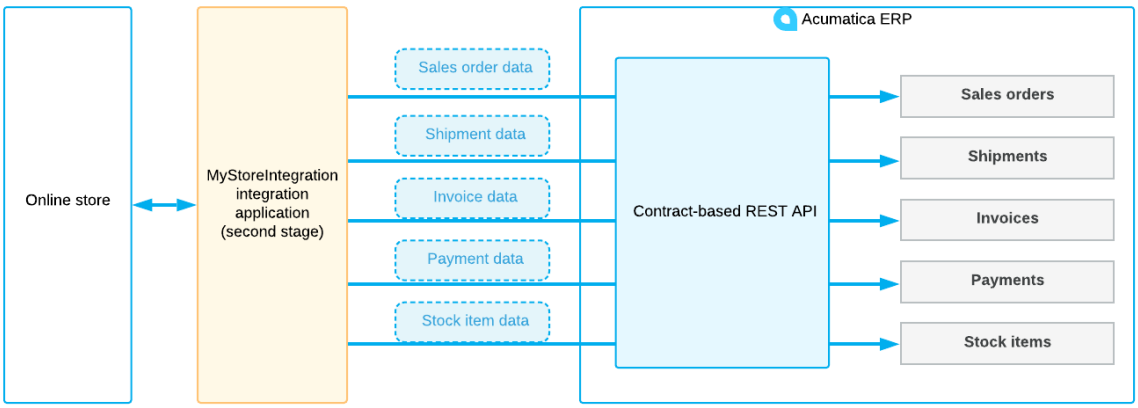

Figure: Integration of the MyStore online store and Acumatica ERP

Integration Requirements

Two types of users work with the online store application of the MyStore company: the customers who purchase goods, and the administrators who manage the online store.

In the second stage of implementation, the MyStoreIntegration application should implement integration with Acumatica ERP to support the following usage scenarios in the online store:

- A registered customer should be able to do the following:
  - Purchase goods
  - Update the purchase before it is shipped
  - Have the goods shipped
  - Register a credit card in the online store
- An administrator should be able to do the following:

- Add new stock items to the catalog of the online store
- Add notes and attachments to stock items

In this course, you do not implement the online store application itself; instead, you implement the integration part between the online store and Acumatica ERP, which provides the support for the listed scenarios in the online store application.

Part 1: Creation of Records

In this part of the course, you will create records through the Acumatica ERP contract-based REST API. You will create a shipment for particular sales orders. You will also create a stock item with attributes.

As a result of completing the lessons of this part, you will know the best practices associated with the creation of a record with detail lines through the contract-based REST API. You will also learn how to identify the attributes whose values you want to assign.

Lesson 1.1: Creating a Shipment for Sales Orders

A customer of the MyStore online store should be able to request a shipment of the goods that the customer has ordered. The customer can add multiple orders to the shipment. Information about the requested shipment should be passed to Acumatica ERP, where each shipment can be displayed on the Shipments (SO302000) form.

In the example of this lesson, you will add to the MyStoreIntegration application a request that creates a shipment for a customer from two sales orders, both of which have the *Open* status; you will do this by using the contract-based REST API. You will add all items from the sales orders to the shipment. To add all items of a sales order to a shipment, you need to specify only the order type and order number in the details of the shipment. You can use a similar request to add any number of sales orders to the shipment.

If you need to include in a shipment particular items (rather than all items) from a sales order, you need to retrieve the sales orders with the included items from Acumatica ERP by the key fields, and then include the needed items in the shipment (by specifying the inventory ID, the warehouse ID, and the type and number of the sales order in which the item is included in the details of the shipment).

Shipments are auto-numbered in the system; therefore, during the creation of the shipment, you will not specify the shipment number, which is the key field of the document. As a result of the example in this lesson being executed, the system will create a shipment with the *On Hold* status and assign the next shipment number to it.

For more information on the workflow of sales order processing, see *Sales of Stock Items: General Information* in the documentation.

You will use the `Shipment` entity of the *Default/20.200.001* endpoint; this entity is mapped to the Shipments form.

Although you are creating a shipment with multiple detail lines, you will use one request for the creation of the shipment. (That is, you do not need to submit each detail of the shipment in a separate request.) For optimal performance of the application, it is important to use the minimum number of requests.

In this lesson, you will not specify the values of any custom fields (that is, fields that have been added to Acumatica ERP forms as part of a customization project) or user-defined fields, which is outside of the scope of this course. For basic information about how to specify the values of custom and user-defined fields, see *Additional Information: Setting of the Values of Custom and User-Defined Fields*.
You will not specify the values of multi-language fields (that is, text boxes on Acumatica ERP forms in which users can type values in multiple languages if multiple locales have been configured in the system) either. For basic information about multi-language fields, see *Additional Information: Setting of the Values of Multi-Language Fields*.

Lesson Objective

In this lesson, you will learn how to create a new record in Acumatica ERP by using the contract-based REST API.

Prerequisites

Before you complete the examples of this lesson, on the *Sales Orders* (SO301000) form, find the sales orders with the order numbers *000004* and *000006*. These are the preconfigured sales orders of the customer with the customer ID *C000000003*. Make sure that these sales orders have the *Open* status. The sales order *000004*, shown in the following screenshot, includes two stock items (that is, two detail lines). The sales order *000006* includes one stock item.

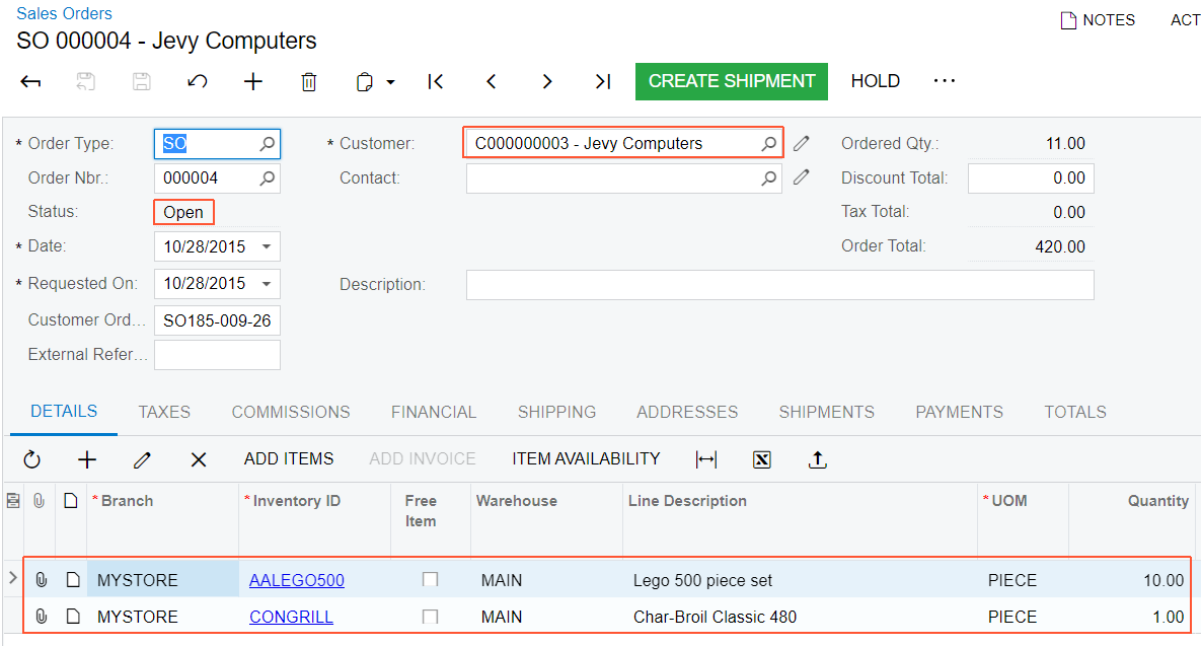

Figure: One of two sales order to create a shipment for

> If you want to perform an example of this lesson multiple times with the same data, you need to remove the shipment that is created as a result of the completion of the example before you try to create the shipment once again. To remove the shipment, on the *Shipments* (SO302000) form, select the shipment, and click **Delete** on the form toolbar.
>
> When you complete the example in this lesson, you can also use the data of any other two sales orders with the *Open* status created for the same customer. If you do this, you need to update the data in the example accordingly.

Example: Using One PUT Request

In this example, by using the contract-based REST API, you will create a shipment for multiple sales orders.

To retrieve sales orders with the included items for the shipment, you will use the `GET` HTTP method with the `$expand` and `$select` parameters. You will specify the key fields of the sales order in the URL of the request.

To submit a shipment with all items of the particular sales orders to Acumatica ERP, you will use one `PUT` HTTP request. You will specify the data of the new shipment in the body of the request. In the `PUT` request, you will use the `$expand` and `$select` parameters to limit the list of fields whose values are returned in the response. The response body will contain only the fields specified in the `$select` parameter and the fields specified in the body of the request. (In the `$expand` parameter, you have to specify all the nested entities whose fields you specify

in the `$select` parameter.) If you do not specify any parameters, the response will contain all fields of the new record.

Creating a Shipment for Sales Orders

To create a shipment for two sales orders by using the contract-based REST API, do the following:

> ⚠ Before you proceed with these instructions, you need to obtain the access token (if your application uses OAuth 2.0 authorization) or perform the sign-in request.

1. Create the shipment for the *000004* and *000006* sales orders as follows:
   a. In the Postman collection, add a contract-based REST API request with the following settings:

> ⚠ Instead of creating a collection, you can import to Postman the collection provided with this course (`REST.postman_collection.json`). This collection has been configured for this course and already contains all the requests that are used in the course. You can use this collection for testing the requests. (You can find the file in *https://github.com/Acumatica/Help-and-Training-Examples/tree/2022R1/IntegrationDevelopment/I330*.)

   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/Shipment*
   - The following parameters of the request

Parameter	Value
`$expand`	`Details`
`$select`	`Type,ShipmentNbr,Status,Details/InventoryID`

   - The headers shown below

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

   - The following body of the request

```
{
 "Type":{"value":"Shipment"},
 "CustomerID":{"value":"C000000003"},
 "WarehouseID":{"value":"MAIN"},
 "Details":[
  {
   "OrderType":{"value":"SO"},
   "OrderNbr":{"value":"000004"}
  },
  {
   "OrderType":{"value":"SO"},
   "OrderNbr":{"value":"000006"}
  }
 ]
```

```
}
```

b. Send the request. If the request is successful, the response contains the `200 OK` status code and includes the list of the requested fields of the new shipment record. The following code example shows the response body.


```
{
    "id": "5b1037da-b27a-eb11-9dd3-9828a61840c3",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "CustomerID": {
        "value": "C000000003"
    },
    "Details": [
        {
            "id": "601037da-b27a-eb11-9dd3-9828a61840c3",
            "rowNumber": 1,
            "note": null,
            "InventoryID": {
                "value": "AALEGO500"
            },
            "OrderNbr": {
                "value": "000004"
            },
            "OrderType": {
                "value": "SO"
            },
            "custom": {}
        },
        {
            "id": "38c554e0-b27a-eb11-9dd3-9828a61840c3",
            "rowNumber": 2,
            "note": null,
            "InventoryID": {
                "value": "CONGRILL"
            },
            "OrderNbr": {
                "value": "000004"
            },
            "OrderType": {
                "value": "SO"
            },
            "custom": {}
        },
        {
            "id": "3ac554e0-b27a-eb11-9dd3-9828a61840c3",
            "rowNumber": 3,
            "note": null,
            "InventoryID": {
                "value": "AAMACHINE1"
            },
            "OrderNbr": {
                "value": "000006"
            },
            "OrderType": {
```

```
                "value": "SO"
            },
            "custom": {}
        }
    ],
    "ShipmentNbr": {
        "value": "000008"
    },
    "Status": {
        "value": "On Hold"
    },
    "Type": {
        "value": "Shipment"
    },
    "WarehouseID": {
        "value": "MAIN"
    },
    "custom": {}
}
```

c. Save the request.

On the *Shipments* (SO302000) form, make sure the shipment with the number returned in the `ShipmentNbr` field exists, has the *On Hold* status, and contains three detail lines, as shown in the following screenshot.

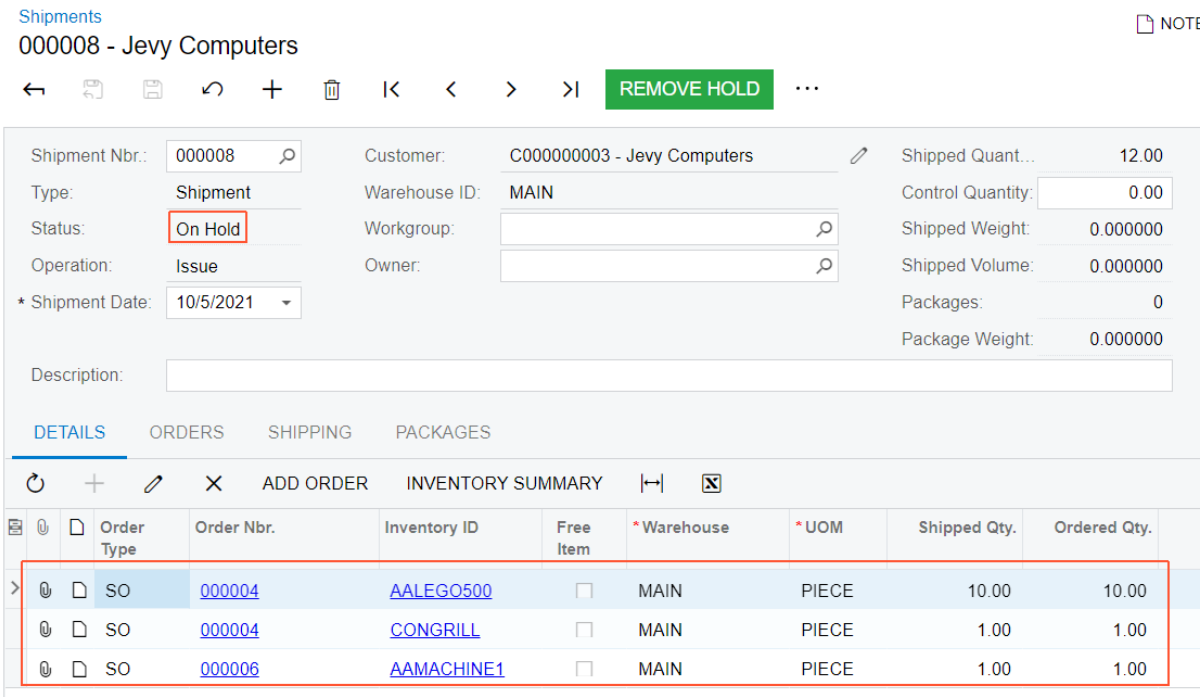

Figure: The shipment with detail lines

Related Links

- *Create a Record*
- *Representation of a Record in JSON Format*
- *Parameters for Retrieving Records*

Additional Information: Setting of the Values of Custom and User-Defined Fields

In a customization project, you can add custom fields to Acumatica ERP forms. You can also add user-defined fields to Acumatica ERP forms and include them in a customization project. (For details about user-defined fields, see *User-Defined Fields*.)

To specify the values of custom and user-defined fields through the contract-based REST API, you specify the values of these fields in the body of the `PUT` request. You specify the values of custom and user-defined fields in JSON format, as described in *Representation of a Record in JSON Format*. You can use the same approach to specify the values of any fields that are not included in the entity definition. For details about working with fields that are not included in the entity definition, see *Custom Fields* in the documentation.

You can also add custom fields to a custom endpoint or endpoint extension and work with them by using the same approach as was described in this lesson for the fields of the system endpoint. User-defined fields cannot be added to custom endpoints and endpoint extensions. For details about custom endpoints and endpoint extensions, see *Custom Endpoints and Endpoint Extensions* in the documentation.

Additional Information: Setting of the Values of Multi-Language Fields

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales have been configured in Acumatica ERP. You can specify the values of these fields through the contract-based REST API. (This scenario is outside of the scope of this course but may be useful to some readers.)

For details about how to specify the values of multi-language fields in REST API requests, see *Specify Localized Values of a Multi-Language Field*.

Lesson Summary

In this lesson, you have learned how to create a record in Acumatica ERP by using the contract-based REST API. During record creation, you have submitted the fully configured record (with all detail lines) in one request.

Lesson 1.2: Creating a Stock Item with Attributes

The administrator of the MyStore company's online store can add inventory items to the catalog by using the administrator interface of the online store. The online store passes the settings of the added stock items to Acumatica ERP by using the contract-based API.

In this lesson, you will add to the MyStoreIntegration REST application a method that creates a stock item record with attributes in Acumatica ERP. An attribute is a special property of an object in the system that specifies additional information that is not defined by the standard properties of the object (that is, those supported by the standard UI elements).

For this lesson, in the Acumatica ERP instance that you are using for the training course, two attributes have been preconfigured to provide the ability to add and track additional information to stock items. The attributes with the *OPERATSYST* and *SOFTVER* identifiers have been configured on the *Attributes* (CS205000) form. These attributes have been assigned to the *STOCKITEM* item class on the *Item Classes* (IN201000) form. Thus, if you select this item class in the **Item Class** box on the **General** tab of the *Stock Items* (IN202500) form for a stock item, the **Attributes** tab is available on the form and you can specify the values of the attributes. In this lesson, you will specify the values of these attributes for a new stock item through the contract-based API.

To create a stock item, you will use the `StockItem` entity of the *Default/20.200.001* endpoint. The `StockItem` entity is mapped to the Stock Items form. In the stock item data, you will specify the value of the inventory ID, which is the key field of a stock item. Because the key field value is passed in the stock item data, the system searches for a stock item record with the specified key and does one of the following:

- If the record has been found, updates this record
- If the record has not been found, adds a new stock item record

To specify the values of attributes, you will use the `Attributes` field of the `StockItem` entity. To identify the attribute whose value you want to specify, in the `AttributeID` field of the `AttributeValue` entity, you will specify the attribute name, which can be found in the **Description** box on the *Attributes* (CS205000) form, or the attribute identifier.

Lesson Objective

In this lesson, you will learn how to create records with attributes.

Example: Using the Attributes Field

In this example, by using the contract-based REST API, you will create in Acumatica ERP the *BASESERV1* stock item, which has the **Operation System** and **Version of Software** attributes specified.

To create a stock item, you will use the `PUT` HTTP method. You will specify the data of the new stock item (including the values of the attributes) in the body of the request. In the `PUT` request, you will use the `$expand` and `$select` parameters to limit the list of fields whose values are returned in the response. The response will contain the values of the fields that you specify in the `$select` parameter and may contain some other fields.

Creating a Stock Item with Attributes

To create a stock item by using the contract-based REST API, do the following:

1. In Postman, add a request with the following settings:
   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/StockItem*
   - The following parameters of the request

Parameter	Value
`$expand`	`Attributes`
`$select`	`InventoryID,Attributes/AttributeDescription,Attributes/Value`

   - The headers shown below

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

   - The following body of the request

```
{
 "InventoryID":{"value":"BASESERV1"},
 "Description":{"value":"Baseline level of performance"},
 "ItemClass":{"value":"STOCKITEM"},
 "Attributes":[
  {
   "AttributeID":{"value":"Operation System"},
   "Value":{"value":"Windows"}
  },
  {
   "AttributeID":{"value":"SOFTVER"},
   "Value":{"value":"Server 2012 R2"}
  }
 ]
}
```

2. Send the request. If the request is successful, the response contains the `200 OK` status and includes the list of requested fields of the new stock item record in JSON format. The following code example shows the response body.

```
{
    "id": "68bf203e-9f7d-eb11-9dd4-9828a61840c3",
    "rowNumber": 1,
    "note": null,
    "Attributes": [
        {
            "id": "af766044-9f7d-eb11-9dd4-9828a61840c3",
            "rowNumber": 1,
            "note": null,
            "AttributeDescription": {
                "value": "Operation System"
            },
            "AttributeID": {
                "value": "OPERATSYST"
            },
            "Value": {
                "value": "Windows"
            },
            "custom": {}
        },
        {
            "id": "b0766044-9f7d-eb11-9dd4-9828a61840c3",
            "rowNumber": 2,
            "note": null,
            "AttributeDescription": {
                "value": "Version of Software"
            },
            "AttributeID": {
                "value": "SOFTVER"
            },
            "Value": {
                "value": "Server 2012 R2"
            },
            "custom": {}
        }
    ],
    "Description": {
```

```
        "value": "Baseline level of performance"
    },
    "InventoryID": {
        "value": "BASESERV1"
    },
    "ItemClass": {
        "value": "STOCKITEM"
    },
    "custom": {}
}
```

3. Save the request.

Related Links

- *Create a Record*

Lesson Summary

In this lesson, you have learned how to create records with attributes through the contract-based REST API. To specify the values of the attributes of a stock item, you have used the `Attributes` field of the `StockItem` entity. To identify the attribute whose value you need to specify, in the `AttributeID` field of the `AttributeValue` entity, you have specified the attribute name or attribute identifier.

Part 2: Update of Records

In this part of the course, you will update records in Acumatica ERP through the contract-based REST API. You will update a customer record by using one request. You will update the detail lines of a sales order by retrieving the sales order with detail lines from Acumatica ERP in one request and then submitting the updated sales order in another request.

As a result of completing the lessons of this part, you will know which techniques to use when you are updating a record or its detail lines through the contract-based REST API.

Lesson 2.1: Updating a Customer Account

In the online store of the MyStore company, a customer can order goods if an account has been created for the customer in the online store. The customer can view and edit the information in the customer account. When a customer account is created or updated, the online store needs to pass customer data to Acumatica ERP.

The MyStore company wants to use email addresses for the authorization of customers to use the online store. Therefore, to search for a customer record in Acumatica ERP, the MyStoreIntegration application will use the customer's email address. If a registered customer needs to update some customer data, the online store will submit the email address of the customer and the updated information to Acumatica ERP, where customer account data is entered and maintained on the *Customers* (AR303000) form.

A customer of the MyStore online store should also be able to specify a billing contact in addition to the main contact, which the customer specifies during initial registration to the online store. In Acumatica ERP, this contact is specified on the **Billing** tab of the Customers form.

In this lesson, you will update a customer record in Acumatica ERP by using the contract-based REST API: You will specify a different customer class for an existing customer record and add a new billing contact for this customer. You will search for the needed record by using the email address of the customer.

You will use the `Customer` entity of the *Default/20.200.001* endpoint. This entity is mapped to the Customers form. You will use one request to update both the customer class and the billing contact.

The removal of a record is outside of the scope of this course. For basic information about this scenario, see *Additional Information: Removal of a Record*.

Lesson Objective

In this lesson, you will learn how to update an existing record by using the contract-based REST API.

Prerequisites

In this lesson, you will update the customer class of the customer record that has the *info@jevy-comp.con* email address. Before you complete the examples of this lesson, on the *Customers* (AR303000) form, select this record, which has the *C000000003* customer ID, and view its settings. Notice that this customer record currently is assigned the *DEFAULT* customer class, as shown in the following screenshot.

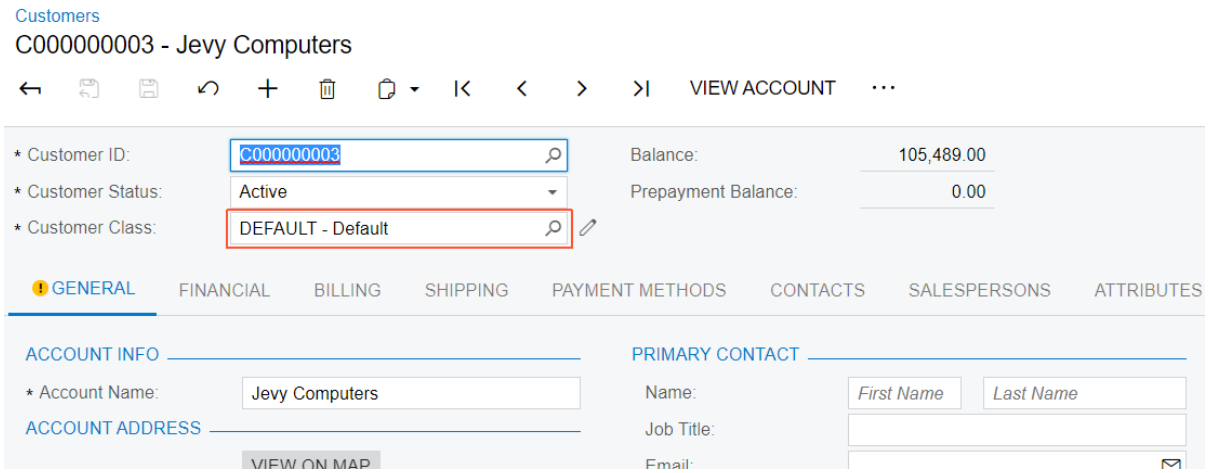

Figure: Customer record

On the **Billing** tab, you can see that this customer currently has the same billing contact as the main contact of the customer. That is, the **Override** check box in the **Bill-To Info** section is cleared, as shown in the following screenshot.

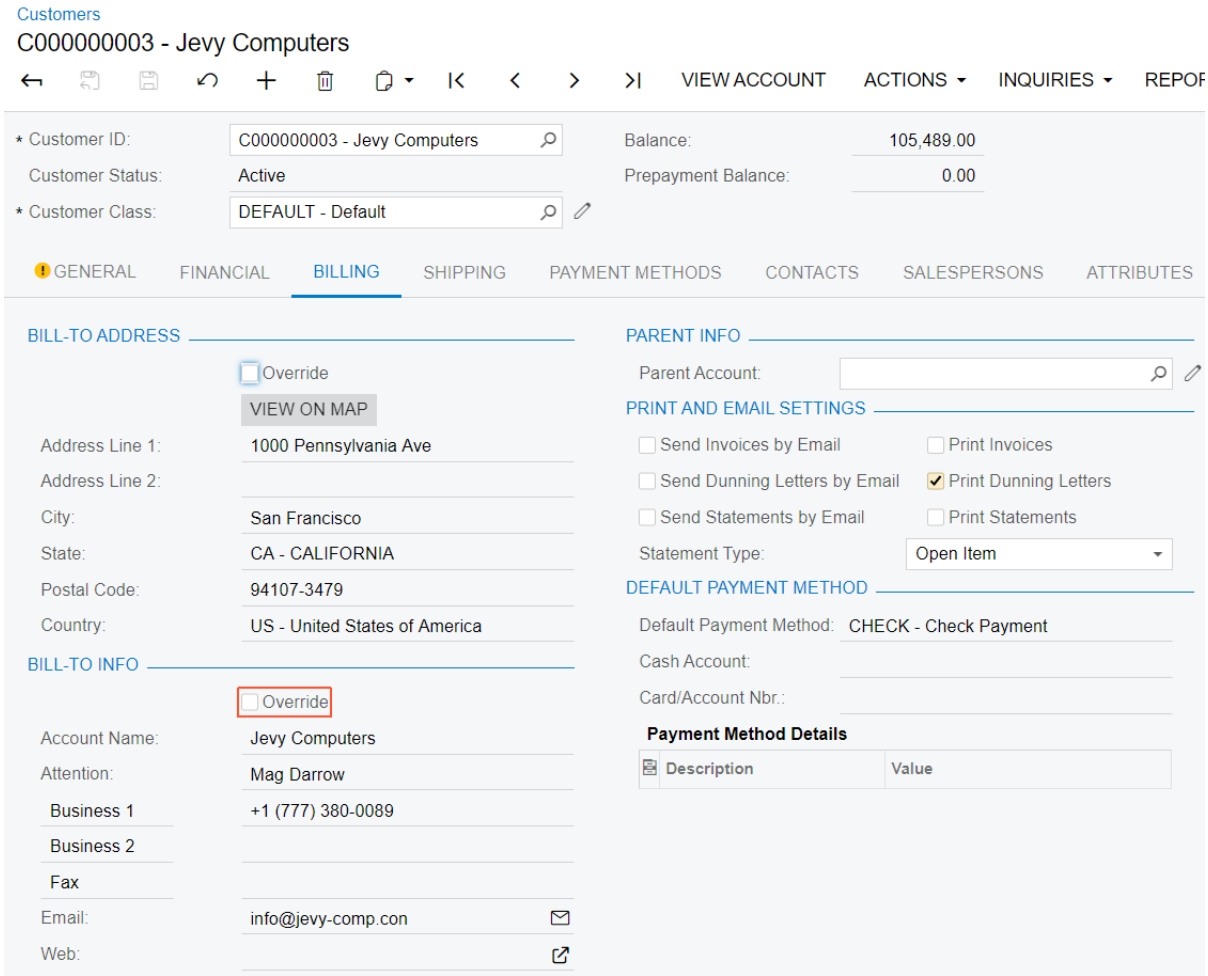

Figure: Customer billing contact

If you want to perform an example of this lesson multiple times with the same data, after you complete the example, you need to restore the customer data on the *Customers* form before you attempt to update the customer once again. To restore the customer data, change the customer class to *Default* in the **Customer Class** box, and clear the **Override** check box in the **Bill-To Info** section of the **Billing** tab.

When you complete an example in this lesson, you can also use any other customer record in the system. If you use a different record, you need to change the email address in the example to the email address of the needed customer.

Example: Using PUT and $filter

This example shows how you can update a customer record by using the contract-based REST API.
You will use the `PUT` HTTP method to update the record. You will specify the `$filter` parameter to find the needed customer record by using the email address.

When the Acumatica ERP contract-based web services receive a `PUT` request that contains at least one `$filter` parameter, Acumatica ERP tries to search for the record by using the specified search value or values. If a record that satisfies the specified conditions is found, Acumatica ERP updates the fields of the record that are specified in the body of the request. If no record that satisfies the specified conditions is found, a new record is created. If multiple records that satisfy the specified conditions are found, Acumatica ERP returns an error.

In the `$filter` parameter, you will specify a field of the `MainContact` child entity. Thus, you will specify the `MainContact` entity in the `$expand` parameter.

Updating a Customer Record

To update a customer record by using the contract-based REST API, do the following:

1. In the Postman collection, add a request with the following settings:
   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/Customer*
   - The following parameters of the request

Parameter	Value
`$filter`	`MainContact/Email eq 'info@jevy-comp.con'`
`$expand`	`MainContact,BillingContact`
`$select`	`CustomerID,CustomerClass,BillingContact/Email`

   - The headers shown below

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

   - The following body of the request

```
{
```

```
 "CustomerClass":{"value":"INTL"},
 "BillingContactOverride":{"value":true},
 "BillingContact":{
  "Email":{"value":"green@jevy-comp.con"},
  "Attention":{"value":"Mr. Jack Green"},
  "JobTitle":{"value":""}
 }
}
```

2. Send the request. If the request is successful, the response contains the `200 OK` status code and includes the list of the requested fields of the customer record in JSON format. The following code example shows a fragment of the response.

```
{
    "id": "12f61a4c-2776-43a8-aca6-a676fe475572",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "BillingContact": {
        ...
    },
    "BillingContactOverride": {
        "value": true
    },
    "CustomerClass": {
        "value": "INTL"
    },
    "CustomerID": {
        "value": "C000000003"
    },
    "custom": {},
    "files": []
}
```

3. Save the request.

Related Links

- *Update a Record*

Additional Information: Removal of a Record

This scenario is outside of the scope of this course but may be useful to some readers.

By the contract-based REST API, you can remove records from Acumatica ERP. For details about how to remove a record, see *Remove a Record by Key Fields*.

Lesson Summary

In this lesson, you have learned how to update an existing customer record and find the record for update by using a non-key field. You have used one request for the update.

Lesson 2.2: Updating the Detail Lines of a Sales Order

The MyStore online store assigns a customer number to each sales order. This number (which is different than the sales order number assigned by Acumatica ERP) is stored in Acumatica ERP in the **Customer Order Nbr.** box of the *Sales Orders* (SO301000) form. Before submitting a sales order for processing, a customer of the online store can select the needed order by using this customer number and then edit and submit the order.
This lesson shows how you can implement this integration scenario by using the contract-based REST API. In this lesson, you will select a sales order in Acumatica ERP by using the value of the customer number of the order and then update the detail lines of the selected sales order.

You will use the `SalesOrder` entity of the *Default/20.200.001* endpoint. This entity is mapped to the Sales Orders form. You will retrieve the needed sales order, update the detail lines of the sales order locally, and submit the updated sales order to Acumatica ERP.

You will identify the detail lines to be updated by using the entity IDs of the detail lines.

The entity ID is a GUID that is assigned to each entity you work with during an Acumatica ERP session. You can obtain the value of the entity ID from the `ID` property of an entity returned from Acumatica ERP.

The records of top-level entities that you retrieve through the contract-based REST API have persistent IDs, which are the values in the `NoteID` column of the corresponding database tables. That is, you can use the value from the `ID` property of a top-level entity returned from Acumatica ERP throughout different sessions with Acumatica ERP. However, if a record does not have a note ID (which could be the case for detail entities, entities that correspond to generic inquiries, or custom entities), this record is assigned the entity ID that is new for each new session. That is, after a new login to Acumatica ERP, you cannot use the entity ID that you received in the previous session to work with the entity.

You will delete a detail line by setting the `delete` system field of the detail entity to `true`.

Lesson Objective

In this lesson, you will learn how to update the detail lines of a document by using the contract-based REST API.

Prerequisites

In the examples of this lesson, you will update the sales order with customer order number *SO248-563-06*; this sales order has been preconfigured. Before you complete the examples of this lesson, on the *Sales Orders* (SO301000) form, make sure that the sales order with this customer order number exists. (In the system, this sales order has the order number *000003*.) Notice that this sales order currently has three detail lines, as shown in the following screenshot.

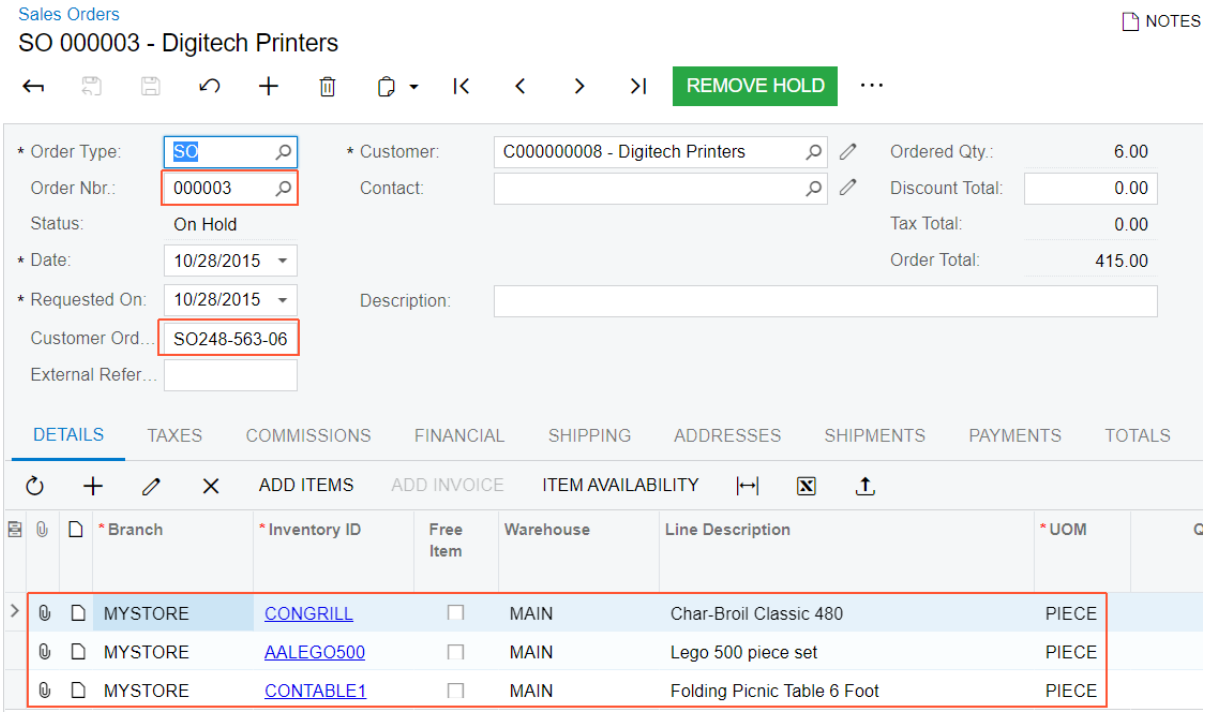

Figure: Sales order to be updated

> If you want to perform an example of this lesson multiple times for the same sales order, you should make sure that the sales order that you are going to update contains the lines that are updated in the example. You can restore the sales order to the state that is described in this topic and shown above (that is, add the deleted lines and update the quantity of the items in the order) or modify the values that are used for the update of detail lines in the code examples.
>
> When you complete the examples in this lesson, you can also use the data of any other sales order with the *On Hold* or *Open* status. If you do this, you need to update the values that are used in the examples accordingly.

Example: Using GET with $filter and PUT

In this example, you will update the detail lines of a sales order record by using the contract-based REST API.

You will retrieve the sales order by using the values of the customer order number (*SO248-563-06*) and the order type (*SO*). Because you do not use the full set of key fields to find the needed record, you will use the `$filter` parameter of the `GET` request to specify the values that are used for the search (instead of specifying the values of key fields in the URL). You will also use the `$expand` and `$select` parameters to request only the key fields and the fields to be updated.

To update the detail lines, you will use the `PUT` request and identify the detail entities by the session IDs of the details returned in the `GET` response. You will delete the detail line for the *CONGRILL* item by setting the `delete` system field of the detail entity to `true`. You will also update the quantity of the *AALEGO500* item.

Updating the Detail Lines of the Sales Order

To update the detail lines of a sales order by using the contract-based REST API, do the following:

1. Retrieve the needed sales order by using the value of the customer order number as follows:

a. In the Postman collection, add a new request with the following settings:

- HTTP method: `GET`
- URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesOrder*
- The following parameters of the request

Parameter	Value
`$expand`	`Details`
`$select`	`OrderNbr,OrderType,Details/InventoryID,Details/WarehouseID`
`$filter`	`OrderType eq 'SO' and CustomerOrder eq 'SO248-563-06'`

- The headers shown below

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

b. Send the request. If the request is successful, the response contains the `200 OK` status code and includes the list of requested fields of the sales order record in JSON format. The following code shows an example of the response.


```
[
    {
        "id": "7313e950-7381-e511-80c0-00155d012302",
        "rowNumber": 1,
        "note": {
            "value": ""
        },
        "CustomerOrder": {
            "value": "SO248-563-06"
        },
        "Details": [
            {
                "id": "08731a64-7381-e511-80c0-00155d012302",
                "rowNumber": 1,
                "note": {
                    "value": ""
                },
                "InventoryID": {
                    "value": "CONGRILL"
                },
                "WarehouseID": {
                    "value": "MAIN"
                },
                "custom": {}
            },
            {
                "id": "f4a7e172-7381-e511-80c0-00155d012302",
                "rowNumber": 2,
```

```
                "note": {
                    "value": ""
                },
                "InventoryID": {
                    "value": "AALEGO500"
                },
                "WarehouseID": {
                    "value": "MAIN"
                },
                "custom": {}
            },
            {
                "id": "091b0d7f-7381-e511-80c0-00155d012302",
                "rowNumber": 3,
                "note": {
                    "value": ""
                },
                "InventoryID": {
                    "value": "CONTABLE1"
                },
                "WarehouseID": {
                    "value": "MAIN"
                },
                "custom": {}
            }
        ],
        "OrderNbr": {
            "value": "000003"
        },
        "OrderType": {
            "value": "SO"
        },
        "custom": {}
    }
]
```


The sales order record in this code example contains three detail lines. You will delete the line for the *CONGRILL* item (which, in the session of this request, has the *08731a64-7381-e511-80c0-00155d012302* identifier) and update the quantity of the *AALEGO500* item (which has the *f4a7e172-7381-e511-80c0-00155d012302* session identifier).

2. Update the sales order record as follows:
   a. In the Postman collection, add a new request with the following settings:
      - HTTP method: `PUT`
      - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesOrder*
      - The following parameters of the request

Parameter	Value
`$expand`	`Details`
`$select`	`OrderType,OrderNbr,Details/OrderQty,Details/InventoryID, OrderedQty,OrderTotal`

- The following body of the request

```
{
 "OrderType":{"value":"SO"},
 "OrderNbr":{"value":"000003"},
 "Hold":{"value":false},
 "Details":[
  {
   "id":"08731a64-7381-e511-80c0-00155d012302",
   "delete":true
  },
  {
   "id":"f4a7e172-7381-e511-80c0-00155d012302",
   "OrderQty":{"value":5.0}
  }
 ]
}
```

  In the body, you specify the IDs of the items from the previous response: the ID of the *CONGRILL* item to delete the item, and the ID of the *AALEGO500* item to update the order quantity.

- The headers shown below

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

b. Send the request. If the request is successful, the response contains the `200 OK` status code and includes the list of requested fields of the updated sales order record in JSON format. The following code shows an example of the response.

```
{
    "id": "7313e950-7381-e511-80c0-00155d012302",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "Details": [
        {
            "id": "f4a7e172-7381-e511-80c0-00155d012302",
            "rowNumber": 1,
            "note": null,
            "InventoryID": {
                "value": "AALEGO500"
            },
            "OrderQty": {
                "value": 5
            },
            "custom": {}
        },
        {
            "id": "091b0d7f-7381-e511-80c0-00155d012302",
            "rowNumber": 2,
            "note": null,
            "InventoryID": {
```

```
                "value": "CONTABLE1"
            },
            "OrderQty": {
                "value": 2
            },
            "custom": {}
        }
    ],
    "Hold": {
        "value": false
    },
    "OrderedQty": {
        "value": 7
    },
    "OrderNbr": {
        "value": "000003"
    },
    "OrderTotal": {
        "value": 295
    },
    "OrderType": {
        "value": "SO"
    },
    "custom": {}
}
```

c. Save the request.

Related Links

- *Update a Record*

Additional Information: Recommendations on Creation or Update of Big Documents

This information applies to cases where you need to create or update documents with a large number of detail lines by using the contract-based REST API. There are two extremes in performing this task:

- You make a single request that contains the whole information you need to pass. In this case there is a risk of an operation timeout.
- You make many requests, each of which contains a single detail line you need to add or alter. In this case the whole task takes a lot of time.

A balanced approach combines these two extremes: you make multiple requests, each of which contains a part of the detail lines you add or alter. The number of detail lines in a single request you select empirically to optimize the performance of the whole task. For example, if you need to send a sales order containing 10,000 detail lines to the server, you can make requests each of which contains 500 detail lines.

Lesson Summary

In this lesson, you have learned how to delete and update the detail lines of a sales order document through the contract-based REST API. You have retrieved the sales order in one request, updated the detail lines of the sales order locally, and submitted the order to Acumatica ERP in another request. You have identified the detail lines for

the update by the entity IDs of the lines. You have used the `delete` system field of the detail entity to remove the detail line.

Part 3: Execution of Actions

In this part of the course, you will perform actions on records in Acumatica ERP through the contract-based REST API. You will update an invoice and invoke the release of the invoice. Because the release of an invoice is a long-running operation, you will monitor the status of the release operation and retrieve the released invoice once the operation is completed.

As a result of completing the lesson of this part, you will know how to execute long-running operations through the contract-based REST API.

Lesson 3.1: Releasing an Invoice

The MyStore online store needs to provide an invoice for each shipped order to the customer so that the customer can review it and confirm the invoice. To make it possible for this invoice to be created and confirmed, the online store passes the data of the customer and the shipped orders to Acumatica ERP and releases the invoice in Acumatica ERP. In Acumatica ERP, the information about whether the invoice was confirmed is reflected by the **Status** value of the invoice on the *Invoices* (SO303000) form. As a result of the method being called, the invoice has the *Open* status.

In this lesson, you will add to the MyStoreIntegration REST application a request that changes the status of an invoice from *On Hold* to *Balanced* and another request that initiates the release of the invoice.

To release an invoice, you will use the `ReleaseSalesInvoice` action of the `SalesInvoice` entity of the *Default/20.200.001* endpoint. Because the release of an invoice is a long-running operation, you will monitor the status of this operation in the method and get the result of processing only after the operation is completed.

In this lesson, you will release a sales order invoice. The processing of other types of documents (such as pro forma invoices) is outside of the scope of this course. For basic information about processing of other types of documents, see *Additional Information: Processing of Pro Forma Invoices*.

Lesson Objective

In this lesson, you will learn how to handle long-running operations, such as the releasing of the invoice, by using the contract-based REST API.

Prerequisites

Before you proceed with any examples in this lesson, clear the **Validate Document Totals on Entry** check box on the *Accounts Receivable Preferences* (AR101000) form. With this check box cleared, the invoice amount does not have to be entered in the **Amount** box on the *Invoices* (SO303000) form, which can be used to validate data during manual entry.

On the *Invoices* form, make sure that the invoice with the *INV000046* reference number exists and that this invoice has the *On Hold* status.

If you want to perform an example of this lesson multiple times, after you complete the example, on the *Invoices* form, you need to create an invoice that has the *On Hold* status. You also need to update the reference number of the invoice in the code example to that of the invoice you created. To create an invoice, you can use the procedure described in *To Prepare an Invoice for a Sales Order* in the documentation or perform a REST API call.

Example: Using POST to Release an Invoice

In this example, by using one `POST` request in the contract-based REST API, you will invoke the release of the *INV000046* invoice, which has the *On Hold* status in the system.

If the request returns the `400 Bad Request`, `401 Unauthorized`, or `500 Internal Server Error` response, the operation has failed.

A response to the `POST` request with the `202 Accepted` status has the `Location` header, which contains a URL that you will use to check the status of the operation by using the `GET` HTTP method. While the status returned by the request is `202 Accepted`, the operation is in progress. You should have a delay between the checks of the status of the operation so that the performance of the application is not impaired.

When the `GET` HTTP method with this URL returns `204 No Content`, the operation is completed. You will then use the `GET` request to retrieve the data of the released invoice from Acumatica ERP.

Releasing an Invoice

To release an invoice by using the contract-based REST API, do the following:

1. Release the invoice from hold as follows:
   a. In the Postman collection, add a new request with the following settings:
      - HTTP method: `PUT`
      - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesInvoice*
      - The following body of the request

```
{
 "Type":{"value":"Invoice"},
 "ReferenceNbr":{"value":"INV000046"},
 "Hold":{"value":false}
}
```

      - The headers shown below

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

   b. Send the request, and make sure the response contains the `200 OK` status.
   c. Save the request.
2. Invoke the release of the invoice as follows:
   a. In the Postman collection, add a new request with the following settings:
      - HTTP method: `POST`
      - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesInvoice/ReleaseSalesInvoice*
      - The following body of the request

```
{
 "entity":{
```

```
  "Type":{"value":"Invoice"},
  "ReferenceNbr":{"value":"INV000046"}
 }
}
```

- The headers shown below

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

b. Send the request, and make sure the response contains the `202 Accepted` status, as shown in the following screenshot. Find the URL in the `Location` header.

Figure: The status and the Location header

c. Save the request.

3. Monitor the status of the release operation by using a request with the following settings:
   - HTTP method: `GET`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesInvoice/ReleaseSalesInvoice/status/389fbc0c-7fc1-42f9-a2d2-822e15959f15* (You copy this URL from the `Location` header of the previous `POST` request.)
   - The headers shown below

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

   Once the `GET` HTTP method returns `204 No Content`, the operation is completed.

4. Retrieve the released invoice by using a request with the following settings:
   - HTTP method: `GET`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/SalesInvoice/Invoice/INV000046*
   - The following parameter of the request

Parameter	Value
`$select`	`ReferenceNbr,Type,Status`

- The headers shown below

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

A successful response contains the `200 OK` status. The status of the invoice is *Open*, as the following code example shows.

```
{
    "id": "5157ccca-3204-e911-b818-00155d408001",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "ReferenceNbr": {
        "value": "INV000046"
    },
    "Status": {
        "value": "Open"
    },
    "Type": {
        "value": "Invoice"
    },
    "custom": {}
}
```

Related Links

- *Execute an Action That Is Present in an Endpoint*

Additional Information: Processing of Pro Forma Invoices

The following scenario, which describes the processing of pro forma invoices, is outside of the scope of this course but may be useful to some readers.

A pro forma invoice is a draft invoice for project billing. You may need to create a pro forma invoice through the REST API if you implement integration of Acumatica ERP projects with external systems. You can find an *example of the creation of a pro forma invoice* in the *REST API Examples* in the documentation.

Lesson Summary

In this lesson, you have learned how to handle long-running operations, such as the releasing of the invoice, by using the contract-based REST API. You have monitored the status of the long-running operation, and when the operation completed, you have retrieved the released invoice from Acumatica ERP.

You have also reviewed how other types of invoices can be processed through the contract-based REST API.

Part 4: Processing of Payments by Credit Cards

Acumatica ERP supports the processing of credit card payments through the Authorize.Net payment gateway.

> ⚠ You can also use other payment gateways for the processing of credit card payments. You can implement a plug-in for the needed payment gateway by yourself or use an existing plug-in available in *https://www.acumatica.com/acumatica-marketplace/*. For details about the creation of custom plug-ins, see *Implementing Plug-Ins for Processing Credit Card Payments* in the documentation.

In this part of the course, you will create a credit card payment method for a customer credit card by using the customer profile ID and the payment profile ID of the customer in Authorize.Net. You will not pass any credit card-specific information to Acumatica ERP.

As a result of completing the lesson of this part, you will learn how to register a customer credit card in Acumatica ERP through the contract-based REST API.

Lesson 4.1: Registering a Customer Credit Card

The MyStore online store needs to process customer payments that are made by using a credit card. To process these payments, the online store uses Acumatica ERP, which supports the processing of credit card payments through the Authorize.Net payment gateway. Acumatica ERP interacts with the payment gateway in PCI DSS–compliant mode.

In this lesson, you will add to the MyStoreIntegration REST application a request that creates a customer payment method, which represents a particular credit card of a customer.

To create this customer payment method, you will submit the customer profile ID and the payment profile ID of the customer in Authorize.Net to the *Customer Payment Methods* (AR303010) form through the contract-based REST API. The customer profile ID and the payment profile ID are used to identify the customer and the customer's card, respectively, across the payment gateway and Acumatica ERP. The payment profile ID, instead of any information specific to the card, is saved to the customer payment method in Acumatica ERP.

You will use the `CustomerPaymentMethod` entity of the *Default/20.200.001* endpoint.

> ⚠ Capturing of the credit card payments is outside of the scope of this course. For basic information about capturing of credit card payments, see *Additional Information: Capturing Credit Card Payments*.

Lesson Objective

In this lesson, you will learn how to create a customer payment method in Acumatica ERP through the contract-based REST API.

Prerequisites

Before you proceed with this lesson, make sure that you have configured the HTTPS connection, as described in *Configuring a Website for HTTPS*.
Additionally, to be able to process credit card payments in the system, you need to do the following before you complete the examples of the lesson:

1. To get a test account at the Authorize.Net payment gateway, create a sandbox account at *https://developer.authorize.net*. (See the link in the following screenshot.)

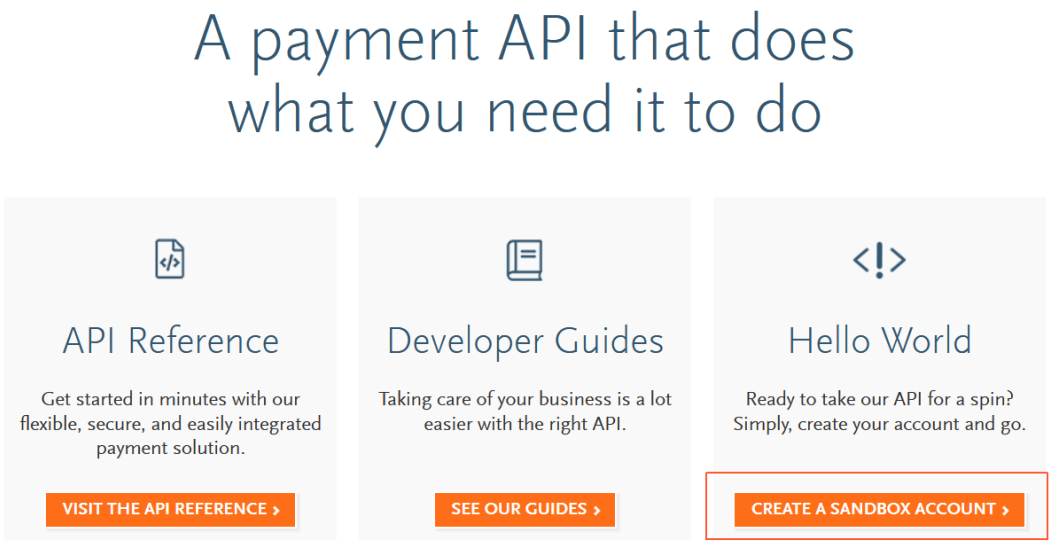

Figure: Create a Sandbox Account link

After you create an account, you will get the credentials to be used in payment processing (API login ID and transaction key). After these credentials are generated, you can view the API login ID and generate a new transaction key on the API Credentials & Keys page of your account on *https://sandbox.authorize.net/* (**Account > Settings > Security Settings > General Security Settings**; see the following screenshots).

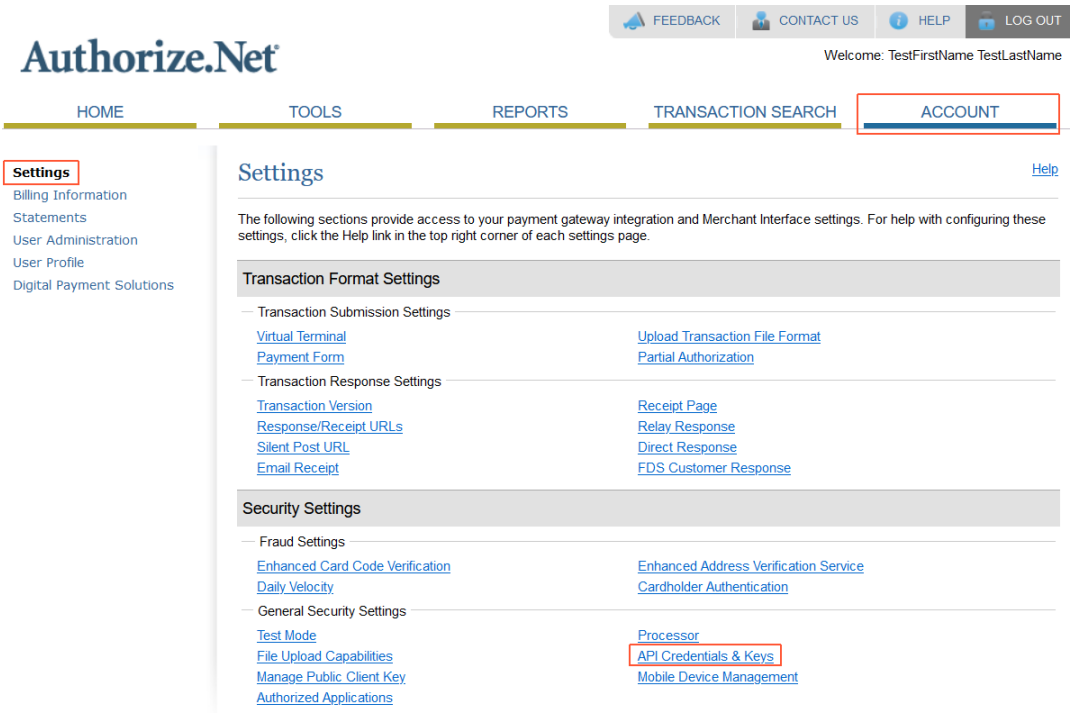

Figure: Sandbox account settings

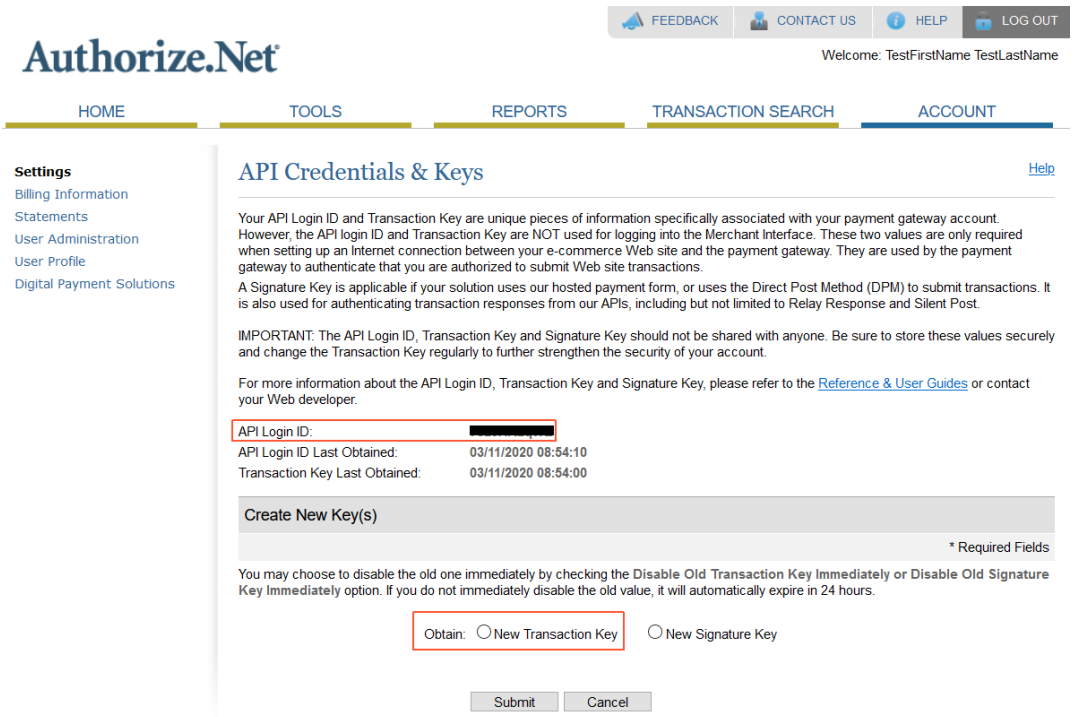

Figure: API Credentials & Keys page

2. Set up the connection with the payment gateway by using your credentials as follows:
   a. On the *Processing Centers* (CA205000) form, select the *AUTHNETUSD* processing center, which has been preconfigured in the system. This processing center is associated with the *VISA* payment method that is used in this example.
   b. To use the payment gateway in test mode (as opposed to live mode), on the **Plug-In Parameters** tab, specify the API login ID of your sandbox account as the value of the *MERCNAME* setting and the transaction key of your sandbox account as the value of the *TRANKEY* setting. Save your changes.

      > ⚠ The payment processing credentials API login ID and transaction key are not the login ID and password that you receive from Authorize.Net to access the Merchant Interface. By using the sandbox account's login ID and password, you can sign in to the sandbox Authorize.Net Merchant Interface (*https://sandbox.authorize.net/*), where you can review the transactions that have been processed at the gateway and manage the registered credit cards.

      > ⚠ For information on settings for live mode, see *Setup of Card Payment Processing* in the documentation.

   c. On the form toolbar, click **Test Credentials** to check the connection settings.

   If the test credentials are accepted by the processing center, the connection to the payment gateway is ready.
3. To process the credit card payments of the customer you will work with in the examples of this lesson by using Authorize.Net, create a customer profile for this customer in your Authorize.Net developer sandbox account as follows:
   a. In the Customer Information Manager tool of Authorize.Net on *https://sandbox.authorize.net/*, add a new customer profile. In the **Customer ID** box of the new customer profile, specify the ID of the customer in Acumatica ERP: *C000000003*, as shown in the following screenshot.

> In your client application, you can create a customer profile by using the API that is provided by Authorize.Net. The integration between your client application and Authorize.Net is outside of the scope of this guide.

b. In the **Card Number** box, type `4007000000027` as the card number (which is the demo Visa card number you can use with Authorize.Net), and in the **Expiration Date** box, specify any date that is later than the current date.

> You can use other test credit card numbers that you have received with your Authorize.Net sandbox account.

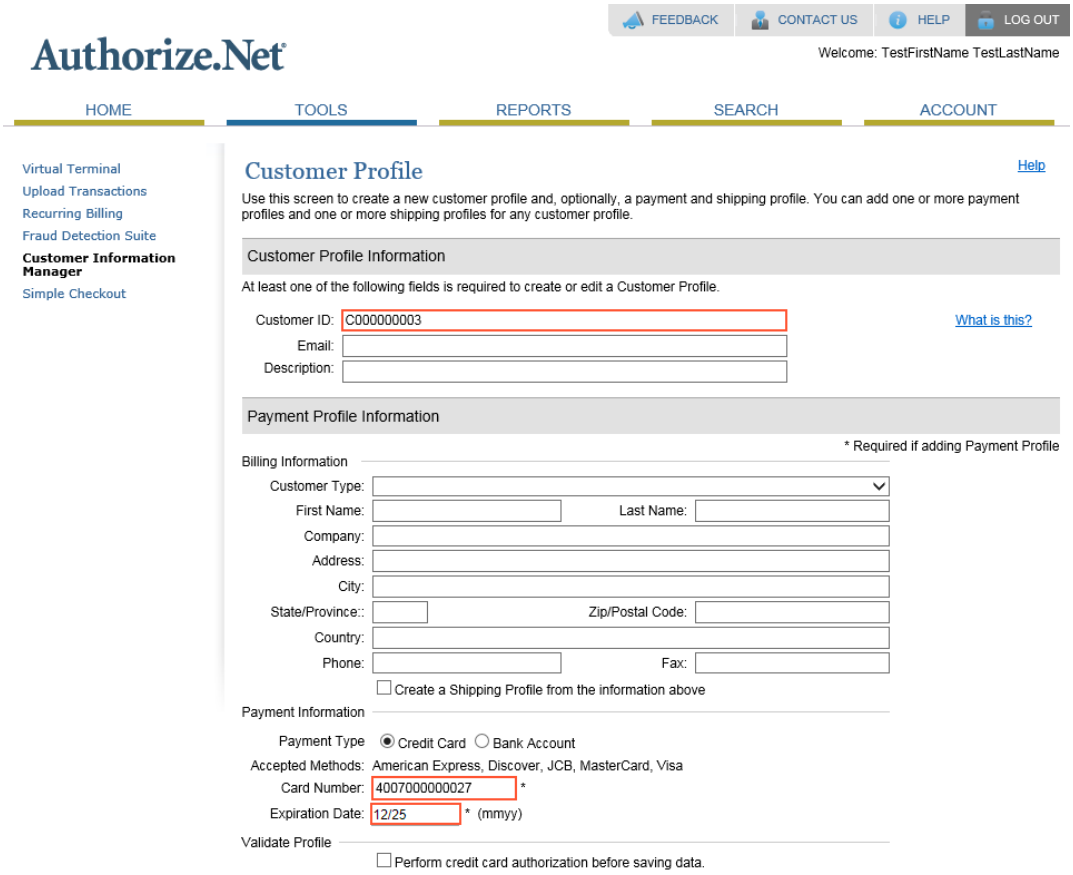

Figure: Customer Information Manager

> Do not use the data of a real credit card in this example! If you do, the money for the testing operations will be charged to this card.

You will use the payment profile ID that has been generated by Authorize.Net as the input value of the method that you create in this lesson.

Example: Using PUT and the CustomerPaymentMethod Entity

In this example, through the contract-based REST API, you will create a customer payment method that corresponds to a customer credit card. To create this customer payment method, you will use the `PUT` HTTP method.

Creating a Customer Payment Method for a Credit Card

To create a customer payment method for a credit card by using the contract-based REST API, do the following:

1. In the Postman collection, add a new request with the following settings:
   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/CustomerPaymentMethod*
   - The following parameters of the request
     - `$expand=Details`
     - `$select=CardAccountNbr,Details/Name`
   - The following body of the request

     ```
     {
      "PaymentMethod":{"value":"VISA"},
      "CustomerID":{"value":"C000000003"},
      "CustomerProfileID":{"value":"1510928386"},
      "CashAccount":{"value":"102050MYST"},
      "Details":[
       {
        "Name":{"value":"Payment Profile ID"},
        "Value":{"value":"1516403307"}
       }
      ]
     }
     ```

     In the body of the request, replace the values of the `CustomerProfileID` field and the `Payment Profile ID` detail with the values that you have received during the registration of the customer payment method in Authorize.Net.
   - The headers shown below

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

2. Send the request. If the request is successful, the response contains the `200 OK` status and includes the created customer payment method in JSON format. The following code shows an example of the response. The `CardAccountNbr` field contains the card identifier in the system, which consists of the payment method ID and the last four numbers of the credit card. This format complies with PCI DSS.

   ```
   {
       "id": "195e8788-947b-eb11-9dd3-9828a61840c3",
       "rowNumber": 1,
       "note": null,
       "CardAccountNbr": {
   ```

```
        "value": "VISA:****-****-****-0027"
    },
    "CashAccount": {
        "value": "102050MYST"
    },
    "CustomerID": {
        "value": "C000000003"
    },
    "CustomerProfileID": {
        "value": "1510928386"
    },
    "Details": [
        {
            "id": "1c5e8788-947b-eb11-9dd3-9828a61840c3",
            "rowNumber": 1,
            "note": null,
            "Name": {
                "value": "CCPID"
            },
            "Value": {
                "value": "1516403307"
            },
            "custom": {}
        }
    ],
    "PaymentMethod": {
        "value": "VISA"
    },
    "custom": {}
}
```


3. Save the request.

Related Links

- *Create a Record*
- *Setup of Card Payment Processing*

Additional Information: Capturing Credit Card Payments

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

To capture a payment through the contract-based REST API, you need to use the `CaptureCreditCardPayment` action available for the `Payment` entity of the *Default/20.200.001* endpoint. Because the capturing of a credit card payment is a long-running operation, you need to monitor the status of the long-running operation before retrieving the result of capturing. (You have learned how to run a long-running operation and monitor its status in *Lesson 3.1: Releasing an Invoice*.)

Lesson Summary

In this lesson, you have learned how to create a customer payment method for a customer credit card. You have used the customer profile ID and the payment profile ID of the customer in Authorize.Net to create a customer payment method in Acumatica ERP. You have also reviewed how you can capture credit card payments.

Part 5: Attachment of Files and Notes

In this part of the guide, you will add notes and attachments to records in Acumatica ERP through the contract-based REST API. You will add a note to a stock item record and attach a file to a stock item record.

As a result of completing the lessons of this part, you will know to which records you can add notes and attachments through the contract-based REST API, and how to add these notes and attachments.

Lesson 5.1: Adding a Note to a Stock Item Record

The administrator of the MyStore company's online store can add notes to inventory items by using the administrator interface of the online store. The online store passes the notes about the stock items to Acumatica ERP by using the contract-based REST API.

In this lesson, you will attach a note to a stock item. You will use the `StockItem` entity of the *Default/20.200.001* endpoint; this entity is mapped to the *Stock Items* (IN202500) form. You will find the needed stock item by using the value of the key field (inventory ID). To add a note, you will use the `note` system field of the `StockItem` entity.

In this lesson, you will not attach notes to detail entities (such as warehouse details of a stock item); this scenario is described briefly in *Additional Information: Addition of Notes to Detail Lines*.

Lesson Objective

In this lesson, you will learn how to add a note to a record through the contract-based REST API.

Example: Using PUT and the note Field

In this example, you will add a note to the *AALEGO500* stock item record by using the `PUT` method and the contract-based REST API.

Adding a Note to a Stock Item Record

To add a note to the stock item record by using the contract-based REST API, do the following:

1. In the Postman collection, add a new request with the following settings:
   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/StockItem*
   - Parameter of the request: `$select=InventoryID`
   - The following body of the request

```
{
 "InventoryID": {"value": "AALEGO500"},
 "note": {"value": "My note"}
}
```

   - The headers shown below

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

2. Send the request. If the request is successful, the response contains the `200 OK` status. The following code shows an example of the response.

```
{
    "id": "cae53ce0-1614-e511-9b82-c86000dddf0b",
    "rowNumber": 1,
    "note": {
        "value": "My note"
    },
    "InventoryID": {
        "value": "AALEGO500"
    },
    "custom": {}
}
```

3. Save the request.

Additional Information: Addition of Notes to Detail Lines

This scenario is outside of the scope of this course but may be useful to some readers.

To add a note to a detail line of a document, you use the same approach that you use to add notes to top-level entities, as described in this lesson. That is, for the contract-based REST API, you specify the value of the `note` system field of the detail entity. For example, to add a note to a warehouse detail line of a stock item, you specify the value of the `note` system field of the `StockItemWarehouseDetail` entity, which is a detail entity of the `StockItem` entity. You identify the detail line that should be updated by the values of key fields or the entity ID.

Lesson Summary

In this lesson, you have learned how to add notes to records in Acumatica ERP by using the contract-based REST API. To add a note, you have used the `note` system field of the entity.

You have also reviewed how to add notes to detail lines of documents.

Lesson 5.2: Attaching a File to a Stock Item Record

The administrator of the MyStore company's online store can attach files to inventory items by using the administrator interface of the online store. The online store passes these files to Acumatica ERP by using the contract-based REST API.

Through the contract-based REST API, files cannot be attached to records that are not available for editing, such as closed AR invoices on the *Invoices and Memos* (AR301000) form. However, the files can be attached to these records through the UI.

In this lesson, you will attach a file to a stock item. You will use the `StockItem` entity of the *Default/20.200.001* endpoint; this entity is mapped to the *Stock Items* (IN202500) form. You will find the needed stock item by using the value of the key field (inventory ID).

Lesson Objective

In this lesson, you will learn how to attach a file to a record through the contract-based REST API.

Example: Using PUT and the Particular Endpoint

In this example, you will attach the `T2MCRO.jpg` file to the *AALEGO500* stock item record by using the `PUT` method. In the URL for the file attachment, you specify the inventory ID (which is the key field of the stock item record) and the name of the file. You have to specify the values of all key fields of the record in the URL. You pass the file in the body of the request.

Attaching a File to a Stock Item Record

To attach a file to the stock item record by using the contract-based REST API, do the following:

1. In the Postman collection, add a new request with the following settings:
   - HTTP method: `PUT`
   - URL: *https://localhost/MyStoreInstance/entity/Default/20.200.001/StockItem/AALEGO500/files/T2MCRO.jpg*
   - Body of the request: The `T2MCRO.jpg` file in binary format
   - The headers shown below

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

2. Send the request. If the request is successful, the response contains the `204 No Content` status.

   On the *Stock Items* (IN202500) form, verify that the file is attached to the *AALEGO500* stock item.

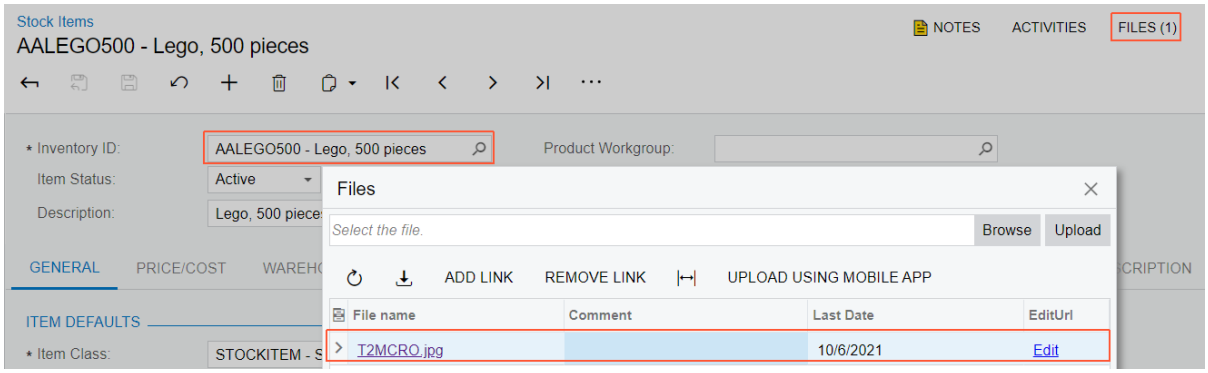

Figure: Attached file

3. Save the request.

Related Links

- *Attach a File to a Record*

Lesson Summary

In this lesson, you have learned how to attach files to records in Acumatica ERP by using the contract-based REST API.

Appendix: Web Integration Scenario Reference

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

- **Creating a stock item with attributes**: *Lesson 1.2: Creating a Stock Item with Attributes*
- **Updating a customer record by using the email address**: *Lesson 2.1: Updating a Customer Account*
- **Updating the detail lines of a sales order** : *Lesson 2.2: Updating the Detail Lines of a Sales Order*
- **Releasing an invoice**: *Lesson 3.1: Releasing an Invoice*
- **Creating a credit card customer payment method**: *Lesson 4.1: Registering a Customer Credit Card*
- **Adding a note to a stock item**: *Lesson 5.1: Adding a Note to a Stock Item Record*
- **Attaching a file to a stock item**: *Lesson 5.2: Attaching a File to a Stock Item Record*

Appendix: Troubleshooting

When I try to create a credit card processing method, I get the error *AR Error #144: Credit card processing error. Processing Center Error: E00040: The record cannot be found.* What should I do?

Make sure that the payment profile ID that you specified when you created a credit card processing method exists for the customer in your Authorize.Net developer sandbox account.

By default, Acumatica ERP deletes the payment profile ID in Authorize.Net if you delete the corresponding payment method in Acumatica ERP. The **Synchronize Deletion** check box is selected by default on the *Processing Centers* (CA205000) form for a processing center you create. If you do not want the payment profile ID to be deleted automatically, clear the **Synchronize Deletion** check box on the Processing Centers form for the needed processing center.

When I try to attach a file to a document, I get the error *Sequence contains no matching element*. What should I do?

Make sure the document that you specified in the request is editable. Through the contract-based APIs, files cannot be attached to records that are not available for editing in Acumatica ERP, such as closed AR invoices on the *Invoices and Memos* (AR301000) form.