EPICOR®

Epicor ERP
REST Implementation
10.2.200

10.2.200

Revision: April 18, 2018 7:04 a.m.

Total pages: 63

sys.ditaval

Contents

Introduction......5
How It Works......6
REST Configuration......7
- Trace REST Calls......7
- REST Root Directory......7
- REST Help Directory......8
  - Epicor REST Help Landing Page......8
  - Tour of the Service List in the REST API Help......9
    - OData List......10
    - Custom Methods List......11
  - Tour of the Business Activity Queries section in the REST API Help......11

Display Data in OData Applications......13
- Link to Browser......13
- Display in Excel Table......13
  - Service Feeds......15
  - BAQ Feeds......16
- Display in Excel PivotTable......17
- Display in Excel PivotChart......18

Epicor Services......20
- OData Datasources......20
- Methods......21

OData Querying......23
- Querying for Collections vs Single Entity......23
- Navigating Relationships......24
- $Parameters Introduction......24
  - Functions and Operators Not Supported......28

REST Examples......29
- Postman......29
  - Executing a Get......29
  - Executing a POST......30
  - Diagnosing an Error......30
- Retrieve Data......31
- Create New Customer......31
- Update Customer......32
- Delete Customer......32
- Update Child Entity......32
- Delete Child User Code; Leave User Type Intact......33
- Run BAQ......33
- Updatable BAQ......34
- Custom Methods......35

Custom Headers....................36
CallSettings Header....................36
SessionInfo Header....................37
OnBehalfOfToken Header....................37
License Header....................38
Trace Flags and Caller Trace Headers....................38
ContextHeader Header....................39
Creating Web Applications Using REST Services....................40
Workshop - Create a Web App with REST....................40
Run the Application....................40
Review the Application Code....................40
Create and Test the Search REST Call in Postman....................42
Generate Code in Postman....................43
Add the Postman Sample Code to the Web App....................43
Display the Search Results on the Web Page....................45
Wire the Search Query to the REST Query....................46
Calling REST with Azure AD token in C#....................47
Create the Visual Studio Project....................47
Add NuGet Package....................47
Add AzureADAuth Class....................47
Add Token Header to the Server Request....................50
Write Code to Call REST API....................51
Calling REST with Azure AD token in JavaScript....................51
WCF Data Services Client Library....................53
SaveChanges Error....................54
Error Handling....................55
Data Security....................56
Basic Authentication....................56
Token Authentication....................56
Single Sign-On....................58
Azure Active Directory Token Authentication....................59
CORS Support....................61

Introduction

Through the REST Application Program Interface (API), these services link applications more easily through internet type connections. The REST architecture considers both data and functions as web resources, and these items are accessible through unique Uniform Resource Identifiers (URIs). A REST server grants access to these web resources, and then REST clients access these resources to display them.

These services use the OData REST Protocol. This protocol adds query capabilities and metadata descriptions that enable OData aware applications to display real time Epicor data. Applications can then communicate with Epicor through REST services in the same way a web browser interacts with a website.

You can use the Odata REST services to connect Microsoft® Excel®, Microsoft® PowerBI®, or other applications that understand OData. For example, link an Epicor business activity query (BAQ) REST Service with Microsoft Excel; Excel then displays real time views of your Epicor data. You can then leverage Excel features, such as PivotTable Report and PivotChart, to further review and evaluate your live data. You can also build custom applications. For example, you could write a new iOS application using obj-c or web application with JavaScript. You could then integrate it with an Epicor REST Service, such as PO Entry, to view and edit purchase orders.

How It Works

Services that utilize this code architecture are called RESTful web services. These services expose a set of resources as URIs, and so various platforms can access this web service to exchange data.

Because the resources are identified as URIs, they provide a global address space for both the resources that contain data and the services which consume them. These resources are updated through a fixed set of HTTP methods:

- **PUT** – Creates a new resource in the database.
- **GET**– Retrieves the current state of a resource from the database.
- **POST** – Updates a changed resource in the database by sending the complete updated object. You can also use the POST method to invoke methods that update records in the database.
- **PATCH** – Similar to POST, this method updates a changed resource in the database as well. However this method only updates data elements that have changed; it does not update the entire object.
- **DELETE** – Removes a resource from the database.

The POST and GET methods are frequently used to invoke methods remotely. If the method will change the state of an object (like saving a record in a database), you use the POST method. When the method does not change the state of an object (like returning a count of open orders), you use the GET method.

For the Epicor application, the REST API uses the PATCH method to update business objects while the POST method is used for invoking methods. By using the PATCH method, the API ensures the data objects update as expected. Then by using the POST method, the API ensures that the business object methods invoke successfully.

Because the resources are available as global URIs, you can access them through a variety of third-party applications and programming languages. Then to further enhance application development, the client installation can transmit data in several formats. For example, a web application written in JavaScript can request data in JavaScript Object Notation (JSON) format, while a C# or Java application can request data in XML format.

Code connections with the URI resources run as self-contained, stateless interactions. The code locates the URI resource within the RESTful web service, pulling the data the resource contains into the receiving application. These transactions are secure, as the services leverage any existing data security defined for the user.

REST Configuration

Use this section to configure REST on your Epicor ERP application server.

Trace REST Calls

You activate server log tracing for REST calls within the AppServer.config file. Your server logs will then contain call entries for the REST services that interact with your database.

To activate REST traces in a server log:

1. Go to your server machine.
2. Navigate to the **Server** folder for your Epicor application. For example, C:\Epicor\2017R\Server
3. Open the **AppServer.config** file.
4. Activate the server tracing and add the following flag:
   - `<add uri="trace://ice/fw/restapi" />`

     The server log will then contain information about REST call processing.
5. You can also add the following trace call:
   - `<add uri="trace://ice/fw/servicecaller" />`

     This shows what service was loaded when a call started its run.

REST Root Directory

The Epicor REST services are grouped together under a top level, or parent, Uniform Resource Locator (URL). This URL is the **REST Service Root URL**, and it is located on your server installation.

Before you can use a REST service, you first locate its unique URL value. You should also learn which actions and parameters you can use with the service.

The directory path uses the following format:

- **https://[MyAppServerHost]/[MyAppServerInstance]/api/**

For example, the root path could be:

- **https://EpicorServer/ERP101600/api/v1**

Note You are prompted for credentials when accessing api via browser. Enter a standard Epicor ERP username and password.

REST Help Directory

The Epicor REST services also have an interactive website hosted help system with each deployment of the Epicor REST services.

Use this website to explore the available services and also test these services in your web browser. Before you set up your third party application to consume Epicor data, review the information available on this web site.

The help directory path uses the following format:

- **https://[MyAppServerHost]/[MyAppServerInstance]/api/Help/**

For example, the directory path could be:

- **https://EpicorServer/ERP101600/api/help**

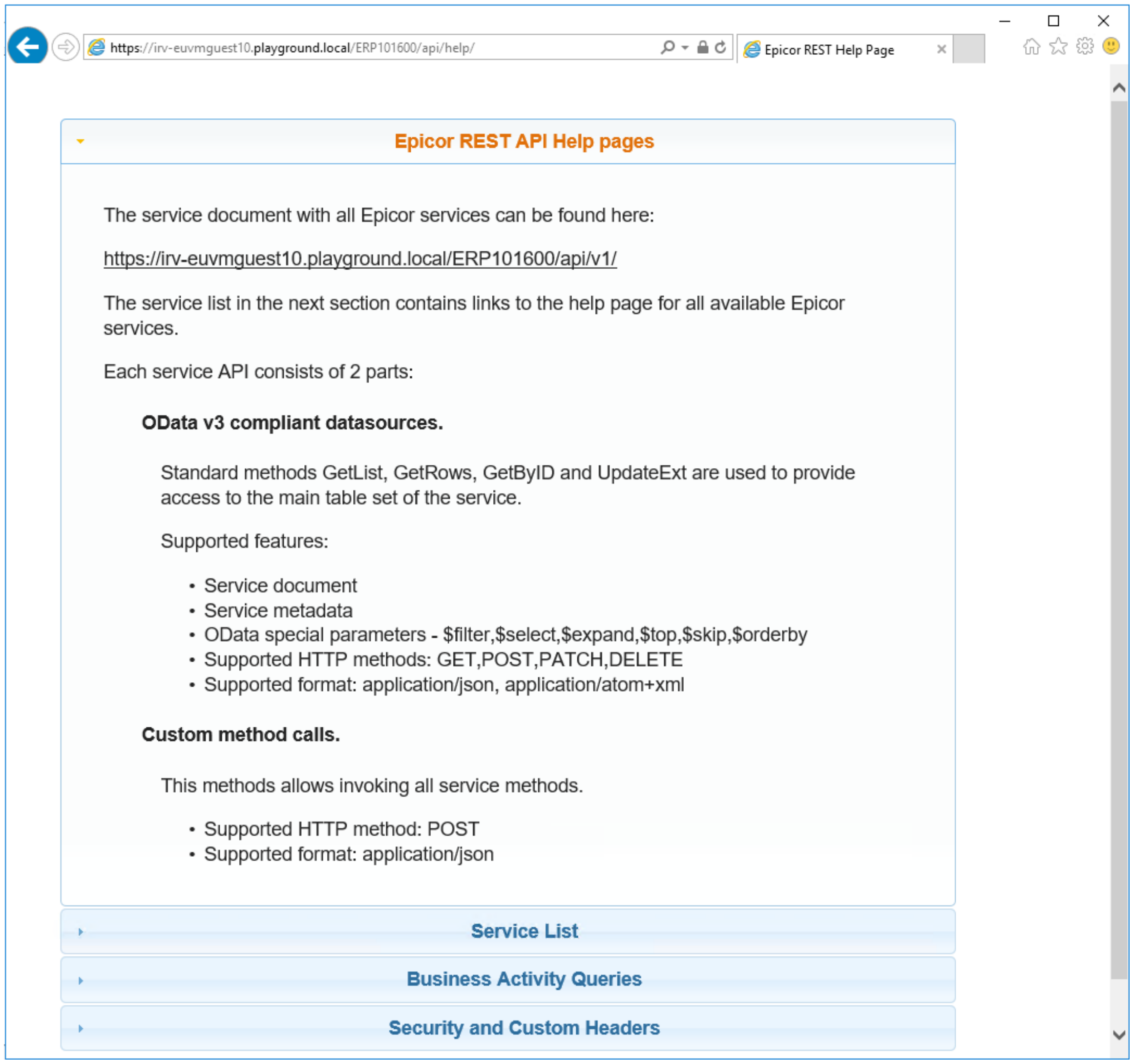
When prompted, use your Epicor credentials to access Epicor API Help.

Epicor REST Help Landing Page

This start page displays the contents of the Epicor REST Help.

The Epicor REST Help landing page has 4 sections as shown on the screenshot below:

- **Epicor REST API Help pages** - the first section expanded by default - provides general description of Help structure.
- **Service List** - toggle the header of this section to display the list of all services (BOs). These mirror the existing services deployed to the Server. If you have had a custom made service for your deployment, it will appear here. Use Search to locate the service you need.
- **Business Activity Queries** - All BAQs on your system will appear here. If you have company specific BAQs, you may need to enter a Company ID.
- **Security and Custom Headers** - to ease integration with other apps, ERP 10 has a security token server. Custom headers can be sent to the server to tell the server to override defaults (e.g. - Last logged in Company, Plant).


https://irv-euvmguest10.playground.local/ERP101600/api/help/
Epicor REST Help Page

Epicor REST API Help pages

The service document with all Epicor services can be found here:

https://irv-euvmguest10.playground.local/ERP101600/api/v1/

The service list in the next section contains links to the help page for all available Epicor services.

Each service API consists of 2 parts:

OData v3 compliant datasources.

Standard methods GetList, GetRows, GetByID and UpdateExt are used to provide access to the main table set of the service.

Supported features:

- Service document
- Service metadata
- OData special parameters - $filter,$select,$expand,$top,$skip,$orderby
- Supported HTTP methods: GET,POST,PATCH,DELETE
- Supported format: application/json, application/atom+xml

Custom method calls.

This methods allows invoking all service methods.

- Supported HTTP method: POST
- Supported format: application/json

Service List

Business Activity Queries

Security and Custom Headers

Tour of the Service List in the REST API Help

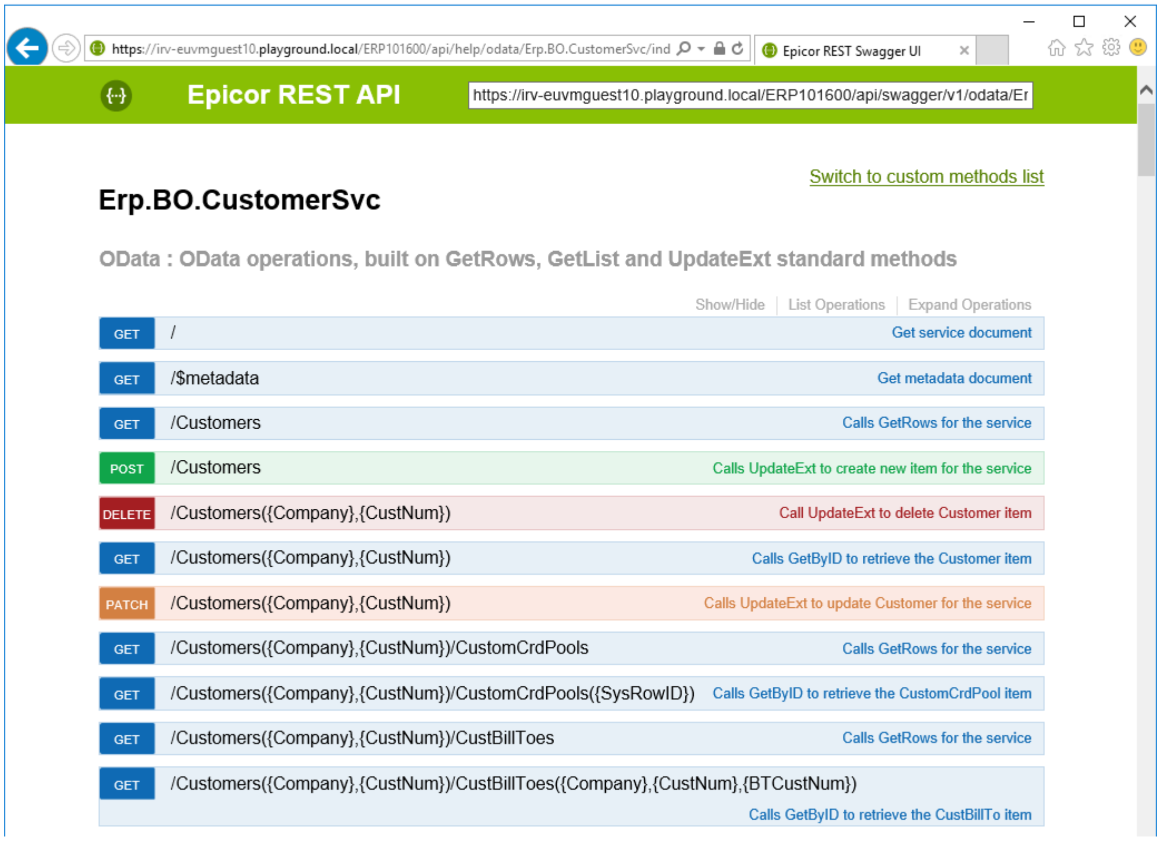
This section shows you how to locate help for the **Customer** business object and describes the data structure of its services.

Follow the same process to locate services for other business objects:

1. Launch your web browser.
2. Navigate to the **REST Help URL**.
   For example: **https://EpicorServer/ERP101600/api/help**. When prompted, use your Epicor credentials.
3. Expand the **Service List** section.
4. Search for a service by entering its name. In this example, you enter **Customer**.
   All the services that contain "Customer" display in the drop-down list.
5. Click on the **CustomerSvc** name.

   Note The help page is built dynamically just like the ERP 10 services are built up and take a moment to cache after a restart of IIS. Large services on a slow server can take more time to download.

A new page displays that shows the available REST resources and actions available within the **Customer** service.



https://irv-euvmguest10.playground.local/ERP101600/api/help/odata/Erp.BO.CustomerSvc/ind

Epicor REST Swagger UI

Epicor REST API https://irv-euvmguest10.playground.local/ERP101600/api/swagger/v1/odata/Er

Switch to custom methods list

Erp.BO.CustomerSvc

OData : OData operations, built on GetRows, GetList and UpdateExt standard methods

Show/Hide | List Operations | Expand Operations

GET	/	Get service document
GET	/$metadata	Get metadata document
GET	/Customers	Calls GetRows for the service
POST	/Customers	Calls UpdateExt to create new item for the service
DELETE	/Customers({Company},{CustNum})	Call UpdateExt to delete Customer item
GET	/Customers({Company},{CustNum})	Calls GetByID to retrieve the Customer item
PATCH	/Customers({Company},{CustNum})	Calls UpdateExt to update Customer for the service
GET	/Customers({Company},{CustNum})/CustomCrdPools	Calls GetRows for the service
GET	/Customers({Company},{CustNum})/CustomCrdPools({SysRowID})	Calls GetByID to retrieve the CustomCrdPool item
GET	/Customers({Company},{CustNum})/CustBillToes	Calls GetRows for the service
GET	/Customers({Company},{CustNum})/CustBillToes({Company},{CustNum},{BTCustNum})	Calls GetByID to retrieve the CustBillTo item

You can now explore the available resources. See the sections on OData and Custom methods for details:

OData List

OData as a standard has different abilities. The first is describing what is in the service.

1. On the **Customer** service (Erp.BO.CustomerSvc) help page, click **/$metadata**.
   To access the service list for **Customer** refer to instructions in section Tour of the Service List in the REST API Help

2. Click **Try it out!**
   Note the Response Body, where all the meta about the service is displayed:
   - EntitySet (Table);
   - FunctionImport (Method);
   - AssociationSet (Table Relationships);
   - EntityType / Key / Property (Table / Keys / Fields).

3. Click **Hide Response** in the **Response Messages** section.
   $metadata section collapses.

4. Click GET / **Customers**.
   In the **Response Class** section **Model Schema** view is displayed by default. Click **Model**. Note:
   - CustomerRow description;

- All the 'XML Comments' about the field are displayed. These are generated from entries in the SDK Tool - **Service Designer**.

5. Click **Model Schema**.
   In the response, note JSON description of the fields useful for tools and displaying headers in programs.

6. Click **Try it out!** Note the sections:
   - Request URL - The URL you can type in your browser to get back information;
   - Response Body - The data response from the URL;
   - Curl - A command line tool to execute URLs;
   - Response Code - Standard web responses codes.

     Example Code 200 = success, 404 = page not found, etc.

   - Response Headers - Any BPM form information would be returned in the Context Header response.

Custom Methods List

Custom methods do not support OData. These are all the adhoc custom methods exposed by the service. These can only be used with the POST verb in code.

1. In the **custom methods list** for **Customer,** click **GetByCustID**.

2. Click on **input params** on the right to set them as input values on the left.

3. Change the **input** to:
```
{
"custID": "addison",
"withShipTo": true
}
```
   Note **Addison** is a sample customer ID from the Epicor training database. If you use a different database, take any existing customer ID from your database to use in this input parameters description.

4. Click **Try it out!**
   Note the **Response Body** results for **Addison**.

Tour of the Business Activity Queries section in the REST API Help

BAQs are exposed directly through the REST APIs.

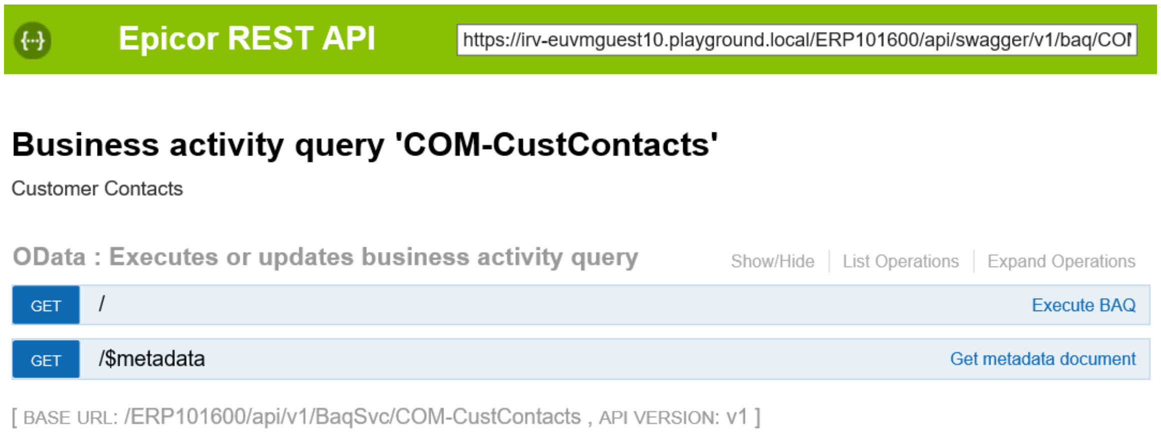
Use a similar process to locate REST help for a business activity query (BAQ) service:

1. Launch your web browser.

2. Navigate to the **REST Help URL**.
   For example: **https://EpicorServer/ERP101600/api/help**. When prompted, use your Epicor credentials.

3. Expand the **Business Activity Queries** section.

4. In the **QueryID** field, enter **'COM'**, in the drop-down list, select **COM-CustContacts** and click the **Get Help** button.
   A new page displays that shows the available REST resources and actions available within this BAQ service.



Business activity query 'COM-CustContacts'

Customer Contacts

OData : Executes or updates business activity query

/

/$metadata

[BASE URL: /ERP101600/api/v1/BaqSvc/COM-CustContacts , API VERSION: v1]

5. Click **/** to expand the section.

6. Click **Try it out!**
   The results of the BAQ are returned.

7. In the **Parameters** section select **$filter** field and enter **CustCnt_Country eq 'USA'**. Click **Try it out!**
   The results of the BAQ are specific to records for the country of USA.

8. You can use this web page to dynamically test the service.

Display Data in OData Applications

Any application that can connect to OData or generate REST services can access data from an Epicor database through REST services.

This section describes how you can display data through two example applications – your internet browser and Microsoft® Excel®. These examples represent just a couple of applications you can use. Review the documentation in other OData aware applications to learn how to connect and display Epicor data through them.

Link to Browser

You can pull data from REST services for display in your internet browser.

Verify your Secure Sockets Layer (SSL) is currently running on your server. When you run a **HTTPS** connection you ensure that the certificates are in place for both REST services and the OData connection. Once you have these items set up, you can browse or call REST services as you need.

As described previously, use the following URL syntax to locate the root folder for the Epicor REST services:

- **https://[MyAppServerHost]/[MyAppServerInstance]/api/[ServiceName]/**

If you want to display data from the SalesOrder service in your web browser, you would enter the Service Root folder, followed by the Service Name:

- **https://EpicorServer/ERP101600/api/v1/Erp.BO.SalesOrderSvc/**

This URL returns an Atom+XML service document which lists the specific resources you can use with this service. For a business object service, you will see one resource for each method and one resource for each type of data entity. For each BAQ service, you will see one service root and one resource. You enter this URL syntax to access a REST resource within a business object service:

- **https://[MyAppServer]/[MyShare]/api/[VersionNumber]/[ServiceName]/[ResourceName]/**

For example, to display all the current sales orders within your web browser, you would enter:

- **https://EpicorServer/ERP101600/api/v1/Erp.BO.SalesOrderSvc/SalesOrders**

This example retrieves data from the SalesOrder business object. It calls the **Erp.Bo.SalesOrderSvc** service and uses the **GetRows** method. If you wish to return data using an abbreviated list format, you call the **GetList** method instead. To do this, use the List resource:

- **https://EpicorServer/ERP101600/api/v1/Erp.BO.SalesOrderSvc/List**

Note that when you browse to the URL, you are prompted with a security dialog box. Enter your manager account in this window.

Display in Excel Table

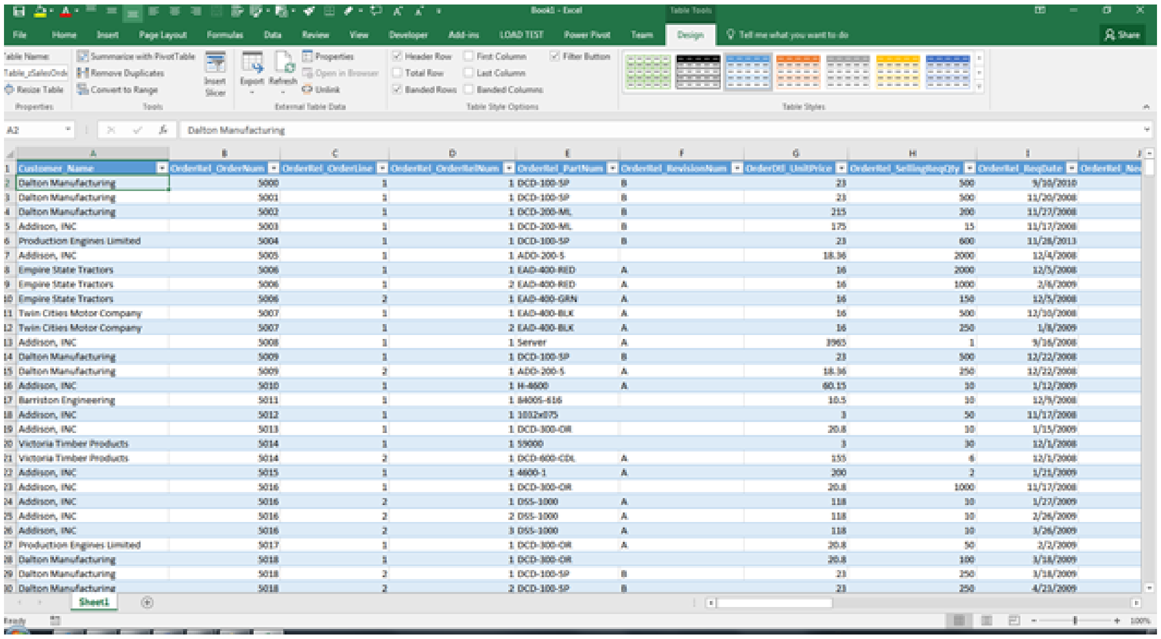
By natively supporting OData, Epicor 10 services allow for easy consumption from tools that leverage the protocol. You can have at anytime refreshable spreadsheet against a live data source. Build pivot tables, charts against the feed for quick reports. Do the following to use REST services to pull Epicor data within an Excel spreadsheet. You can then format this live data display as you need.

Important Odata v3 is supported by MS Excel 2013 or higher.

If you use Excel 2010, you need to download and instal the PowerPivot add-in for Excel. This download is available on the https://www.microsoft.com/en-us/download/confirmation.aspx?id=29074 web page. Some of the steps you follow will be different from what is documented in this section.

1. Launch **Microsoft Excel**.
2. Select the **Data** tab.
3. Using the **Menu Ribbon**, select the **From Other Sources** option.
4. From the list, select the **From OData Feed**. The **Data Connection Wizard** displays.
5. For the Location of the data feed, either enter or click the **Browse_** button to define the REST data feed you wish to connect with this Excel spreadsheet. For example:
   - **https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/**
   - **https://appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/**
6. Now enter **User Name** and **Password**.
7. Click **Next**. The **Select Tables** wizard step displays.
8. Select the table(s) you want to connect to this spreadsheet.
9. Click **Next**.
10. Enter a **File Name** for the **Data Connection** file. This File contains the settings you defined through the **Data Connection Wizard**.
11. Enter a concise **Description** that explains the purpose for the external database connection.
12. Click **Finish**. The **Import Data** dialog box displays.
13. Select the **Table** option to display the information in a grid.
14. Click **OK**.

The external data displays in your spreadsheet. You can then use the table design tools to format how this data displays; review your Microsoft® Excel® documentation for more information. The next illustration shows real time Epicor data displaying within an Excel spreadsheet.

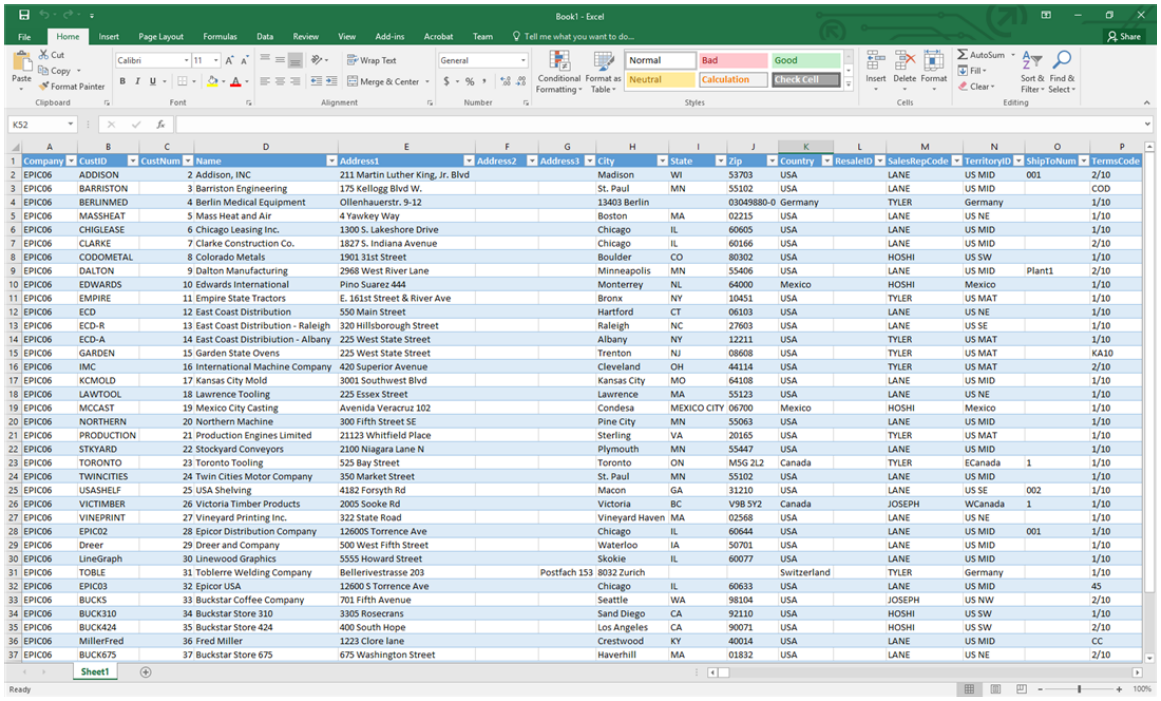


Customer_Name	OrderRel_OrderNum	OrderRel_OrderLine	OrderRel_OrderRelNum	OrderRel_PartNum	OrderRel_RevisionNum	OrderDtl_UnitPrice	OrderRel_SellingReqQty	OrderRel_ReqDate	OrderRel_Ne
Dalton Manufacturing	5000	1	1	DCD-100-SP	B	23	500	9/10/2010	
Dalton Manufacturing	5001	1	1	DCD-100-SP	B	23	500	11/20/2008	
Dalton Manufacturing	5002	1	1	DCD-200-ML	B	215	200	11/27/2008	
Addison, INC	5003	1	1	DCD-200-ML	B	175	15	11/17/2008	
Production Engines Limited	5004	1	1	DCD-100-SP	B	23	600	11/28/2013	
Addison, INC	5005	1	1	ADD-200-S		18.36	2000	12/4/2008	
Empire State Tractors	5006	1	1	EAD-400-RED	A	16	2000	12/5/2008	
Empire State Tractors	5006	1	2	EAD-400-RED	A	16	1000	2/6/2009	
Empire State Tractors	5006	2	1	EAD-400-GRN	A	16	150	12/5/2008	
Twin Cities Motor Company	5007	1	1	EAD-400-BLK	A	16	500	12/10/2008	
Twin Cities Motor Company	5007	1	2	EAD-400-BLK	A	16	250	1/8/2009	
Addison, INC	5008	1	1	Server	A	3965	1	9/16/2008	
Dalton Manufacturing	5009	1	1	DCD-100-SP	B	23	500	12/22/2008	
Dalton Manufacturing	5009	2	1	ADD-200-S	A	18.36	250	12/22/2008	
Addison, INC	5010	1	1	H-4600	A	60.15	10	1/12/2009	
Barriston Engineering	5011	1	1	84005-616		10.5	10	12/9/2008	
Addison, INC	5012	1	1	1032x075		3	50	11/17/2008	
Addison, INC	5013	1	1	DCD-300-OR		20.8	10	1/15/2009	
Victoria Timber Products	5014	1	1	59000		3	30	12/1/2008	
Victoria Timber Products	5014	2	1	DCD-600-CDL	A	155	6	12/1/2008	
Addison, INC	5015	1	1	4600-1	A	200	2	1/21/2009	
Addison, INC	5016	1	1	DCD-300-OR		20.8	1000	11/17/2008	
Addison, INC	5016	2	1	DSS-1000	A	118	10	1/27/2009	
Addison, INC	5016	2	2	DSS-1000	A	118	10	2/26/2009	
Addison, INC	5016	2	3	DSS-1000	A	118	10	3/26/2009	
Production Engines Limited	5017	1	1	DCD-300-OR	A	20.8	50	2/2/2009	
Dalton Manufacturing	5018	1	1	DCD-300-OR		20.8	100	3/18/2009	
Dalton Manufacturing	5018	2	1	DCD-100-SP	B	23	250	3/18/2009	
Dalton Manufacturing	5018	2	2	DCD-100-SP	B	23	250	4/23/2009	

Service Feeds

1. Open a blank worksheet in Excel.
2. Click **Data / From Other Sources / From OData Data Feed**.
3. Paste in the Address / Request URL from the Customer example above:
   https://EpicorSI/Erp10/api/v1/Erp.BO.CustomerSvc/Customers
4. Select Use this name and password. Enter your Epicor credentials.
5. Click **Next**.
6. Check the **Customer** table.
7. Click **Finish**.
8. Click OK for **Import Data** into a **Table**.
9. Note the Customer List retrieved in Excel.
10. Click on **Data**.
11. Click on **Refresh All**.

The external data displays in your spreadsheet. The next illustration shows real time Epicor data displaying within an Excel spreadsheet.

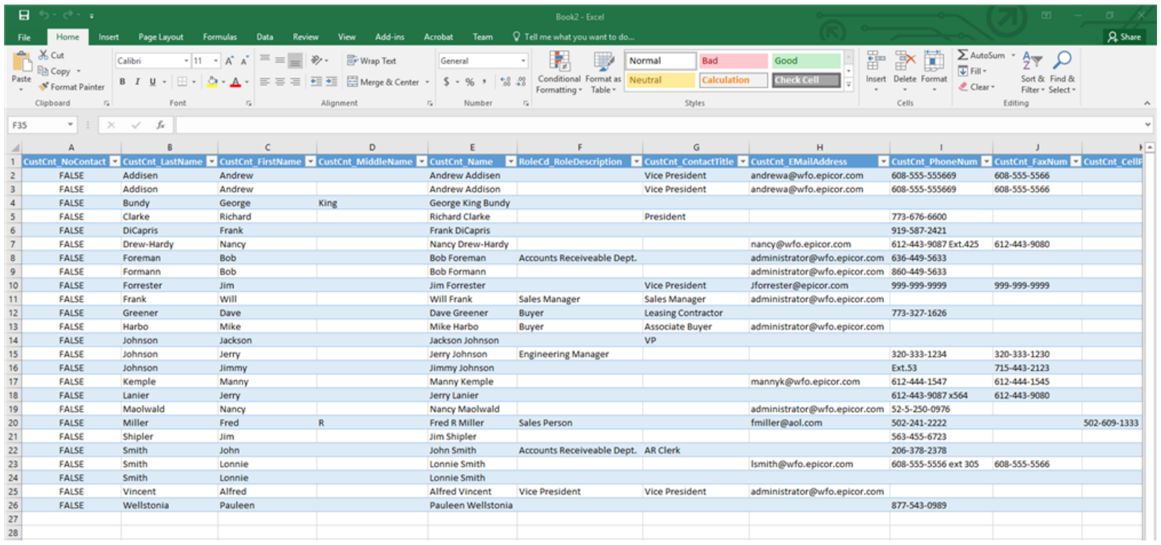


Company	CustID	CustNum	Name	Address1	Address2	Address3	City	State	Zip	Country	ResaleID	SalesRepCode	TerritoryID	ShipToNum	TermsCode
EPIC06	ADDISON	2	Addison, INC	211 Martin Luther King, Jr. Blvd			Madison	WI	53703	USA		LANE	US MID	001	2/10
EPIC06	BARRISTON	3	Barriston Engineering	175 Kellogg Blvd W.			St. Paul	MN	55102	USA		LANE	US MID		COD
EPIC06	BERLINMED	4	Berlin Medical Equipment	Ollenhauerstr. 9-12			13403 Berlin		03049880-0	Germany		TYLER	Germany		1/10
EPIC06	MASSHEAT	5	Mass Heat and Air	4 Yawkey Way			Boston	MA	02215	USA		LANE	US NE		1/10
EPIC06	CHIGLEASE	6	Chicago Leasing Inc.	1300 S. Lakeshore Drive			Chicago	IL	60605	USA		LANE	US MID		1/10
EPIC06	CLARKE	7	Clarke Construction Co.	1827 S. Indiana Avenue			Chicago	IL	60166	USA		LANE	US MID		2/10
EPIC06	CODOMETAL	8	Colorado Metals	1901 31st Street			Boulder	CO	80302	USA		HOSHI	US SW		1/10
EPIC06	DALTON	9	Dalton Manufacturing	2968 West River Lane			Minneapolis	MN	55406	USA		LANE	US MID	Plant1	2/10
EPIC06	EDWARDS	10	Edwards International	Pino Suarez 444			Monterrey	NL	64000	Mexico		HOSHI	Mexico		1/10
EPIC06	EMPIRE	11	Empire State Tractors	E. 161st Street & River Ave			Bronx	NY	10451	USA		TYLER	US MAT		1/10
EPIC06	ECD	12	East Coast Distribution	550 Main Street			Hartford	CT	06103	USA		LANE	US NE		1/10
EPIC06	ECD-R	13	East Coast Distribution - Raleigh	320 Hillsborough Street			Raleigh	NC	27603	USA		LANE	US SE		1/10
EPIC06	ECD-A	14	East Coast Distribiution - Albany	225 West State Street			Albany	NY	12211	USA		TYLER	US MAT		1/10
EPIC06	GARDEN	15	Garden State Ovens	225 West State Street			Trenton	NJ	08608	USA		TYLER	US MAT		KA10
EPIC06	IMC	16	International Machine Company	420 Superior Avenue			Cleveland	OH	44114	USA		TYLER	US MAT		2/10
EPIC06	KCMOLD	17	Kansas City Mold	3001 Southwest Blvd			Kansas City	MO	64108	USA		LANE	US MID		1/10
EPIC06	LAWTOOL	18	Lawrence Tooling	225 Essex Street			Lawrence	MA	55123	USA		LANE	US NE		1/10
EPIC06	MCCAST	19	Mexico City Casting	Avenida Veracruz 102			Condesa	MEXICO CITY	06700	Mexico		HOSHI	Mexico		1/10
EPIC06	NORTHERN	20	Northern Machine	300 Fifth Street SE			Pine City	MN	55063	USA		LANE	US MID		1/10
EPIC06	PRODUCTION	21	Production Engines Limited	21123 Whitfield Place			Sterling	VA	20165	USA		TYLER	US MAT		1/10
EPIC06	STKYARD	22	Stockyard Conveyors	2100 Niagara Lane N			Plymouth	MN	55447	USA		LANE	US MID		1/10
EPIC06	TORONTO	23	Toronto Tooling	525 Bay Street			Toronto	ON	M5G 2L2	Canada		TYLER	ECanada	1	1/10
EPIC06	TWINCITIES	24	Twin Cities Motor Company	350 Market Street			St. Paul	MN	55102	USA		LANE	US MID		1/10
EPIC06	USASHELF	25	USA Shelving	4182 Forsyth Rd			Macon	GA	31210	USA		LANE	US SE	002	1/10
EPIC06	VICTIMBER	26	Victoria Timber Products	2005 Sooke Rd			Victoria	BC	V9B 5Y2	Canada		JOSEPH	WCanada	1	1/10
EPIC06	VINEPRINT	27	Vineyard Printing Inc.	322 State Road			Vineyard Haven	MA	02568	USA		LANE	US NE		1/10
EPIC06	EPIC02	28	Epicor Distribution Company	12600S Torrence Ave			Chicago	IL	60644	USA		LANE	US MID	001	1/10
EPIC06	Dreer	29	Dreer and Company	500 West Fifth Street			Waterloo	IA	50701	USA		LANE	US MID		1/10
EPIC06	LineGraph	30	Linewood Graphics	5555 Howard Street			Skokie	IL	60077	USA		LANE	US MID		1/10
EPIC06	TOBLE	31	Toblerre Welding Company	Bellerivestrasse 203		Postfach 153	8032 Zurich			Switzerland		TYLER	Germany		1/10
EPIC06	EPIC03	32	Epicor USA	12600 S Torrence Ave			Chicago	IL	60633	USA		LANE	US MID		45
EPIC06	BUCKS	33	Buckstar Coffee Company	701 Fifth Avenue			Seattle	WA	98104	USA		JOSEPH	US NW		2/10
EPIC06	BUCK310	34	Buckstar Store 310	3305 Rosecrans			Sand Diego	CA	92110	USA		HOSHI	US SW		1/10
EPIC06	BUCK424	35	Buckstar Store 424	400 South Hope			Los Angeles	CA	90071	USA		HOSHI	US SW		2/10
EPIC06	MillerFred	36	Fred Miller	1223 Clore lane			Crestwood	KY	40014	USA		LANE	US MID		CC
EPIC06	BUCK675	37	Buckstar Store 675	675 Washington Street			Haverhill	MA	01832	USA		LANE	US NE		2/10

BAQ Feeds

1. Open a blank worksheet in Excel.
2. Click **Data / From Other Sources / From OData Feed**.
3. Paste in the Address / Request URL from the Customer example above: **https://EpicorSI/Erp10/api/v1/BaqSvc/COM-CustContacts/**
4. Select use this name and password. Enter your Epicor credentials.
5. Click **Next**.
6. Check the **COM-CustContacts** table.
7. Click **Finish**.
8. Click OK for **Import Data** into a **Table**.
9. Note the Customer Contact List retrieved in Excel.
10. Click on **Data**.
11. Click on **Refresh All**.

The external data displays in your spreadsheet. The next illustration shows real time Epicor data displaying within an Excel spreadsheet



	A	B	C	D	E	F	G	H	I	J	
1	CustCnt_NoContact	CustCnt_LastName	CustCnt_FirstName	CustCnt_MiddleName	CustCnt_Name	RoleCd_RoleDescription	CustCnt_ContactTitle	CustCnt_EMailAddress	CustCnt_PhoneNum	CustCnt_FaxNum	CustCnt_CellP
2	FALSE	Addisen	Andrew		Andrew Addisen		Vice President	andrewa@wfo.epicor.com	608-555-555669	608-555-5566	
3	FALSE	Addison	Andrew		Andrew Addison		Vice President	andrewa@wfo.epicor.com	608-555-555669	608-555-5566	
4	FALSE	Bundy	George	King	George King Bundy						
5	FALSE	Clarke	Richard		Richard Clarke		President		773-676-6600		
6	FALSE	DiCapris	Frank		Frank DiCapris				919-587-2421		
7	FALSE	Drew-Hardy	Nancy		Nancy Drew-Hardy			nancy@wfo.epicor.com	612-443-9087 Ext.425	612-443-9080	
8	FALSE	Foreman	Bob		Bob Foreman	Accounts Receiveable Dept.		administrator@wfo.epicor.com	636-449-5633		
9	FALSE	Formann	Bob		Bob Formann			administrator@wfo.epicor.com	860-449-5633		
10	FALSE	Forrester	Jim		Jim Forrester		Vice President	Jforrester@epicor.com	999-999-9999	999-999-9999	
11	FALSE	Frank	Will		Will Frank	Sales Manager	Sales Manager	administrator@wfo.epicor.com			
12	FALSE	Greener	Dave		Dave Greener	Buyer	Leasing Contractor		773-327-1626		
13	FALSE	Harbo	Mike		Mike Harbo	Buyer	Associate Buyer	administrator@wfo.epicor.com			
14	FALSE	Johnson	Jackson		Jackson Johnson		VP				
15	FALSE	Johnson	Jerry		Jerry Johnson	Engineering Manager			320-333-1234	320-333-1230	
16	FALSE	Johnson	Jimmy		Jimmy Johnson				Ext.53	715-443-2123	
17	FALSE	Kemple	Manny		Manny Kemple			mannyk@wfo.epicor.com	612-444-1547	612-444-1545	
18	FALSE	Lanier	Jerry		Jerry Lanier				612-443-9087 x564	612-443-9080	
19	FALSE	Maolwald	Nancy		Nancy Maolwald			administrator@wfo.epicor.com	52-5-250-0976		
20	FALSE	Miller	Fred	R	Fred R Miller	Sales Person		fmiller@aol.com	502-241-2222		502-609-1333
21	FALSE	Shipler	Jim		Jim Shipler				563-455-6723		
22	FALSE	Smith	John		John Smith	Accounts Receiveable Dept.	AR Clerk		206-378-2378		
23	FALSE	Smith	Lonnie		Lonnie Smith			lsmith@wfo.epicor.com	608-555-5556 ext 305	608-555-5566	
24	FALSE	Smith	Lonnie		Lonnie Smith						
25	FALSE	Vincent	Alfred		Alfred Vincent	Vice President	Vice President	administrator@wfo.epicor.com			
26	FALSE	Wellstonia	Pauleen		Pauleen Wellstonia				877-543-0989		
27											
28											

Display in Excel PivotTable

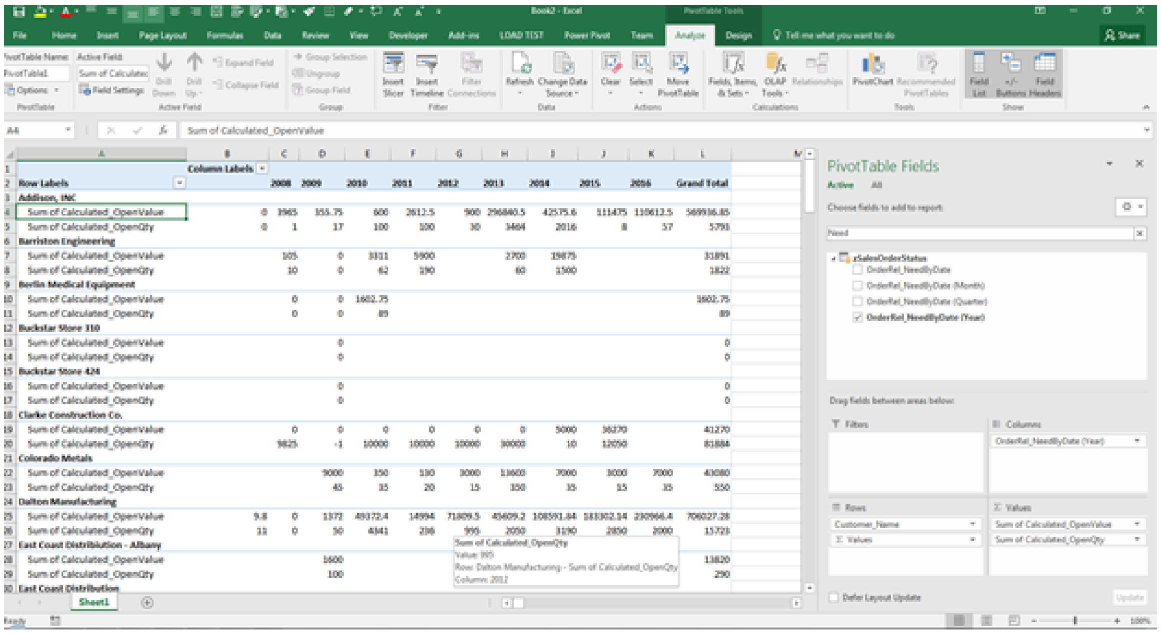
Do the following to use REST services to link Epicor data for display in a PivotTable Report.

1. Launch **Microsoft Excel**.
2. Click on the **Data** ribbon.
3. Select the **Connections** button.
   The **Existing Connections** window displays.
4. From the **Connections** tab, select one of the existing connections and click **Open**.
   The **Data Connection Wizard** displays; the connections currently available from your REST services appear in this window.
5. Accept the **Location of the data feed** that displays; enter your **User Name** and **Password**.
6. Click **Next**.
   The **Select Tables** window displays.
7. Verify the table you want to populate with your query data displays.
8. Click **Finish**.
   A PivotTable now appears in the spreadsheet and the PivotTable Fields pane displays.
9. Use the **Microsoft Excel** tools to set up the pivot table. You can select the fields you want to display as rows
10. You can also create calculations against selected fields to generate summary rows.
11. Next determine the **Columns** you want to display on the PivotTable.

    Tip For more details on how to create a PivotTable Report, review your Microsoft documentation.

12. **Save** the spreadsheet.

The data now displays within your PivotTable Report. As users enter data within the Epicor application, you can update this report by click the **Refresh Data** button. The current data updates within the spreadsheet. The following illustration depicts live Epicor data displaying within an Excel PivotTable Report.



Sum of Calculated_OpenValue

	Column Labels										
Row Labels		2008	2009	2010	2011	2012	2013	2014	2015	2016	Grand Total
Addison, INC											
Sum of Calculated_OpenValue	0	3965	355.75	600	2612.5	900	296840.5	42575.6	111475	110612.5	569936.85
Sum of Calculated_OpenQty	0	1	17	100	100	30	3464	2016	8	57	5793
Barriston Engineering											
Sum of Calculated_OpenValue		105	0	3311	5900		2700	19875			31891
Sum of Calculated_OpenQty		10	0	62	190		60	1500			1822
Berlin Medical Equipment											
Sum of Calculated_OpenValue		0	0	1602.75							1602.75
Sum of Calculated_OpenQty		0	0	89							89
Buckstar Store 310											
Sum of Calculated_OpenValue			0								0
Sum of Calculated_OpenQty			0								0
Buckstar Store 424											
Sum of Calculated_OpenValue			0								0
Sum of Calculated_OpenQty			0								0
Clarke Construction Co.											
Sum of Calculated_OpenValue		0	0	0	0	0	0	5000	36270		41270
Sum of Calculated_OpenQty		9825	-1	10000	10000	10000	30000	10	12050		81884
Colorado Metals											
Sum of Calculated_OpenValue			9000	350	130	3000	13600	7000	3000	7000	43080
Sum of Calculated_OpenQty			45	35	20	15	350	35	15	35	550
Dalton Manufacturing											
Sum of Calculated_OpenValue	9.8	0	1372	49372.4	14994	71809.5	43609.2	108591.84	183302.14	230966.4	706027.28
Sum of Calculated_OpenQty	11	0	30	4341	236	995	2050	3190	2850	2000	15723
East Coast Distribution - Albany											
Sum of Calculated_OpenValue			1600								13820
Sum of Calculated_OpenQty			100								290
East Coast Distribution											

Sum of Calculated_OpenQty
Value: 995
Row: Dalton Manufacturing - Sum of Calculated_OpenQty
Column: 2012

PivotTable Fields

Choose fields to add to report: Need

zSalesOrderStatus
- [ ] OrderRel_NeedByDate
- [ ] OrderRel_NeedByDate (Month)
- [ ] OrderRel_NeedByDate (Quarter)
- [✓] OrderRel_NeedByDate (Year)

Drag fields between areas below:

Filters:
Columns: OrderRel_NeedByDate (Year)
Rows: Customer_Name; Σ Values
Values: Sum of Calculated_OpenValue; Sum of Calculated_OpenQty

Display in Excel PivotChart

Do the following to use REST services to link real time Epicor data for display in a PivotChart.

1. Launch **Microsoft Excel**.
2. Select the **Data** tab.
3. Using the **Menu Ribbon**, select the **Other Sources** option.
4. From the list, select the **From OData Feed** option.
   The **Data Connection Wizard** displays.
5. For the **Location of the data feed**, ether enter or click the **Browse…** button to define the data feed you wish to connect with this Excel spreadsheet. For example:
   - **https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers**
   - **https://appserver01/2017R/api/v1/BaqSvc/zSalesOrderStatus**
6. Next enter your **User Name** and **Password**.
7. Click **Next**.
   The **Select Tables** wizard step displays.
8. Select the table(s) you want to connect to this spreadsheet.
9. Click **Next**.

10. Now enter a **File Name** for the **Data Connection** file.

    This file contains the settings you defined through the Data Connection Wizard.

11. Enter a concise **Description** that explains the purpose for the external database connection.

12. Click **Finish**.
    The **Import Data** dialog box displays.

13. Indicate how you want the data to display in the spreadsheet. Select the **PivotChart** option.

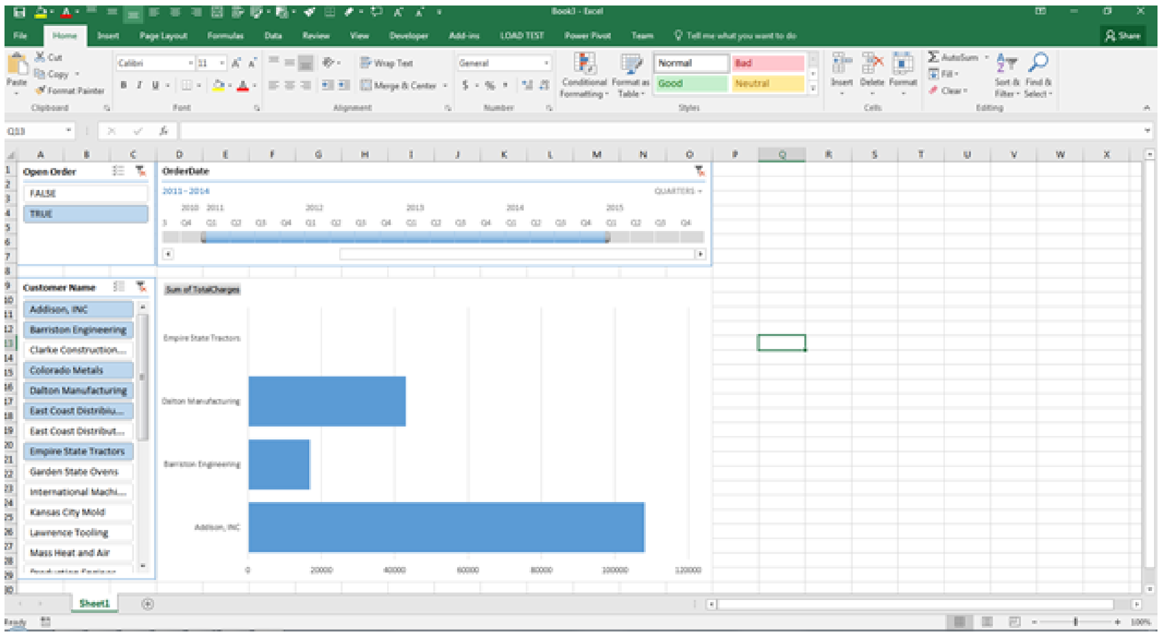
14. Click **OK**.
    A PivotChart now appears in the spreadsheet and the **PivotChart Fields** pane displays.

15. Use the **PivotChart Fields** pane to set up the pivot chart. You can select the fields that contain the data you want to display within your chart.

    Tip For more details on how to create a PivotChart, review your Microsoft documentation.

16. **Save** the spreadsheet.

The data now displays within your PivotChart. As users enter data within the Epicor, this chart updates the values, displaying current data within the spreadsheet. The next illustration shows an example of real time Epicor data that displays through an Excel Pivot table.



Epicor Services

This section documents the primary details about the Epicor REST services available on your server. This information explains how the Epicor REST services interact with your Epicor database.

Each Epicor service contains these main parts:

- **OData Datasources** – REST uses these datasources to display and modify the data in a format consumable by web applications.
- **Methods** – Use both standard and custom methods to access, add, and update data within the main table set for each service.

OData Datasources

The OData Datasources are compliant with the protocols for OData version 3.

Because of this, you can access the features available with this version. It does this by running **Create**, **Read**, **Update**, and **Delete** (CRUD) actions on specific business objects. It also performance OData query options on the collection of business objects.

Tip You can review the OData Version 3.0 Core Protocol on this website: http://www.odata.org/documentation/odata-version-3-0/odata-version-3-0-core-protocol/

Service Directories

You use the following URLs to access the Epicor REST services:

- Example directory path to all services –

  https://appserver01/2017R/api/v1/
- Example directory path to a specific service –

  https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/
- Example directory for service metadata for a specific service –

  https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/$metadata

Business Activity Query (BAQ) Services

You run the same Create, Read, Update, and Delete actions on specific BAQ objects, as well as run OData query operations on business object collections. However to run an action, you use the POST HTTP method. Each BAQ resource has these features:

- OData $metadata for the BAQ; this data describes the data structure of the BAQ results.
- OData query options such as $select and $filter; these options limit the data returned by the BAQ.
- Custom query parameters (if supported by the BAQ).
- RESTful update actions (if supported by the BAQ).

When you call a BAQ service, use the following syntax:

- **https://[MyAppServer]/[MyShare]/api/[VersionNumber]/BaqSvc/[BaqName]?[baq-param]=x&[baqParam2]=y**

Use the following URLs to access business activity queries (BAQs) through the Epicor REST services:

- Example directory for all BAQs in the current company –

  https://appserver01/2017R/api/v1/BaqSvc/
- Example directory for all BAQs for a specific company –

  https://appserver01/2017R/api/v1/BaqSvc/?Company='EPIC03'
- Example directory for service metadata for a specific BAQ –

  https://appserver01/2017R/api/v1/BaqSvc/COM-90daysSORel/$metadata
- Example directory for service metadata for a specific BAQ in a specific company–

  https://appserver01/2017R/api/v1/BaqSvc/COM-90daysSORel('EPIC03')/$/$metadata

Methods

The Epicor REST services support the **GET**, **POST**, **PATCH**, and **DELETE** HTTP methods. When you need to access business activity queries (BAQs) and updatable business activity queries (uBAQs) through HTTP, you use the **GET** and **PATCH** methods.

You can then run both standard and custom methods against the Epicor REST services.

Standard Methods

The standard methods require case sensitive syntax to properly execute. You can run the **GetList**, **GetRows**, **GetByID**, and **UpdateExt** methods to display data and update the main tableset for each service.

The OData datasource also supports these additional, special parameters:

- **$filter** – Limits the data results through criteria you define.
- **$select** – Defines specific columns to display in the data results.
- **$expand** – Includes all the child entities in the data results.
- **$top** – Pages through the list of data results.
- **$skip**— Pages through the list of data results.
- **$orderby** – Sorts the data results by a value you define.

The Epicor REST services support the following code formats:

- **application/json**
- **application/atom+xml**

All OData URI elements are case sensitive. For your standard method calls to run properly, any field names, $-like parameter names, and values must also be case sensitive. The following example pulls the ‘A’ record from the ABCCodes dataset:

- **https://appserver01/2017R/api/v1/Erp.BO.ABCCodeSvc/ABCCodes?$filter=ABCCode1 eq 'A'**

Notice that the call to the ABCCodes tab, the $filter parameter, and ABCCode1 dataset are all case sensitive values. You must also use ABCCode1 in this syntax, because the of the name collision between the table and the column name.

The following example uses a $filter parameter to pull a date time value:

- **https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers/?$select=EstDate&$filter=EstDate eq datetime'2016-02-22T00:00:00'**

The $filter parameter also only works with fields that exist internally in the database. This parameter uses **Where** clauses for both the **GetRows** and **GetList** service calls, and these calls are translated into stored procedure parameters.

Custom Methods

You can create custom methods that can access any Epicor REST service. These services must be called using the **POST** HTTP method. You must also write custom methods using the **application/json** format. However unlike standard methods, custom methods do not require case sensitive values. You also do not need to rename tablesets; all fields in your custom code can use the same value as their actual tableset names.

OData Querying

OData uses special parameters on URLs to do querying via REST similar to using SQL to query a database.

The URL of a service support a couple of different patterns. You can enter a URL to identity a single Entity or a Collection of Entities. You can query for 'Parent / Child' data by navigating the relations defined in the service. You can also apply one or more parameters to the query.

Important The names of tables and columns in queries are case sensitive and must match tables and columns' names in Tablesets.

Example In the query for a **Customer** with a **CustID** of **Addision**, the column **CustID** must match the column in the Tableset:

https://EpicorServer/ERP101600/api/v1/Erp.Bo.CustomerSvc/Customers?$filter=CustID eq 'Addison'

Querying for Collections vs Single Entity

Collections

Querying for a collection is as simple as not specifying any parameters.

1. Navigate to the **REST Help URL**. For example: **https://EpicorServer/ERP101600/api/help**. When prompted, use your Epicor credentials.
2. Expand the **Service List** section.
3. Search for a service by entering its name. In this example, you enter **Customer**. All the services that contain "Customer" display in the drop-down list. Click on the **CustomerSvc** name.
4. Click GET / Customers tp expand the section.
5. Click **Try it out!**

   The collection of results is returned.

Single Entity

When you want a specific item, you can query for that record by using its key fields.

1. Click **GET /Customers({Company},{CustNum})** to expand the section.
2. In the Company text box, enter **'Epic06'**.
3. In the CustNum text box, enter **2**.
4. Click **Try it out!**

   The single record returned for **Addison**

Note **Addison** (Customer Number 2) is a sample customer ID from the Epicor training database. If you use a different database, take any existing customer number from your database to use in this query.

Note Strings and decimals are enclosed in single quotes **'**.

Navigating Relationships

Relationships exist within the Epicor 'ZData' system that is used for generating Tablesets and Datasets. These relations can be used to navigate from a parent record to its child or children. Note only the children records are returned. To have both the parent and children returned, the $expand parameter needs to be used.

Child Collection

You can get the 'children' of a record by appending the related entity to the URL.

1. Click **/Customers({Company},{CustNum})/ShipToes** to expand the section.
2. In the Company text box, enter '**Epic06'**.
3. In the CustNum text box, enter **2**.
4. Click **Try it out!**

   The results are the ShipToes for **Addison**.

Single Child

You can get the single child of a record by appending the related entity and key to the URL.

1. Click **/Customers({Company},{CustNum})/ShipToes/({Company},{CustNum},{ShipToNum})** to expand the section.
2. In the Company text box, enter '**Epic06'**.
3. In the CustNum text box, enter **2**.
4. In the ShipToNum text box, enter **'001'**.
5. Click **Try it out!**

   The results are the ShipToes with ShipToNum val;ue equal to **001** for **Addison**.

$Parameters Introduction

Syntax Rules

- Query Parameters are prefixed with a **$** character.
- Query Parameters can be appended on a URL with an **&**.
- Many Parameters take multiple values and are separated by a comma.

Overview of Parameters

Customer service will be used for demonstration.

1. In your browser, navigate to the REST Help URL. For example: https://EpicorServer/ERP101600/api/help. When prompted, use your Epicor credentials.

2. Expand the Service List section. Search for a service by entering its name. In this example, you enter "Customer". All the services that contain "Customer" display in the drop-down list.

3. Click on the CustomerSvc name. A new page displays that shows the available REST resources and actions available within the Customer service.

$select

This parameter selects specific columns:

1. Click **GET /Customers** to expand the section. Click **Try it out!**

   All the columns are returned.

2. In the $select text box, enter **Company, CustID, CustNum**. Click **Try it out!**

The results just include the columns requested.

$filter

This parameter is equivalent of where clauses in SQL:

1. Keep the $select entry from the previous step.

2. In the $filter text box, enter **State eq 'MN'**.

3. Click **Try it out!**

The results just include records in the state of Minnesota.

Operator List

OData supports many operators such as the ones below. A complete list is at the OData specification site.

Note ERP 10 does not support the following operators yet: IsOf, Any, All

Logical Operators

eq	Equal	/Suppliers?$filter=Address/City eq 'Redmond'
ne	Not equal	/Suppliers?$filter=Address/City ne 'London'
gt	Greater than	/Products?$filter=Price gt 20
ge	Greater than or equal	/Products?$filter=Price ge 10
lt	Less than	/Products?$filter=Price lt 20
le	Less than or equal	/Products?$filter=Price le 100
and	Logical and	/Products?$filter=Price le 200 and Price gt 3.5
or	Logical or	/Products?$filter=Price le 3.5 or Price gt 200
not	Logical negation	/Products?$filter=not endswith(Description,'milk')

Arithmetic Operators

add	Addition	/Products?$filter=Price add 5 gt 10
sub	Subtraction	/Products?$filter=Price sub 5 gt 10
mul	Multiplication	/Products?$filter=Price mul 2 gt 2000
div	Division	/Products?$filter=Price div 2 gt 4
mod	Modulo	/Products?$filter=Price mod 2 eq 0

Grouping Operators

( )	Precedence grouping	/Products?$filter=(Price sub 5) gt 10

String Operators

bool substringof(string searchString, string searchInString)	substringof('Alfreds',CompanyName)
bool endswith(string string, string suffixString)	endswith(CompanyName,'Futterkiste')
bool startswith(string string, string prefixString)	startswith(CompanyName,'Alfr')
int length(string string)	length(CompanyName) eq 19
int indexof(string searchInString, string searchString)	indexof(CompanyName,'lfreds') eq 1
string replace(string searchInString, string searchString, string replaceString)	replace(CompanyName,' ', '') eq 'AlfredsFutterkiste'
string substring(string string, int pos)	substring(CompanyName,1) eq 'lfreds Futterkiste'
string substring(string string, int pos, int length)	substring(CompanyName,1, 2) eq 'lf'
string tolower(string string)	tolower(CompanyName) eq 'alfreds futterkiste'
string toupper(string string)	toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
string trim(string string)	trim(CompanyName) eq 'Alfreds Futterkiste'
string concat(string string1, string string2)	concat(concat(City,', '), Country) eq 'Berlin, Germany'

Date Operators

int day(DateTime datetimeValue)	day(BirthDate) eq 8
int hour(DateTime datetimeValue)	hour(BirthDate) eq 1
int minute(DateTime datetimeValue)	minute(BirthDate) eq 0
int month(DateTime datetimeValue)	month(BirthDate) eq 12
int second(DateTime datetimeValue)	second(BirthDate) eq 0
int year(DateTime datetimeValue)	year(BirthDate) eq 1948

Math Operators

double round(double doubleValue)	round(Freight) eq 32
decimal round(decimal decimalValue)	round(Freight) eq 32
double floor(double doubleValue)	floor(Freight) eq 32
decimal floor(decimal datetimeValue)	floor(Freight) eq 32
double ceiling(double doubleValue)	ceiling(Freight) eq 33

decimal ceiling(decimal datetimeValue)	ceiling(Freight) eq 33

$top

This parameter retrieves a certain selected number of records:

1. Clear the $filter text box.
2. In the $top text box, enter **2**.
3. Click **Try it out!**

Only 2 records are returned.

$skip

This parameter is used with top for paging. Skip the first y-number of records and grab the next z-number of them.

1. In the $top text box, enter **2**.
2. In the $top text box, enter **4**.
3. Click **Try it out!**

Only 2 records are returned.

If you compare this to the complete list of Customer results, you will note the records 5 & 6 were retrieved. (skip 1-4, grab next 2).

$orderby

This parameter allows to set a list of columns in either ascending or descending order:

1. Clear $top and $skip.
2. In the $orderby text box, enter **Name Asc**.
3. Click **Try it out!**

The results are sorted alphabetically by Name.

$expand

Navigation was described above as the ability to walk from a parent to its child record and retrieve the child. $expand gives the ability to return a hierarchy of parent and child in one result set:

1. Click **GET /Customers({Company},{CustNum})** to expand the section.
2. In the Company text box, enter '**Epic06'**.
3. In the CustNum text box, enter **2**.
4. In the $expand text box, enter **ShipToes**. Click **Try it out!**

   All the hierarchy returned.
5. In the $select text box, enter **Company, CustID, CustNum**, **ShipToes**. Click **Try it out!**

   Just the few columns from the Customer parent record return with the full ShipToes record.

6. In the $select text box, enter **Company, CustID, CustNum**, **ShipToes/ShipToNum**

7. Click **Try it out!**

   Just the columns entered appear in the hierarchy returned.

Functions and Operators Not Supported

The list of items unsupported by Epicor REST API by category.

1. **Type Functions**
   a. `isof`.

2. **Lambda operators**
   a. `any`.
   b. `all`.

REST Examples

This section illustrates some code examples for creating and updating records, connecting through WCF, linking updatable BAQs, and other functions you can customize through REST services. Use these code examples to help you leverage REST services for your SDK projects.

Tip While you can write your own code through C# and JSON, a number of tools are available you can use to generate code that interfaces with Epicor services. For example, one of these services is Postman; it generates runnable code in the programming language you need. You can download this app from: https://www.getpostman.com.

Postman

Postman is a free tool for API developers to share, test, document and monitor APIs.

When you start doing 'POST' work against custom methods and updates, the Swagger help page quickly becomes limited. With Postman, real development level calls can be done. The use of Postman against REST is similar to the use of BLTester against the SOA services. Additionally, the calls can be saved as a query within a larger collection and shared or used as a test suite.

Go to this web site to download the free Postman App.

Executing a Get

1. Open Postman.
2. Expand the collection in tree on the left **Epicor 10 | Rest Samples**.
3. Select the first item - 'Get 1.1 OrderList with GetRows'.
4. In the Upper Right, Dropdown the Environment combobox to see the different Environments for which you are configured.
   You can create a test, dev, production environment and flip between them quickly to isolate data problems.
5. Client on the eye to the Environment combobox.
   - Note that the key 'Host' has a value pointing to the server.
   - Close the popup dialog.
6. Note the URL at the top of the main client area: *{{Host}}/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders*.
   {{Host}} is a key to be replaced during execution by the environment setting previously shown.
7. Select Type and choose Basic Authentication.
8. Enter your Epicor credentials.
9. Select Update Request.

10. Click **Send**.
    - Note the results are SalesOrders.

11. Click **Headers**.
    - Note the Context Header information for BPMs.

Executing a POST

1. Select the first item - **POST 1 Create new Customer**.
2. Click on **Body**.
   - Note the data being sent to the REST service.
   - Make sure 'JSON' is the format.
3. Click **Send**.
   - Note the new record created at the bottom with a new CustNum.

Diagnosing an Error

Many times the REST APIs might give helpful information. In these cases, you need to review the results in the Windows Event Viewer. Epicor ERP 10 has a 'Flight Data Recorder' log.

Note Go to **System Management > Working with System Management > Performance Tuning Guide > Microsoft Tools > Event Viewer** for details on Windows Event Viewer.

1. Click **Send**.
2. If an Error Message is returned, click on the 'Start button' and enter Event Viewer in the search text box. Select **Event Viewer** from the list provided.
3. In the Window Event Viewer, expand **Applications and Services Logs**.
4. Select **Epicor App Server**.
5. Select the error.

   Note The description of the exception is displayed.

Retrieve Data

This section illustrates some way you can retrieve data from the sales order and customer REST services.

These examples use the GET HTTP method. Some of them also use additional OData parameters $expand and $filter. Use this same format to access data from other Epicor REST services.

- Display a Sales Order List –

  https://appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders/

- Use the Atom Format –

  https://appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders/?$format=atom

- Display a list of sales orders with a smaller entity size (it uses the GetList method to pull data) –

  https://appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/List

- Display a list of customers with a smaller entity size (it uses the GetList method to pull data) –

  https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/List

- Display a list of sales orders that expand to include order detail lines --

  https: //appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders?$expand=OrderDtls

- Display a list of sales orders that expand to include order detail lines and releases –

  https: //appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders?$expand=OrderDtls/OrderRels

- Use the $select filter to limit the results to return order and order lines –

  https: //appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/SalesOrders?$expand=OrderDtls&$select=OrderNum,OrderDtls/OrderLine

- Use the $top filter to limit the number of records that return –

  https: //appserver01/2017R/ api/v1/Erp.Bo.SalesOrderSvc/SalesOrders?$top=2&$expand=OrderDtls&$select=OrderNum,OrderDtls/OrderLine

Create New Customer

The following example illustrates how to add a new customer to the Epicor database.

Use this code as a base example for creating new records through REST in the Epicor database.

```
POST https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers HTTP/1.1
Host: appserver01
Content-Type: application/json
Authorization: Basic
Cache-Control: no-cache
    {
      "Company": "EPIC06",
      "CustID": "MYTEST",
      "CustNum": 0,
      "Name": "My test customer",
      "TerritoryID":  "US MID"
    }
```

Now when you save this record, the CustNum value becomes the primary key for the new customer record. You can then later retrieve this customer number through a GET HTTP method. For example:

```
GET https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers ('EPIC06',47)
 HTTP/1.1
```

Update Customer

The following code example illustrates how you can update an existing customer record.

Use this code as a base example for updating existing records.

```
PATCH https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers ('EPIC06',4
7) HTTP/1.1
Host: appserver01
Content-Type: application/json
Authorization: Basic
    {
      "Company": "EPIC06",
      "CustNum": 47,
      "Name": "Another Name for test customer"
    }
```

Delete Customer

This code example removes an existing customer from the database.

```
DELETE https://appserver01/2017R/api/v1/Erp.Bo.CustomerSvc/Customers('EPIC06',4
7) HTTP/1.1
Host: appserver01
Content-Type: application/json
Authorization: Basic
```

Update Child Entity

You can also update the entities for records.

This C# code example shows you how to update entities within user code records.

```
public void CreateUserCode()
        {
            HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post
, ServiceUrl + "/Ice.Bo.UserCodesSvc/UserCodes");

            request.Content = new StringContent(JsonConvert.SerializeObject(new

                {
                    Company = "EPIC03",
                    CodeTypeID = "type1",
                    CodeTypeDesc = "type",
                    LongDesc = "Description of ud type",

                    UDCodes = new[]
                    {
                        new  {
                                Company = "EPIC03",
                                CodeTypeID = "type1",
                                CodeID = "my UD code",
                                CodeDesc = "my code desc",
                             }
                    }
                }),
                Encoding.Default,
                "application/json");
            HttpResponseMessage response = client.SendAsync(request).Result;
        }
```

Delete Child User Code; Leave User Type Intact

This C# code example illustrates how you remove a user code, but do not remove a user type.

```
string parentKeystr = "Company='EPIC03',CodeTypeID='type1'";

            string childKeystr =
                "Company='EPIC03',CodeTypeID='type1',CodeID = 'my UD code'";

            request = new HttpRequestMessage(HttpMethod.Delete, ServiceUrl +
                String.Format("/Ice.Bo.UserCodesSvc/UserCodes({0})/UDCodes({1})
", parentKeystr, childKeystr));

            response = client.SendAsync(request).Result;
```

Run BAQ

You can use custom code to run a specific business activity query (BAQ) and display specific records from that query.

For example:

- **https://appserver01/2017R/api/v1/BaqSvc/BaqName?param='2016-12-01'**

To run this query, you substitute the **BaqName** value with the **Query ID** for the query you wish to access.

The optional param value defines any additional parameters you wish to run against the query. These parameter values are specified through a query string. Any Date parameters must use the ISO form ('2016-12-01') illustrated in this example. If you want to display a value list, you can specify it several times. For example if you have a value list parameter defined in the BAQ Designer named listParam, your query string uses the following syntax:

- **listParam=value1&listParam=value2&listParam=value3**

You can also run the $filter OData parameter against the BAQ results. These filters are sent as query execution parameters.

Updatable BAQ

The following example illustrates how to create an empty updatable business activity query (uBAQ) row where a new record can be entered and submitted to the database. This example uses the GET HTTP method.

https://appserver01/2017R/api/v1/BaqSvc/<uBAQname>/GetNew - change <uBAQname> to the name of uBAQ you use.

```
GET https://appserver01/2017R/api/v1/BaqSvc/Udtip/GetNew HTTP/1.1
Host: appserver01
Authorization: Basic
```

You then update the record using the JSON format.

```
PATCH https://appserver01/2017R/api/v1/BaqSvc/ Udtip HTTP/1.1
Host: appserver01
Authorization: Basic
Content-Type: application/json
{
      "Tip_Company": "EPIC03",
      "Tip_MfgSys": "EP",
      "Tip_TipNum": 1,
      "Tip_Active": true,
      "Tip_TipTitle": "2121",
      "Tip_TipText": "####Changed TipText of tip",
      "Tip_ShowNum": 0

  }
```

You can also delete records using uBAQ. First, you need to configure the mapping for RowMod in the uBAQ definition:

1. Open **Business Activity Query Designer**.

   Menu Path: System Management > Business Activity Queries > Business Activity Query

   Important This program is not available in Epicor Web Access.

2. Load the uBAQ you want to use for deleting records.

3. Navigate to the **Update > Update Processing** sheet.

4. In the **Processing Method** pane, select the **BPM Update** radio-button. The **BPM Update Processing** section activates.

5. The **Select Business Object** window displays. Select the business object you want from the **Suggested** business objects list.

6. In the **Query to Object Column Mapping** grid, add a mapping between a BO table RowMod field and the query RowMod column, for example, Tip.RowMod = ttResult.RowMod.

7. Click **Save**.

When the mapping is defined, you can set RowMode to D (delete) in the PATCH request.

```
PATCH https://appserver01/2017R/api/v1/BaqSvc/ Udtip HTTP/1.1
Host: appserver01
Authorization: Basic
```

```
Content-Type: application/json
{
      "Tip_Company": "EPIC03",
      "Tip_MfgSys": "EP",
      "Tip_TipNum": 1,
             "RowMod": "D"

  }
```

Custom Methods

You can also create custom methods that interact with REST services.

You can invoke non-standard service methods. These custom methods must use the **POST** HTTP method call, so these methods can only update existing records.

Custom methods also must use the **application/json** format. Because of this, they contain input parameters sent as JSON object properties. The results returned from the method call are sent in a JSON object. Within this object, the **returnObj** setting contains the return value and parameters – including the **Out** and **ByRef** parameters. Null values do not return from the method query.

Example 1

```
POST https://appserver01/2017R/api/v1/Ice.BO.LangTranSvc/UpdateSingleTranslatio
n/
Request body {"pcLanguage" : "rus", "pcProgram" : "qq", "pcText" : "test", "pcT
ranslation" : "     ", "pcProgTranslation": "     "}
Response
{
  "parameters": {
    "bSeccess": true
  }
}
```

Example 2

```
POST https://appserver01/2017R/api/v1/Ice.BO.TipSvc/GetNewTip/
 Request {"mfgSys":"EP","ds":{"Tip":[]}}
Response
{
  "parameters": {
    "ds": {
      "Tip": [
        {
         "Company": "EPIC03",
          "MfgSys": "EP",
          "TipNum": 0,
          "Active": true,
          "TipTitle": "",
          "TipText": "",
          "SystemFlag": false,
          "SysRevID": 0,
          "SysRowID": "00000000-0000-0000-0000-000000000000",
          "AllCompanies": false,
          "BitFlag": 0,
          "RowMod": "A",
        }
      ]
    }
  }
}
```

Example 3

```
POST https://appserver01/2017R/api/v1/Ice.BO.TipSvc/GetBySysRowIDs/
Request {"ids": ["e830ab94-680b-49fe-882c-9f9d0117ac39"]}
Response
 {
  "returnObj": {
    "Tip": [
      {
        "Company": "",
        "MfgSys": "EP",
        "TipNum": 13,
        "Active": true,
        "TipTitle": "Quick Calendar Access",
        "TipText": "You can quickly access calendars by right-clicking any Date
 field.@@##",
        "SystemFlag": true,
        "SysRevID": 15286382,
        "SysRowID": "e830ab94-680b-49fe-882c-9f9d0117ac39",
        "AllCompanies": true,
        "BitFlag": 0,
        "RowMod": "",
      }
    ]
  }
}
```

Custom Headers

You can create custom headers to adjust server call behavior.

These headers implement **ICallHeader** interface from Epicor.ServiceModel and are used the same way ICallHeader is used in WCF.

Custom headers are sent as HTTP request/response headers. Request headers are always in JSON regardless of the specified **Content-Type**. Response headers are also in JSON, except for the **CallerTrace** header, which is a base64-encoded string.

Below is the list of available custom headers with examples.

CallSettings Header

Used to specify additional settings for a call.

You can use the **CallSettings** header to allow the client to specify both **Company** and **Site** information in a call.

C# Example

```
private static HttpClient client = new HttpClient();

            ICallHeader hdr = new CallSettings("EPIC03", "MfgSys", "", "");
            client.DefaultRequestHeaders.Add(hdr.Name, JsonConvert.SerializeObj
ect(hdr));
```

Http Header Example

```
Header name: CallSettings
Header value: {"Company":"EPIC03","Plant":"MfgSys","Language":"","FormatCulture
":"" }
```

SessionInfo Header

SessionInfo header sends SessionID to the server.

SessionID can be obtained from the **SessionMod** service, **Login** method, for example:

```
POST /ice3/api/v1/Ice.Lib.SessionModSvc/Login HTTP/1.1
Host: localhost
Authorization: …
Content-Type: application/json
```

The result of the call will contain **SessionID** to use in the session:

```
{
    "returnObj": "232a2868-cf60-46f1-8dba-1ac53328a66b"
}
```

SessionInfo header will look like this:

```
SessionInfo: {"SessionID": "232a2868-cf60-46f1-8dba-1ac53328a66b"}
```

Now it can be sent with each REST request during this session.

When a session is no longer needed, **Logout** method of **SessionMod** should be called:

```
POST /ice3/api/v1/Ice.Lib.SessionModSvc/Logout HTTP/1.1
Host: localhost
Authorization: …
Content-Type: application/json
SessionInfo: {"SessionID": 232a2868-cf60-46f1-8dba-1ac53328a66b"}
```

OnBehalfOfToken Header

In a REST call, this header allows a user with impersonation rights accomplish tasks on behalf of other users in Epicor ERP.

If for any reason a task needs to be run on behalf of another user, add the OnBehalfOfToken header and specify the user on whose behalf you will run this task.

For example, you have primary user **Agent** for call authorization, but you need to make this call on behalf of user **Epicor**. Add the below value to the OnBehalfOfToken header:

Key	**Value**
OnBehalfOfToken	{"Username":"Epicor"}

Your request may look like this:

```
POST /ice3/api/v1/Ice.BO.UserFileSvc/GetUserFile HTTP/1.1
Host: localhost
Authorization: Basic …
Content-Type: application/json
OnBehalfOfToken: {"Username":"Epicor"}
```

Note the response: your call is accomplished on behalf of user **Epicor**. Top 10 lines of the response to the above request:

```
{
"returnObj": {
```

```
"UserFile": [
{
"UserID": "Epicor",
"Name": "Epicor",
"Address1": "",
"Address2": "",
"City": "",
"State": "",
```

Important The primary user (e.g. **Agent**) must be allowed **Session Impersonation** in the **User Account Security Maintenance** program.

License Header

Sends the license ID that should be used for a call.

For example, you can specify the CRM license in the header:

```
GET /ice3/api/v1/Ice.BO.TipSvc/Tips HTTP/1.1
Host: localhost
Authorization: Basic …
License: {ClaimedLicense: "00000003-15F2-4059-8C79-11849198F488"}
```

Trace Flags and Caller Trace Headers

Allow returning server traces stored in OperationTracer to the client.

If the **TraceFlags** request header is specified as empty class, like below:

```
GET /ice3/api/v1/Ice.BO.TipSvc/Tips HTTP/1.1
Host: localhost
Authorization: Basic …
TraceFlags: {}
```

the response will contain the **CallerTrace** header with **OperationTracer** content defined by trace flags on the server side (in the **AppServer.config** file).

The **CallerTrace** response header contains base64-encoded data:

```
Callertrace PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTE2Ij8+DQo8T3AgVXRjPSI
yMDE3LTEwLTEzVDE5OjM4OjUyLjU2NzY2OTJaIiBhY3Q9IiIgZHVyPSIyNzYuNzM5NyIgY2xpPSIiIH
Vzcj0iIiBtYWNoaW5lPSJSTTA2MTctUEYwUlM5WUciIHBpZD0iOTU0MCIgdGlkPSIxMzAiIC8+
```

To get its text content, use **Base64** decode functions. For example, in **C#**, the header can be decoded as:

```
byte[] data = Convert.FromBase64String(encodedTrace);
string trace = Encoding.UTF8.GetString(data);
```

After decoding, the header content will look like this:

```
<?xml version="1.0" encoding="utf-16"?>
<Op Utc="2017-10-13T19:38:52.5676692Z" act="" dur="276.7397" cli="" usr="" mach
ine="RM0617-PF0RS9YG" pid="9540" tid="130" />
```

There is a possibility to specify additional flags that you want to see in the header.

For example, add new flags in the **FlagList** attribute of the **TraceFlags** header:

```
GET /ice3/api/v1/Ice.BO.TipSvc/Tips HTTP/1.1
Host: localhost
Authorization: Basic …
```

```
License: {ClaimedLicense: "00000003-15F2-4059-8C79-11849198F488"}
TraceFlags: {FlagList:["trace://ice/fw/license", "profile://ice/fw/tableset"]}
```

Now, the **CallerTrace** will contain additional traces. The response to the above request (after decoding) will look like this:

```
<?xml version="1.0" encoding="utf-16"?>
<Op Utc="2017-10-13T19:49:43.3950234Z" act="" dur="230.2327" cli="" usr="" mach
ine="RM0617-PF0RS9YG" pid="9540" tid="99">
  <License msg="Obtaining license for session type:'CRM' ID='00000003-15f2-4059
-8c79-11849198f488'" />
  <License msg="Granted 'CRM' license #1 of 30000." />
  <License msg="Claim on license extended until: SessionID:19247e63-a49e-4996-a
173-110eaba1cc78 LicenseExpiration: 10/13/2017 8:04:00 PM" />
  <Sproc name="[Ice]._ZFW_Tip_GetRows" duration="65.8362" />
  <RowEvent table="Tip" method="AfterGetRows" rows="100" duration="0.0159" />
  <RowCount tableset="TipTableset" count="100" />
</Op>
```

Note Not all existing traces from AppServer.config are added to the OperationTracer data. Some flags are only used in server-side logging. Such traces will not be returned in this **CallerTrace** header, even if they are specified.

Important Trace size in the response can be quite large if server operation was lengthy and traces are very detailed. There are limits for headers' size in IIS (16 KB) that will not allow such long headers. Therefore, the limits must be adjusted if a very long **CallerTrace** is expected. Alternately, reduce the list of trace flags in the request.

ContextHeader Header

Allows sending and receiving additional call information.

This header is primarily used by BPM. Here is an example of the ContextHeader response header:

```
contextheader
{"Context":{"BpmData":[{"Password":"","ButtonValue":0,"Character01":"","Charact
er02":"","Character03":"","Character04":"","Character05":"","Character06":"","C
haracter07":"","Character08":"","Character09":"","Character10":"","Character11"
:"","Character12":"","Character13":"","Character14":"","Character15":"","Charac
ter16":"","Character17":"","Character18":"","Character19":"","Character20":"","
Number01":0.0,"Number02":0.0,"Number03":0.0,"Number04":0.0,"Number05":0.0,"Numb
er06":0.0,"Number07":0.0,"Number08":0.0,"Number09":0.0,"Number10":0.0,"Number11
":0.0,"Number12":0.0,"Number13":0.0,"Number14":0.0,"Number15":0.0,"Number16":0.
0,"Number17":0.0,"Number18":0.0,"Number19":0.0,"Number20":0.0,"Date01":null,"Da
te02":null,"Date03":null,"Date04":null,"Date05":null,"Date06":null,"Date07":nul
l,"Date08":null,"Date09":null,"Date10":null,"Checkbox01":false,"Checkbox02":fal
se,"Checkbox03":false,"Checkbox04":false,"Checkbox05":false,"Checkbox06":false,
"Checkbox07":false,"Checkbox08":false,"Checkbox09":false,"Checkbox10":false,"Sh
ortChar01":"","ShortChar02":"","ShortChar03":"","ShortChar04":"","ShortChar05":
"","ShortChar06":"","ShortChar07":"","ShortChar08":"","ShortChar09":"","ShortCh
ar10":"","SysRowID":"a88bb7c0-7cff-4432-800a-84f56829df08","RowMod":"","Specifi
edProperties":"AAAAAAAAAAAAAQ==","UserDefinedColumns":{}}],"CallStack":[],"Call
State":[],"Client":[{"ClientType":"","ProcessId":"","AssemblyName":"","Customiz
ationId":"","CurrentUserId":"Epicor","CurrentCompany":"EPIC03","CurrentPlant":"
MfgSys","CGCCode":"","SysRowID":"12a3f052-abaf-4803-9752-eb921c1f8fa4","RowMod"
:"","SpecifiedProperties":"/wE=","UserDefinedColumns":{}}],"ClientHandler":[],"
InfoMessage":[],"ExtensionTables":[]}}
```

Creating Web Applications Using REST Services

You may use REST services for creating and modifying web applications.

Workshop - Create a Web App with REST

In this workshop you will build a pure single page web application that can search for Customer records in ERP using REST Services.

While we are using Customer as an example service the application can be easily modified to search for any type of business object and perform different types of searches. You may use the knowledge you gain from this workshop to try and further modify the application to search other types of business object such as Parts, Vendors, Orders, or search through the results of a BAQ.

Before You Begin

Make sure you have installed the following programs: Postman, Visual Studio Code, as well as downloaded the folder with sample code:

- Download Visual Studio Code here.
- Download Postman here.
- Go to the link below to download the zipped folder with prewritten code for your app. Unzip the folder and place it in the C:\\ root directory on your computer. The path to this folder should be like this **C:\Rest\**.

Run the Application

Since this workshop is about learning what you can do with Epicor ERP's REST services we've written most of the JavaScript, CSS, and HTML code for you already. Before we start working on the code, let's just run the scaffolded application and see what it looks like.

1. Open **File Explorer** and navigate to **C:\Rest\RestWebApp\**.
2. Double click **index.html** to open the web page in your browser.
   As you can see this application is just a simple search page. It has a **Title**, a **Search Box**, and space below to show your search results.
3. Leave the web page open in your browser and proceed to the next step.

Review the Application Code

Before you make any changes, take a look at the application code. In addition to the information in this section you can find more details about how the application works in comments within index.html, app.css, and most importantly app.js.

1. Open **Visual Studio Code**.
2. Open the Project folder in Visual Studio Code by navigating to **File** > **Open Folder**. Select the **C:\Rest\RestWebApp\** folder.

3. Review the contents of the Project folder:
   - **index.html** – contains all html for the project.
   - **node_modules** – contains libraries the project depends on including Vue.js, JQuery, and Bootstrap.
   - **scripts\app.js** – contains the application's javascript.
   - **styles\app.css** – contains custom css styles for the application.

4. Open **scripts\app.js** by double clicking on the file in the Explorer panel on the lefthand side of Visual Studio Code.

   Vue.js, a light weight web framework, has been used to simplify writing data to the web page after it is loaded from REST. This may seem complicated, but it's simpler than using JQuery or plain JavaScript to update the page with data.

   App.js contains a single JavaScript object, the Vue Object, which contains all of the data you will display on the web page as well as all the methods used to call the REST services to load data.

5. Press **Ctrl-F** to activate the **Find** dialog and search for **new Vue(**.
   The Vue object contains three main elements:
   - **el** – This tells Vue what HTML element bind data within. In our case it is a <div> with `id="app"`.
   - **data** – Each data element we want to display on the page, or retrieve from user input is a property of the data object.
   - **methods** – Each function we want the UI to be able to call, such as searching for data in the rest service, is contained within the methods object.

6. Open **index.html** by double clicking the file in the Explorer panel.

7. Press **Ctrl-F** to activate the **Find** dialog and search for **{{appTitle}}**
   Vue.js uses {{mustache}} syntax to inject data into the web page as it is updated. Each place you see a {{…}} in the index.html you can inject data into the web page. Each {{variableName}} in index.html should have a corresponding variable in the data object in app.js. Changing the value in the data object will automatically update the page.

   Note Additionally, notice that there are other **v-** xxxx tags such as **v-model**, **v-if**, **v-on**, **v-for**, etc. These special tags let Vue.js interact with the page in other ways to retrieve input from the user, hide and show page elements, send events such as typing and clicking events, and loop through data for display.

8. Switch back to the **app.js** file which should be open in a tab in VSCode.

9. Press **Ctrl-** F to activate the **Find** dialog and search for **appTitle**.
   The code should look like this. If you want to change the title of the Application, update this variable and reload the web page.

```
data: {
    appTitle: 'Awesome REST App',
   …
   }
```

10. Change the appTitle to '**Customer Search**'.

11. **Save** the file.

12. Return to your browser window, where **index.html** should still be open and **Refresh** the page.

The application title should now say **Customer Search**.

Create and Test the Search REST Call in Postman

Now that you have made a simple change to the application, you can move on to hooking up REST services to search for Customers. Before you add code into your web application to call the rest services it is convenient and useful to first test out the calls you want to make in a testing application such as Postman or the ERP 10 REST Help.

Postman has the added benefit of generating working code in a variety of programming languages so in this section you will create a Customer business object search call in Postman, test it, and then generate JavaScript code that you can copy & paste into our Web Application to get started quickly.

1. Leave your browser and Visual Studio Code running.

2. Open the **Postman** App.

3. On the left-hand side in the **Collections Pane** open **Epicor 10 | Rest Samples** and select sample **6.2 Customers with filter startsWith**.
   This sample shows how to query the customer service for a list of customer objects and filter the list by Customers with CustID that start with "A":
   ```
   {{Host}}/api/v1/Erp.Bo.CustomerSvc/Customers?$filter=startswith(CustID ,'A'
   )&$select=CustID
   ```

4. Click **Send** and confirm that the query works.
   The query works, but it only returns CustID columns and it would be better for our application if we could search on both CustID and the Customer Name.

5. Remove the **&$select=CustID** from the query
   Our REST query now looks like:
   ```
   {{Host}}/api/v1/Erp.Bo.CustomerSvc/Customers?$filter=startswith(CustID ,'A'
   )
   ```

6. Click **Send** to test the new query.
   Now you get a list of all customers where CustID starts with A, including all columns.

7. To enable the query to search for customers by Name, add the following to the end of the query " **or startswith(Name, 'A')**".

   Note Make sure to include the space at the beginning, and exclude the "" marks.

8. Click **Send** to test the new query.
   Now you can see the new query searches for customers who have either a Name or a CustID starting with "A".

9. If you want the query results to be ordered by the Customer Name, add the following to the end of the query **&$orderby=Name**.

10. Click **Send** to test the new query.
    The query should now look like this. It searches for a customer by Name or ID and returns a list of customers ordered by Name:
    ```
    {{Host}}/api/v1/Erp.Bo.CustomerSvc/Customers?$filter=startswith(CustID ,'A'
    ) or startswith(Name, 'A')&$orderby=Name
    ```

11. Click dropdown button next to the **Save** button and choose **Save As**.
    A **Save Request** dialog displays.

12. Set the **Request Name** to "**Customer Search**".

13. In the **Or create new collection** box enter "E10 Rest Web App".

14. Click **Save**.
    You've now saved your new customer search query into a new collection, so it is easier to find in the future.

    Note It can be useful to keep a collection of queries for your application while you are developing it that you can return to modify and test.

Generate Code in Postman

Now that we know what request we want to send in our web application and we have tested that it works we can use Postman's code generation option to create a JavaScript code sample which we can use as a starting place in our Web Application.

1. Still in Postman, with our Customer Search query open locate the **Code** link in the page and select it. This option is located below the Save button on the far right and is in small orange text.
   The Generate Code Snippets dialog box is shown.

2. Open the **language** drop-down at the top left.
   Notice that there are a wide variety of languages that you can generate a sample code snippet in such as Java, Python, Swift, and of course JavaScript. Postman can help you get started writing code in many languages to talk to our REST services.

3. Select **JavaScript > JQuery AJAX** from the drop-down list.
   A code snippet showing how to run our Customer Search query in JavaScript is shown.

4. Select all the text (**Ctrl-A**), then click **Copy to Clipboard**.

Add the Postman Sample Code to the Web App

Now that you have a working code sample for running a customer search you can paste that example code into the web application and modify it to wire up the search input and results display in the application.

1. Return to **Visual Studio Code** and open **scripts/app.js**

2. Locate the **Search** function located at about line **64**.
   The search function is located within the **methods** object. As discussed earlier any function placed within **methods** can be called from the web page by **Vue.js**. The starter code sample is set up to call the **search** method any time the user types something into the search input box.

3. In the Search method where it says **/// Place the code from POSTMAN Here** paste the code you copied to the clipboard.
   Your pasted code should look something like this. There are two main parts:

   - the settings variable which contains the URL, all of the rest headers, etc.

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://appserver01/ERP10/api/v1/Erp.Bo.CustomerSvc/Customers?%
24filter=startswith(CustID%20%2C'A')%20or%20startswith(Name%2C%20'A')&%24
orderby=Name",
  "method": "GET",
  "headers": {
    "content": "application/json",
    "accepts": "application/json",
    "authorization": "Basic ",
    "cache-control": "no-cache",
    "postman-token": "  "
  }
}
```

- The $.ajax call will run the REST request and write the results to the browser's JavaScript console when it returns.

```
$.ajax(settings).done(function (response) {
  console.log(response);
});
```

4. **Save** app.js
   After you have updated the search method you can run the application and verify that typing something into the search box runs our sample query. It won't yet show results in the user interface, but you can use the JavaScript debugger to verify that the rest call is sent and the response is written into the JavaScript console.

5. Return to your browser and verify the application is still running.

6. Press **F12** to open Developer tools in your browser and resize it so you can see all of the tabs at the top.
   You can use the developer tools to find issues in our web application, monitor the network calls it makes as it tries to run our REST request, and view errors and information messages in the console. For now, we want to look at the **Network** tab and the **Console**.

7. Click on the **Network** tab in Developer tools window.
   The network tab lets us see each network request made by the web page. That includes loading JavaScript files, images, and critically calls from the application to REST services.

8. Click on the **Console** tab in developer tools window.

9. Press the Refresh button on the browser to load the latest version.
   As you can see when you refresh the web page the network panel records each network request as it loads the files required by the application. Now you can try to type something into the application and see if the REST request is run in the Network panel.

10. In the web page type any character into the search box.
    If the application is working correctly two things should happen in Developer tools:

    - A new event should appear in the network panel showing the REST request being run. You can click on the name to view the REST request and inspect what was sent and what was returned.
    - Because in app.js the response method runs console.log(response) a new Object should appear in the JavaScript in Developer tools showing what was sent back from ERP.

      Note This is the code that runs and retrieves the results in app.js:

      ```
      $.ajax(settings).done(function (response) {
        console.log(response);
      });
      ```

Display the Search Results on the Web Page

You are now running a REST query when the user types a search, but the results are not shown on the page. You will need to modify the response function to take the results and bind them to a data element which you can use to show results on the web page.

1. Leave your browser running and return to Visual Studio Code.

2. Open **app.js**.

3. Click **Ctrl-F** to open the **Find** dialog and search for **data: {**
   Just like the appTitle we have created other variables in the data object to help show the search results:

   - **query** is automatically filled out with whatever query the user has typed in.
   - **searching** is set to true while the search is running and false when it's done. This is used to show and hide an animation while the search is in progress.
   - **searchResults** is an array that is bound to the search results table in the ui. If we add data into the array a table will appear in the ui to show the data.

4. Click **Ctrl-F** to open the **Find** dialog and search for **$.ajax**.

   When the $.ajax call is completed to our REST service it will then call the done() function. You need to modify this function to set the searchResults array to the data sent back from REST. To do that you need to make two changes.

   An important point about **Vue.js** is if you want to access any of the data variables such as data.searchResults within a **Vue** method that data element is accessible through the **this** variable. At the top of the method you can see us set **this.searching** = true which sets the **data.searching** value to true.

   In a callback, such as the $ajax().done() function we need to tell JQuery to pass a reference to the data object, **this**, so it is available inside of the done function using the bind(this) function.

5. Locate the last line of the $.ajax call which looks like **});**

6. Change that line to **}.bind(this));**
   Your code should now look like this:

```
$.ajax(settings).done(function (response) {
          console.log(response);
        }.bind(this));
```

7. After the console.log line add a line **this.searchResults = response.value;**
   This line will take the REST response data in response.value and assign it to the data.searchResults property. Vue.js will automatically display the results in a table in the UI:

```
$.ajax(settings).done(function (response) {
          console.log(response);
          this.searchResults = response.value;
        }.bind(this));
```

8. Add an additional line to the done method **this.searching = false**.

   Setting searching = false will simply hide the searching animation now that the search is complete.

   Your completed code should now look like this:

```
$.ajax(settings).done(function (response) {
          console.log(response);
     //assign the search results
```

```
          this.searchResults = response.value;
          this.searching = false;
        }.bind(this));
```

9. **Save** app.js and return to your browser window. Click **Refresh** to reload the application with your changes.

10. Type any character into the search input box.
    This time you should see a searching animation play and then a table with search results appear showing all customers that start with "A".

Wire the Search Query to the REST Query

You have made great progress, but what the user types is still ignored. The final step to complete our search application is to add the text the user types into the search box into the REST URL, so it performs the search they asked for.

1. Leave your browser running and return to Visual Studio Code.

2. Open **app.js**.

3. Click **Ctrl-F** to open the **Find** dialog and search for **var settings = {**
   The settings object, as discussed earlier, contains the URL that will be used when we make our REST call. All you need to do is modify the URL to include **this.query** which contains the search text the user typed in.

4. Edit the URL replaceing the **'A'** in each startswith function with '**"+ this.query "'**.

   Note You need to include both a single quote ' followed by a double quote " at the beginning and a double quote " followed by a single quote ' at the end.

   Your URL should now look like this:

   ```
   "url": "https://epicorsi/ERP10/api/v1/Erp.Bo.CustomerSvc/Customers?%24filte
   r=startswith(CustID%20%2C'"+ this.query +"')%20or%20startswith(Name%2C%20'"
   + this.query +"')&%24orderby=Name",
   ```

5. Return to your browser window and click **Refresh** to reload the application with your changes.

6. Type **epic** into the search input box.
   This time you should see a searching animation play and then a table with search results appear showing all customers that have either a CustID or Name that starts with "Epic".

Congratulations! You've built a fully functional Customer Search Application using ERP REST Services.

Now that you have a fully functional application you can see how easy it would be to continue to modify the REST search that is done. You could easily change out the URL with another rest service or call multiple rest services to search across multiple objects. You could change this view to a BAQ search, or an Order search by simply changing out the Service URL and which fields you want to search on. You can add $select, $orderby, $top, $skip, or other types of $filter options to change the results that are sent back. If time remains please feel free to try modifying the application further using the REST Services Help to find any service you want to try.

Calling REST with Azure AD token in C#

Topics in this section outline how to perform Rest calls in C# when Azure AD authentication is implemented.

Tip In Epicor ERP Application Help, you may download an example project using the link at the bottom of this section.

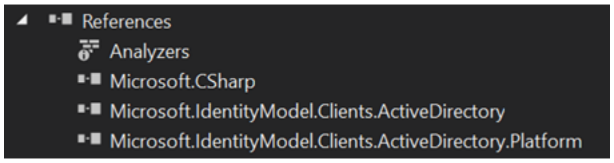
Create the Visual Studio Project

1. Launch **Microsoft® Visual Studio®**.
2. Select the **File > New > Project** menu item.
3. In the **New Project** dialog box, verify **Visual C#** is selected in the tree view.
4. Next in the **Detail** panel, select **Console Application**.
5. For the project **Name**, enter **AbcCodeServiceClientAzureAD**.
6. Select the location where you will save the project.
7. Click **OK**.

Add NuGet Package

First, add the Active Directory Authentication Library (ADAL) NuGet package to the project.

1. In the **Solution Explorer**, right-click the AbcCodeServiceClientAzureAD project item and select **Manage NuGet Packages...** menu item.
2. Click **Browse** and search for **Microsoft.IdentityModel.Clients.ActiveDirectory package**.
3. Click **Install** and step through the wizard.
4. Verify the following Microsoft.Identity references are added to your project.



```
References
    Analyzers
    Microsoft.CSharp
    Microsoft.IdentityModel.Clients.ActiveDirectory
    Microsoft.IdentityModel.Clients.ActiveDirectory.Platform
```

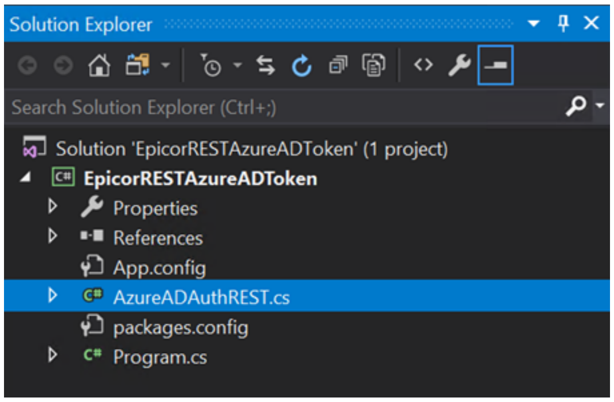
Add AzureADAuth Class

Within a project, implement AzureADAuth public class to handle Azure AD token creation and management.

1. In the **Solution Explorer**, right-click the **AbcCodeServiceClientAzureAD** project item.
2. Select the **Add > Class…** menu item.
   The **Add New Item** dialog box displays. On the **List** view, the **Class** item should be selected by default.

3. Next in the **Name** field, enter **AzureADAuthREST**.

4. Click **Add**.



5. Adjust the using statements to read the following:

```
using System;
using System.Globalization;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
```

6. First, add the following Azure AD settings and adjust them accordingly:

```
//Settings to be adjusted according your Azure AD installation

        /// <summary>
        /// Directory ID - take it from Azure Active Directory properties
        /// </summary>
        private const string DirectoryID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXX
XXXXX";
        /// <summary>
        /// Application ID of the Epicor Web application
        /// </summary>
        private const string WebAppID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX
XX";
        /// <summary>
        /// Application ID of the Epicor native application
        /// </summary>
        private const string NativeClientAppID = "XXXXXXXX-XXXX-XXXX-XXXX-X
XXXXXXXXXXX";
        /// <summary>
        /// Redirect URL specified for Epicor native application.
        /// </summary>
        private static readonly Uri redirectURL = new Uri("https://localhos
t");

        /// <summary>
        /// Azure AD logon URL, used for any tenancy.
        /// </summary>
        private const string aadInstance = "https://login.microsoftonline.c
om/{0}";
```

Tip For a complete information on the above properties, see either of the below sources:

- **Application Help**: System Management > Working With System Management > System Administration Guide > Manage Epicor ERP > Authentication (User Identity) > Azure AD Authentication
- **Epicor ERP 10 Installation guide**: Appendices > Azure AD Authentication

7. Add property to hold Authentication context instance. This will manage all necessary token processing internally, without an intervention:

```
private static AuthenticationContext _authContext;
        /// <summary>
        /// Property to store authentication context for the Azure AD token
s
        /// </summary>
        private static AuthenticationContext AuthContext
        {
            get
            {
                if (_authContext == null)
                {
                    var authority = string.Format(CultureInfo.InvariantCult
ure, aadInstance, DirectoryID);

                    _authContext = new AuthenticationContext(authority);
                }
                return _authContext;
            }
        }
```

8. Add function to get the token. You may use wither of the below approaches:

a. Show ADAL standard window to ask for name and password, when no valid token exists:

```
/// <summary>
        /// Get token for the user
        /// </summary>
        /// <returns>Access token for the WCF header</returns>
        public static string GetToken()
        {
            if (string.IsNullOrEmpty(NativeClientAppID))
                throw new Exception("No settings specified in AzureADAuth
");

            //Specify that dialog should be shown only if prompt for user
 credentials is necessary
            var platformParameters = new PlatformParameters(PromptBehavio
r.Auto);

            //use AuthContext to get token. ADAL will internally handle t
oken caching and refreshing
            AuthenticationResult result = AuthContext.AcquireTokenAsync(
                WebAppID,
                NativeClientAppID,
                RedirectURL,
                platformParameters
            ).Result;

            //Return AccessToken, which should be sent to Epicor App Serv
er as WCF header
            return result?.AccessToken;
       }
```

b. Specify Azure AD user name and password and obtain a token the user. This approach can be used to avoid user interaction:

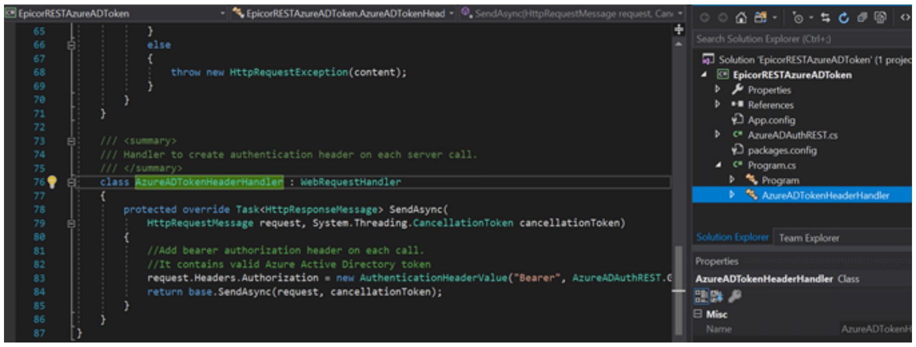
```
/// <summary>
        /// Gets token for current user
        /// </summary>
        /// <returns>Access token for the WCF header</returns>
        public static string GetToken()
        {
            if (string.IsNullOrEmpty(NativeClientAppID))
                throw new Exception("No settings specified in AzureADAuth
");

            //get username and password from settings. Consent should be
already received for this user in the applications
            const string userName = "USER@tenant.onmicrosoft.com";
            const string password = "Password";

            var userCredentials = new UserPasswordCredential(userName, pa
ssword);

            //use AuthContext to get token for this user.  ADAL will inte
rnally handle token caching and refreshing
            var result = AuthContext.AcquireTokenAsync(WebAppID, NativeCl
ientAppID, userCredentials).Result;

            //Return AccessToken, which should be sent to Epicor App Serv
er as WCF header
            return result?.AccessToken;
}
```

Note The first time after applications are created, user logon must be done interactively, because user consent should be received. If this was not done, then the example above will fail with the error: The user or administrator has not consented to use the application.

Add Token Header to the Server Request

In **Program.cs**, create a class that will add Authorization header with Azure AD token to every call.

```
/// <summary>
    /// Handler to create authentication header on each server call.
    /// </summary>
    class AzureADTokenHeaderHandler : WebRequestHandler
    {
        protected override Task<HttpResponseMessage> SendAsync(
            HttpRequestMessage request, System.Threading.CancellationToken canc
ellationToken)
        {
            //Add bearer authorization header on each call.
            //It contains valid Azure Active Directory token
            request.Headers.Authorization = new AuthenticationHeaderValue("Bear
er", AzureADAuthREST.GetToken());
            return base.SendAsync(request, cancellationToken);
        }
    }
```



EpicorRESTAzureADToken
EpicorRESTAzureADToken.AzureADTokenHead
SendAsync(HttpRequestMessage request, Can
Search Solution Explorer (Ctrl+;)
Solution 'EpicorRESTAzureADToken' (1 projec
EpicorRESTAzureADToken
Properties
References
App.config
AzureADAuthREST.cs
packages.config
Program.cs
Program
AzureADTokenHeaderHandler
Solution Explorer
Team Explorer
Properties
AzureADTokenHeaderHandler Class
Misc
Name
AzureADTokenH

```
                }
                else
                {
                    throw new HttpRequestException(content);
                }
            }
        }

        /// <summary>
        /// Handler to create authentication header on each server call.
        /// </summary>
        class AzureADTokenHeaderHandler : WebRequestHandler
        {
            protected override Task<HttpResponseMessage> SendAsync(
                HttpRequestMessage request, System.Threading.CancellationToken cancellationToken)
            {
                //Add bearer authorization header on each call.
                //It contains valid Azure Active Directory token
                request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", AzureADAuthREST.G
                return base.SendAsync(request, cancellationToken);
            }
        }
    }
```

Write Code to Call REST API

1. In the Main function define `restUserCallUrl` as:

```
private static string  restUserCallUrl = "https://<SERVER>/<AppServerName>/
api/v1/Ice.BO.UserFileSvc/GetUserFile";
```

2. Then, create HttpClient instance, that will use AzureADTokenHeaderHandler class as parameter. After that, you can use this client to make calls with HttpRequestMessage & HttpRequestMessage classes.

```
using (var client = new HttpClient(new AzureADTokenHeaderHandler()))
{
  //Create request to make REST call
                HttpRequestMessage request = new HttpRequestMessage(HttpMet
hod.Post, restUserCallUrl);

                Console.WriteLine($"Calling POST {restUserCallUrl}");
                //Get response
                HttpResponseMessage response = client.SendAsync(request).Re
sult;
}
```

Calling REST with Azure AD token in JavaScript

Use the Active Directory Authentication Library for JavaScript (ADAL JS) to handle Azure AD authentication in your single page applications. This library works with both plain JS as well as AngularJS applications.

For a complete information on usage of this library, see the following GitHub page: https://github.com/AzureAD/azure-activedirectory-library-for-js.

Configuration settings can be set as follows:

- tenant - Directory ID of your Azure Active Directory.
- clientId - Application ID of your Web App application. (NativeClientAppID is not used in JavaScript scenario)

Example The below is the configuration example from the ADAL.Js documentation:

```
adalAuthenticationServiceProvider.init(
        {
            // Directory ID - take it from Azure Active Directory properties
            tenant: "52d4b072-9470-49fb-8721-bc3a1c9912a1", // Optional by defa
ult, it sends common
     // Application ID of the Epicor Web application
            clientId: "e9a5a8b6-8af7-4719-9821-0deef255f68e" // Required
        },
        $httpProvider   // pass http provider to inject request interceptor to
attach tokens
        );
```

When authentication provider is set up, then tokens can be used in Epicor REST API calls. For example, for JQuery:

```
// Get Epicor user Data
                    $.ajax({
                        type: "POST",
                        url: "/Ice3/api/v1/Ice.Bo.UserFileSvc/GetUserFile/",
                        headers: {
                            'Authorization': 'Bearer ' + token,
                        },
                    }).done(function(data) {

                        Alert(" UserID : " +
                            data.returnObj.UserFile[0].UserID +
                            "\nExternalIdentity : " +
                            data.returnObj.UserFile[0].ExternalIdentity);

                    }).fail(function() {
                        alert("Call failed");
                    });
```

For XMLHttpRequest, the syntax may look as follows:

```
var getCurrentUser = function (access_token) {
            document.getElementById('api_response').textContent = 'Calling API.
..';
            var xhr = new XMLHttpRequest();
            xhr.open('POST', '/Ice3/api/v1/Ice.Bo.UserFileSvc/GetUserFile/', tr
ue);
            xhr.setRequestHeader('Authorization', 'Bearer ' + access_token);
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    // Do something with the response
                    document.getElementById('api_response').textContent =
                        JSON.stringify(JSON.parse(xhr.responseText), null, '  '
);
                } else {
                    // TODO: Do something with the error
                    document.getElementById('api_response').textContent =
                        'ERROR:\n\n' + xhr.responseText;
                }
            };
            xhr.send();
        }
```

Note Exact URL of a logon page where authentication happens must be specified in the

Reply URLs setting of the Web application in the Azure Active Directory. Otherwise, the following error will be shown when a user is trying to logon: `AADSTS50011: The reply address '<ApplicationURL>' does not match the reply addresses configured for the application: '<Web Application ID>'.`

Tip For more examples on for ADAL.JS usage, see the following sources:

- https://github.com/Azure-Samples/active-directory-javascript-singlepageapp-dotnet-webapi
- https://github.com/Azure-Samples/active-directory-angularjs-singlepageapp

WCF Data Services Client Library

The following example illustrates how you link client code using a WCF Data Services Client library.

1. Launch **Microsoft Visual Studio**.

2. Create a console application.

3. Now add a **Service Reference** to the service. To do this, right-click the services; from the context menu, select **References > Add Service Reference**.

4. Enter the **Service Address**. For example:
   `https://appserver01/2017R/api/v1/Erp.Bo.SalesOrderSvc/`

5. Next enter your **User Account** credentials.

6. When the **Service Reference** window displays, click **OK**.

7. The proxy code you need generates. The following is an example of service call code:

```
static void Main(string[] args)
        {
            //create service instance
            var svcUri = new Uri("https://localhost/MyEpicorInstance/Server
/Erp/BO/Erp.Bo.SalesOrderSvc/");
            var svc = new SalesOrderSvc.Erp.RestApi(svcUri);
            //specify creadentials
            var serviceCreds = new NetworkCredential("manager", "Epicor123"
);
            var cache = new CredentialCache {{svcUri, "Basic", serviceCreds
}};
            svc.Credentials = cache;

            //list first 3 orders and count their orderlines
            foreach (var so in svc.SalesOrders.Expand(so =>so.OrderDtls).Ta
ke(3))
            {
                Console.WriteLine("Order nume: {0}  customer: {1}, line cou
nt {2}",so.OrderNum, so.CustNum, so.OrderDtls.Count);
            }
            //create new order
            var newSo = new SalesOrder()
            {
                Company="EPIC06",
                OrderNum = 0,
                CustNum = 2,
                BTCustNum = 2,
                ShipToCustNum = 2,
                TermsCode =  "2/10"

            };

            svc.AddToSalesOrders(newSo);
            var response = svc.SaveChanges(SaveChangesOptions.PatchOnUpdate
);

        }
```

SaveChanges Error

You may receive an error message when you call the **SaveChanges** method.

This message states you need version 3.0, but the **MaxProtocolVersion** for the data service is set to an earlier version. You need to set the MaxProtocolVersion to the higher version. To do this:

1. Open the **Reference.cs** file.

2. Locate the following call:

```
public RestApi(global::System.Uri serviceRoot)
     :base(serviceRoot)
```

3. Replace it with the following code:

```
public RestApi(global::System.Uri serviceRoot) :
      base(serviceRoot, DataServiceProtocolVersion.V3)
```

Error Handling

The Web.API uses the <system.web><customErrors> mode to control how errors display in client installations.

This mode value is case-sensitive. If you do not specify a value, the system uses the <customErrors mode="RemoteOnly" /> setting. Through this setting, the local users see all error information while the remote users see a brief, generic message that states they should contact their system administrator.

If you want to display the brief, generic error message for both remote users and client users, set the option to this value:

```
<system.web>
     <customErrors mode="On" />
</system.web>
```

However if you wish to display the complete ASP.NET error message to both remote users and client users, set the option to this value:

```
<system.web>
     <customErrors mode="Off" />
</system.web>
```

Tip To learn more about how to use the customErrors setting, review the following Microsoft topic: https://msdn.microsoft.com/en-us/library/h0hfz6fc(v=vs.100).aspx

Data Security

You secure the data that comes from REST by implementing authentication codes.

This section describes the code you enter to secure the data you display through REST services. You then secure the streaming data from malicious activity.

You should always use the HTTPS protocols with REST services. These protocols are a secure way for REST to transfer data between the Epicor application and your third party application. REST supports two authentication methods:

- **Basic Authentication** – The User Name and Password are combined into a string, separated by a colon, and encoded by using Base64. The resulting data is placed in an Authorization header. It uses both Authorization: Basic and an encoded string.
- **Token Authentication** – You first get the bearer token for the user from the token service. Then send this access token in the Authentication header using this format: **Authorization: Bearer** (token string).

Basic Authentication

This Basic Authentication code example specifies the User Name and Password in the call header. You can also add optional call settings to specify the Company and Plant.

```
static HttpClient CreateClient()
        {
            HttpClient client = new HttpClient();

            client.DefaultRequestHeaders.Authorization = new AuthenticationHead
erValue("Basic",
                Convert.ToBase64String(
ASCIIEncoding.ASCII.GetBytes(string.Format("{0}:{1}", "manager", "Epicor123")))
);

            //header to set current company
            ICallHeader hdr = new CallSettings("EPIC06", "MfgSys", "", "");
            client.DefaultRequestHeaders.Add(hdr.Name, JsonConvert.SerializeObj
ect(hdr));

            return client;
        }
```

To use this authentication code:

```
HttpClient client = CreateClient();
HttpResponseMessage response = client.GetAsync(ServiceUrl + "/Ice.Bo.ProcessTas
kSvc/ProcessTasks").Result;
```

Token Authentication

This Token Authentication code example gets the token from the token resource.

```
private static string ObtainToken()
        {
            HttpClient client = new HttpClient();
            // Add an Accept header for JSON format.
            client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHea
derValue("application/json"));
            try
            {
                const string TokenCreateURL = "https://MyServerInstance.epicor.
net/MyEpicorInstance/TokenResource.svc/";
                var request = new HttpRequestMessage()
                {
                    RequestUri = new Uri(TokenCreateURL),
                    Method = HttpMethod.Post,
                };
                //send user credential
                request.Headers.Add("username", "manager");
                request.Headers.Add("password", "Epicor123");

                // Get response
                HttpResponseMessage response = client.SendAsync(request).Result
;

                if (response.IsSuccessStatusCode)
                {
                    //Parse the response body.
                    var tokenObj = response.Content.ReadAsAsync<TokenObject>().
Result;
                    //Get access token value
                    return tokenObj.AccessToken;
                }
                else
                {
                    Console.WriteLine("{0} ({1})", (int) response.StatusCode, r
esponse.ReasonPhrase);
                }

            }
            catch (AggregateException ex)
            {
                Console.WriteLine(ex.Message);

            }
            return null;
        }
```

You can now use this token to authenticate calls:

```
//get token from token resource
          var accessToken = ObtainToken();

            if (!string.IsNullOrEmpty(accessToken))
            {
                HttpClient  getTipCall = new HttpClient();
                //put token into Authentication header
                getTipCall.DefaultRequestHeaders.Authorization = new Authentica
tionHeaderValue("Bearer", accessToken);

                string tipUrl = "https://localhost/MyEpicorInstance/Server/Ice/
BO/Ice.Bo.TipsSvc/Tips";

                HttpResponseMessage response = getTipCall.GetAsync(tipUrl).Resu
lt;

                if (response.IsSuccessStatusCode)
                {
                    var tipObj = response.Content.ReadAsAsync<JObject>().Result
;

                    foreach (var p in tipObj)
                    {
                        Console.WriteLine("{0} ({1})", p.Key, p.Value);
                    }

                }
                else
                {
                    Console.WriteLine("{0} ({1})", (int)response.StatusCode, re
sponse.ReasonPhrase);
                }

            }
```

Single Sign-On

To keep the REST transfer secure while making it easier for users to access the data, implement the Single Sign-On feature.

You do this by activating Windows Authentication within your system. To activate Windows Authentication:

1. Access your server machine.
2. Launch **Internet Information Services (IIS) Manager**.
3. Now for the virtual folder, activate the **Authentication** folder.
4. Enable **Windows Authentication**.

This user can now connect to REST services without entering credentials. The system uses the header to verify the account is secure and permitted to access the service.

Note that when you authenticate Windows in the Internet Information Services (IIS) Manager, the TokenResource.svc and other resources that use webHttpBinding (Web.svc, ECC.svc, and so on) cannot start. If you need both token authentication and Windows Authentication to work on the application server, change the webHttpBinding definition in the web.config file. Change the **<transport clientCredentialType>** setting to use Windows; by default this setting is set to **None**.

```
<webHttpBinding>
        <binding name="RestHttps" transferMode="Buffered">
          <security mode="Transport">
            <transport clientCredentialType="Windows" />
          </security>
          <readerQuotas maxArrayLength="2147483647" maxBytesPerRead="2147483647
" maxDepth="2147483647" maxStringContentLength="2147483647" />
```

```
        </binding>
      </webHttpBinding>
```

Azure Active Directory Token Authentication

Azure Active Directory token should be sent to Epicor REST API the same way as internal Epicor token by adding Authorization header and set its value to Bearer <Access token value>.

The following examples illustrate how to retrieve authentication token:

- Show ADAL standard window to ask for name and password, when no valid token exists:

```
/// <summary>
        /// Get token for the user
        /// </summary>
        /// <returns>Access token for the WCF header</returns>
        public static string GetToken()
        {
            if (string.IsNullOrEmpty(NativeClientAppID))
                throw new Exception("No settings specified in AzureADAuth");

            //Specify that dialog should be shown only if prompt for user cre
dentials is necessary
            var platformParameters = new PlatformParameters(PromptBehavior.Au
to);

            //use AuthContext to get token. ADAL will internally handle token
 caching and refreshing
            AuthenticationResult result = AuthContext.AcquireTokenAsync(
                WebAppID,
                NativeClientAppID,
                RedirectURL,
                platformParameters
            ).Result;

            //Return AccessToken, which should be sent to Epicor App Server a
s WCF header
            return result?.AccessToken;
       }
```

- Specify Azure AD user name and password and obtain a token the user. This approach can be used to avoid user interaction:

```
/// <summary>
        /// Gets token for current user
        /// </summary>
        /// <returns>Access token for the WCF header</returns>
        public static string GetToken()
        {
            if (string.IsNullOrEmpty(NativeClientAppID))
                throw new Exception("No settings specified in AzureADAuth");

            //get username and password from settings. Consent should be alre
ady received for this user in the applications
            const string userName = "USER@tenant.onmicrosoft.com";
            const string password = "Password";

            var userCredentials = new UserPasswordCredential(userName, passwo
rd);

            //use AuthContext to get token for this user.  ADAL will internal
ly handle token caching and refreshing
            var result = AuthContext.AcquireTokenAsync(WebAppID, NativeClient
AppID, userCredentials).Result;

            //Return AccessToken, which should be sent to Epicor App Server a
s WCF header
            return result?.AccessToken;
}
```

CORS Support

Through Cross-Origin Resource Sharing (CORS), restricted web resources from one domain can be requested from another outside domain.

By default, cross-domain requests are not allowed because they can perform **POST**, **PUT**, **DELETE**, and other advanced HTTP requests which should be restricted to authorized clients only.

However through the CORS restriction configuration, you can authorize specific trusted hosts to access the REST services. For example, if you create a web application that uses the REST services hosted on its own server, you need to enable a CORS exception for that server to access the services.

Important While it is possible to disable all CORS restrictions and allow any host to access the services, you should not disable these restrictions. Instead you should enable access for the specific hosts that need access to the service.

Currently only limited CORS functionality is available with your Epicor application. You can set up the web.config to use the CorsOrgings application settings to specify the origin of the requests. Available host options:

- Define a specific host
- Define a comma-delimited list of hosts

Examples:

- `<add key="CorsOrigins" value="https://localhost" />` - CORS only allows requests from only https://localhost .
- `<add key="CorsOrigins" value="https://localhost, https://localhost" />` - CORS only allows access from https://localhost and https://localhost .

Index

N

nuget package, example #3 47

V

visual studio project, example #3 47

W

wcf services example #3 47